Finite Automata for Efficient Graph Recognition

Frank Drewes
Umeå universitet
SE-90187 Umeå
Sweden
drewes@cs.umu.se

Berthold Hoffmann
Universität Bremen
D-28334 Bremen
Germany
hof@uni-bremen.de

Mark Minas
Universität der Bundeswehr München
D-85577 Neubiberg
Germany
mark.minas@unibw.de

Engelfriet and Vereijken have shown that linear graph grammars based on hyperedge replacement generate graph languages that can be considered as interpretations of regular string languages over typed symbols. In this paper we show that finite automata can be lifted from strings to graphs within the same framework. For the efficient recognition of graphs with these automata, we make them deterministic by a modified powerset construction, and state sufficient conditions under which deterministic finite graph automata recognize graphs without the need to use backtracking.

1 Introduction

Engelfriet and Vereijken [17] have shown that linear graph grammars based on hyperedge replacement can be considered as interpretations of linear string grammars: typed ("doubly ranked") symbols of an alphabet are interpreted as basic graphs that have front and rear interfaces of nodes, and string concatenation is interpreted as the composition of two graphs by gluing the rear of the first to the front of the second graph. Graph languages constructed in this way are of bounded pathwidth, and are thus potentially more efficiently recognizable than general hyperedge replacement languages, which are known to be NP-complete. However, without additional restrictions even these graph languages are NP-complete [1].

In this paper we study how finite automata over graph symbols can be interpreted to recognize graph languages efficiently. Given a graph as input, the transitions of such an automaton consume the graph step by step while changing the state of the automaton, in the end reaching either an accepting or a rejecting state. If the automaton used is nondeterministic, a naive decision procedure for determining whether the input graph is accepted would have to use backtracking. While we show in this paper that the transition relation can be made deterministic by a modified powerset construction, backtracking may still be needed. This is due to the fact that, whereas a string starts with a unique first symbol to be read by the first transition of the computation of a deterministic finite automaton, in general several alternative basic graphs represented by that symbol may be spelled off at the front of a graph. We provide two sufficient criteria under which automata can choose between them without the need to backtrack. These criteria resemble similar criteria known from efficient parsing algorithms for context-free hyperedge replacement languages [13, 14].

Work on efficient parsing algorithms for grammars started in the late 1980s, initiated by the realization that, in general, the graph languages generated by these grammars can be NP-complete [1, 28]. Early polynomial algorithms were based on restrictions which either ensure efficiency of the well-known Cocke-Younger-Kasami algorithm (adapted to hyperedge replacement grammars) or make sure that the derivation trees of generated graphs mirror a unique

J. Endrullis, D. Grzelak, T. Heindel and J. Kosiol (Eds):
GCM 2023 and GCM 2024
EPTCS 417, 2025, pp. 134–156, doi:10.4204/EPTCS.417.8

© F. Drewes, B. Hoffmann, M. Minas
This work is licensed under the
Creative Commons Attribution License.

recursive decomposition of graphs into smaller and smaller subgraphs, see [29, 31, 10]. Later work on parsing hyperedge replacement languages include [8, 21] as well as the authors' own work on top-down and bottom-up parsers for these languages; see, e.g., [11, 12, 25, 14, 15]. For a more extensive overview of work on efficient parsing for graph grammars, including other types of grammars than those based on hyperedge replacement, see [14, Section 10].

For the efficient recognition of graph languages, finite automata have not attained the importance that they have for string languages. Some early work exists on finite automata for algebraic structures [2, 22], rooted directed acyclic graphs [32], and infinite directed acyclic graphs [27]. Brandenburg and Skodinis have devised finite automata for linear node replacement grammars [6]. Bozapalidis and Kalampakas have studied automata on the hypergraphs of Engelfriet and Vereijken [17] in an algebraic setting [5, 26]. And, Brugging, König *et al.* have recast these hypergraphs as cospans [3], and have shown that finite automata defining hypergraph languages are equivalent to Courcelle's recognizable hypergraph languages [9]; later these results have been generalized to hereditary pushout categories [7]. All this work has focussed on *defining* the graph languages accepted by finite automata (e.g., by composing the graphs labeling the edges passed on a walk through the transition diagrams of the automata), rather then on *recognizing* graph languages in the sense of efficiently deciding their membership problem. In particular, the efficiency of the recognition process has not been a matter of concern. In this paper, we focus on this aspect, which is of interest because the analysis of graphs by finite automata corresponds intuitively to a lexical analysis in the traditional string-based setting.

The technical contributions of this paper revolve mainly around different aspects of coping with nondeterminism, which differs in subtle ways from the string case. Given a nondeterministic automaton over graph symbols, we first apply a powerset construction to it (Section 4.1). Interestingly, this construction does not always result in a deterministic automaton. We show, however, that the resulting automaton is indeed deterministic if the construction starts from an unambiguous automaton, a condition that can always be fulfilled (Section 4.2). Next, as mentioned above, even a deterministic automaton is only deterministic at the string level. As the edges of a graph do not come in a predefined processing order, recognition still requires backtracking in general: anytime in the process, several different transitions may be applicable, and a given transition may have several alternative edges it may consume. We deal with these problems in Sections 4.3 and 4.4, resulting in linear-time recognition without backtracking.

The remainder of this paper is structured as follows. We define graphs with front and rear interfaces and their composition in Section 2. In Section 3, we introduce finite automata over graph symbols and define how they can be used to recognize graph languages. Section 4 contains the main technical contributions, as described above. In Section 5, we conclude the paper and indicate directions of future research.

2 Graphs and Graph Composition

We let $\mathbb{N}$ denote the set of non-negative integers, and $[n]$ the set $\{1,\dots,n\}$ for all $n \in \mathbb{N}$. A^* denotes the set of all finite sequences over a set A; the empty sequence is denoted by ε, $A^* \setminus \{\varepsilon\}$ by A^+, and the length of a sequence α by $|\alpha|$. For a sequence $s = a_1 \cdots a_n$ with $n \in \mathbb{N}$ and $a_1,\dots,a_n \in A$, we let $[s] = \{a_1,\dots,a_n\}$ and $s(i) = a_i$ for all $n \in \mathbb{N}$. For $f\colon A \to B^*$, we let $f(a,i) = f(a)(i)$ if $a \in A$ and $i \in [|f(a)|]$. We silently extend functions to sequences and let $f(a_1 \cdots a_n) = f(a_1) \cdots f(a_n)$, for all $n \in \mathbb{N}$, $a_1,\dots,a_n \in A$, and all functions $f\colon A \to B$. As usual,

given a binary relation $R \subseteq A \times B$, we denote the transitive closure by R^+ and the transitive reflexive closure by R^*. Further, $dom(R)$ denotes the domain of R, i.e., $dom(R) = \{a \in A \mid \exists b \in B\colon (a,b) \in R\}$. Finally, we let $R(a) = \{b \in B \mid (a,b) \in R\}$ for every $a \in A$.

We consider edge-labeled hypergraphs (which we simply call graphs). Like Habel in [23], we supply them with a front and a rear interface, each being a sequence of nodes.[1]

For the labeling of edges, we consider a ranked alphabet $(\Sigma, rank)$ consisting of a set of symbols and a function $rank\colon \Sigma \to \mathbb{N}$ which assigns a rank to each symbol $a \in \Sigma$. The pair $(\Sigma, rank)$ is usually identified with Σ, keeping $rank$ implicit.

Definition 2.1 (Graph)**.** A *graph* (over Σ) is a tuple $G = (\dot{G}, \bar{G}, att_G, lab_G, front_G, rear_G)$, where $\dot{G}$ and $\bar{G}$ are disjoint finite sets of *nodes* and *edges*, respectively, the function $att_G\colon \bar{G} \to \dot{G}^*$ attaches repetition-free sequences of nodes to edges, the function $lab_G\colon G \to \Sigma$ labels edges with symbols in such a way that $|att_G(e)| = rank(lab_G(e))$ for every edge $e \in \bar{G}$, and the repetition-free node sequences $front_G, rear_G \in \dot{G}^*$ specify the *front* and *rear* interface nodes.

The lengths of front and rear interfaces classify graphs as follows: A graph G *has type* (i,j) if $|front_G| = i$ and $|rear_G| = j$; we then write $type(G) = (i,j)$. $\mathbb{G}_\Sigma^{(i,j)}$ denotes the set of all graphs of type (i,j), and $\mathbb{G}_\Sigma = \bigcup_{i,j \in \mathbb{N}} \mathbb{G}_\Sigma^{(i,j)}$ denotes the set of all graphs over Σ, regardless of type.

For graphs G and H, a *morphism* $m\colon G \to H$ is a pair $m = (\dot{m}, \bar{m})$ of functions $\dot{m}\colon \dot{G} \to \dot{H}$ and $\bar{m}\colon \bar{G} \to \bar{H}$ that preserve attachments and labels, i.e., $att_H(\bar{m}(e)) = \dot{m}(att_G(e))$ and $lab_H(\bar{m}(e)) = lab_G(e)$ for all $e \in \bar{G}$. (Note that fronts and rears need not be preserved.) The morphism m is *injective* or *surjective* if both $\dot{m}$ and $\bar{m}$ have this property, and a *subgraph inclusion* of G in H if $m(x) = x$ for every node and edge x in G; then we write $G \subseteq H$. G and H are *isomorphic*, $G \cong H$, if there exists a surjective and injective morphism $m\colon G \to H$ such that $front_H = m(front_G)$ and $rear_H = m(rear_G)$.

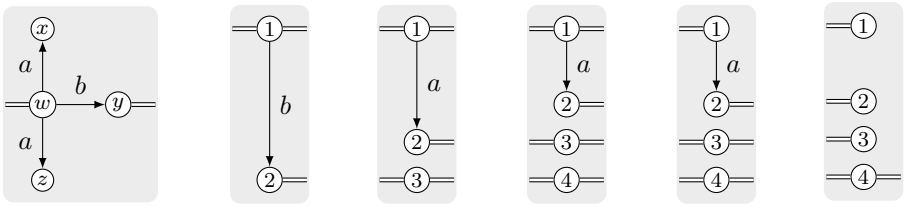
Consider $a \in \Sigma$, $n \in \mathbb{N}$, and repetition-free sequences $\varphi, \varrho \in [n]^*$ such that $[\varphi] \cup [rank(a)] = [n]$ (where $[\varphi]$ and $[rank(a)]$ are not necessarily disjoint). Then $\langle a \rangle_\varrho^\varphi$ denotes the graph A with $\dot{A} = [n]$, $\bar{A} = \{e\}$, $lab_A(e) = a$, $att_A(e) = 1 \cdots rank(a)$, $front_A = \varphi$, and $rear_A = \varrho$, whereas $\langle \varepsilon \rangle_\varrho^{(n)}$ denotes the discrete graph B with $\bar{B} = \varnothing$, $\dot{B} = [n]$, $front_B = 1 \cdots n$, and $rear_A = \varrho$. We call $\langle a \rangle_\varrho^\varphi$ an *atom* and $\langle \varepsilon \rangle_\varrho^{(n)}$ a *blank*.

Note that $\langle \varepsilon \rangle_\varepsilon^{(0)}$ is the empty graph. Further note that no atom has any node that neither is a front node, nor is it attached to its only edge. In particular, every rear node is also a front node or attached to the unique edge of the atom, or both. Moreover, all nodes of a blank occur in its front interface. Finally recall that our objective is efficient recognition of graphs, which tries to compose an input graph from a sequence of atoms and a blank. The requirements on front and rear interfaces differ because recognition will process graphs from front to rear.

Example 2.1. Figure 1 shows a "star" graph, atoms $\langle b \rangle_{12}^1$, $\langle a \rangle_{123}^{13}$, $\langle a \rangle_{1234}^{134}$, and $\langle a \rangle_{234}^{134}$ using the symbols a and b, both of rank 2, and the blank $\langle \varepsilon \rangle_4^{(4)}$. As usual, we represent nodes by circles and binary edges by arrows. Front interface nodes are connected with double lines to the left border of the graph, rear interface nodes with double lines to its right border. Front and rear nodes are ordered from top to bottom; the "star" graph, e.g., has the front interface w and the rear interface y.

We follow Engelfriet and Vereijken [17] in defining the composition of graphs in a way resembling string concatenation.

[1] Other than in [23], however, we do not divide the attached nodes of edges into sources and targets.



"Star" graph $\langle b\rangle_{12}^{1}$ $\langle a\rangle_{123}^{13}$ $\langle a\rangle_{1234}^{134}$ $\langle a\rangle_{234}^{134}$ $\langle \varepsilon\rangle_{4}^{(4)}$

Figure 1: A "star" graph, four atoms, and a blank.

Definition 2.2 (Graph Composition)**.** Let $G \in \mathbb{G}_\Sigma^{(i,k)}$ and $H \in \mathbb{G}_\Sigma^{(k,j)}$. We assume for simplicity that $\mathit{rear}_G = \mathit{front}_H$, $\dot{G} \cap \dot{H} = [\mathit{rear}_G] = [\mathit{front}_H]$, and $\bar{G} \cap \bar{H} = \varnothing$ (otherwise an appropriate isomorphic copy of G or H is used). The (*typed*) *composition* $G \odot H$ of G and H is the graph C such that $\dot{C} = \dot{G} \cup \dot{H}$, $\bar{C} = \bar{G} \cup \bar{H}$, $\mathit{att}_C = \mathit{att}_G \cup \mathit{att}_H$, $\mathit{lab}_C = \mathit{lab}_G \cup \mathit{lab}_H$, $\mathit{front}_C = \mathit{front}_G$, and $\mathit{rear}_C = \mathit{rear}_H$. Thus $C \in \mathbb{G}_\Sigma^{(i,j)}$.

Note that the composition $G \odot H$ is defined on concrete graphs if the assumptions in Definition 2.2 are satisfied, but is only defined up to isomorphism if an isomorphic copy of G or H needs to be taken. To avoid unnecessary technicalities, we shall assume that these assumptions are indeed satisfied whenever convenient.

Graphs can be constructed from a finite set of basic graphs using composition and disjoint union [18, 4] (where the latter concatenates the fronts and rears of the two graphs involved). If we just use composition, we need finitely many basic graphs per pair of front and rear interfaces. Instead of the simpler "atomic graphs" proposed in [17] and by Blume et al. in [3], we use atoms and blanks here.

Lemma 2.3. *Every graph G whose isolated nodes all occur in its front interface is of the form $A_1 \odot \cdots \odot A_n \odot B$ where $n = |\bar{G}|$, $A_1, \ldots, A_n$ are atoms, and B is a blank.*

Proof. Let $G \in \mathbb{G}_\Sigma$ with $|\bar{G}| = n$ such that every isolated node of G occurs in $[\mathit{front}_G]$. We prove by induction on n that there are atoms $A_1, \ldots, A_n$ and a blank B such that $G \cong A_1 \odot \cdots \odot A_n \odot B$. If $n = 0$, all nodes of G are isolated, and hence $\dot{G} = [\mathit{front}_G]$. Then $G \cong \langle \varepsilon \rangle_\varrho^{(m)}$ where $m = |\dot{G}|$ and ϱ is a sequence of length $k = |\mathit{rear}_G|$ such that $\mathit{rear}_G(i) = \mathit{front}_G(\varrho(i))$ for all $i \in [k]$.

Now consider $n > 0$ and assume, as an induction hypothesis, that the proposition holds for all graphs with $n-1$ edges. Pick any edge of G, say $e \in \bar{G}$, and let G' be the graph obtained from G by removing e and setting $\mathit{front}_{G'}$ to any permutation of $[\mathit{front}_G] \cup [\mathit{att}_G(e)]$. Since G' has $n-1$ edges, there are $n-1$ atoms $A_2, \ldots, A_n$ and a blank B such that $G' \cong A_2 \odot \cdots \odot A_n \odot B$ by the induction hypothesis. Now let $m = |\mathit{front}_G|$, $k = |\mathit{front}_{G'}|$, and $A_1 = \langle \mathit{lab}_G(e) \rangle_\varrho^\varphi$ where ϱ is a permutation of $[k]$, and φ is a sequence of length m such that the following holds: $\mathit{front}_{G'}(i) = \mathit{att}_G(e, \varrho(i))$ for all $i \in [k]$ with $\mathit{front}_{G'}(i) \in [\mathit{att}_G(e)]$. Moreover, $\varphi(i) = \varrho(j)$ if $\mathit{front}_G(i) = \mathit{front}_{G'}(j)$, for all $i \in [m]$ and $j \in [k]$. It is easy to see that $G \cong A_1 \odot G' \cong A_1 \odot \cdots \odot A_n \odot B$. □

Note that the size of interfaces required to build G according to Lemma 2.3 depends on G. Hence, the lemma does not contradict the fact, mentioned in Section 1, that the set of all graphs composed from a given finite set of atoms and blanks is of bounded pathwidth.

Example 2.2. Following the construction in the proof, the "star" graph G in Figure 1 can be composed by $G \cong \langle b\rangle_{12}^{1} \odot \langle a\rangle_{123}^{13} \odot \langle a\rangle_{1234}^{134} \odot \langle \varepsilon\rangle_{4}^{(4)}$, all shown in Figure 1. However, this is not the only composition. A much simpler one is $G \cong \langle a\rangle_{1}^{1} \odot \langle a\rangle_{1}^{1} \odot \langle b\rangle_{2}^{1}$.

The example shows that, unlike a string, a graph can be composed from atoms and blanks in different ways. Moreover, the size of atoms and blanks can also vary substantially.

3 Finite Automata over Graph Symbols

We now follow the idea of Engelfriet and Vereijken [17] to make use of the close resemblance of graphs under (typed) composition and strings under concatenation, and denote graphs by strings. Each symbol is interpreted as a graph, and thus a string translates into a graph, provided that the types of composed graphs fit. To this end, we type each symbol by a pair (i,j) – the type of the graph it will represent.

A *typed alphabet* is a pair $(\mathbf{\Theta}, \mathit{type})$ consisting of a (possibly infinite) set $\mathbf{\Theta}$ of symbols and a function $\mathit{type}\colon \mathbf{\Theta} \to \mathbb{N}\times\mathbb{N}$ which assigns a pair $\mathit{type}(a)$ of front and rear ranks to each symbol $a \in \mathbf{\Theta}$. The pair $(\mathbf{\Theta}, \mathit{type})$ is usually identified with $\mathbf{\Theta}$, and type is kept implicit.

A string $w = a_1 \cdots a_n \in \mathbf{\Theta}^+$ is *typed* if there are $k_0, \ldots, k_n \in \mathbb{N}$ such that $\mathit{type}(a_i) = (k_{i-1}, k_i)$ for all $i \in [n]$. We let $\mathit{type}(w) = (k_0, k_n)$. The set of all typed strings over $\mathbf{\Theta}$ is written $\mathbf{\Theta}^{\oplus}$. The concatenation $u \cdot v$ of typed strings $u, v \in \mathbf{\Theta}^{\oplus}$ with $\mathit{type}(u) = (i,j)$ and $\mathit{type}(v) = (m,n)$ is only defined if $j = m$.

Note that ordinary (untyped) alphabets and strings over them can be considered as special cases of typed ones by setting $\mathit{type}(a) = (1,1)$ for every symbol a. Further note that there is no empty typed string $\varepsilon \in \mathbf{\Theta}^+$ because its type would be undefined. Instead, we introduce so-called *blanks* below.

For a typed alphabet $\mathbf{\Theta}$ and a ranked alphabet Σ, an *interpretation operator* $[\![\cdot]\!] : \mathbf{\Theta} \to \mathbb{G}_\Sigma$ assigns a graph $[\![a]\!] \in \mathbb{G}_\Sigma$ with $\mathit{type}([\![a]\!]) = \mathit{type}(a)$ to each symbol $a \in \mathbf{\Theta}$. We extend $[\![\cdot]\!]$ to typed strings over $\mathbf{\Theta}$ by $[\![a_1 \cdots a_n]\!] \cong [\![a_1]\!] \odot \cdots \odot [\![a_n]\!]$ where $a_i \in \mathbf{\Theta}$ for $i \in [n]$.

Every interpretation operator $[\![\cdot]\!] : \mathbf{\Theta} \to \mathbb{G}_\Sigma$ defines a congruence relation $\sim$ on typed strings, as follows: for all $u, v \in \mathbf{\Theta}^{\oplus}$, $u \sim v$ if and only if $[\![u]\!] \cong [\![v]\!]$.

Given a ranked alphabet Σ, the *canonical alphabet* $(\mathbf{\Theta}_\Sigma, \mathit{type})$ is the typed alphabet given by

$$\begin{aligned}
\mathbf{\Theta}_\Sigma &= \{a_\varrho^\varphi \mid \langle a\rangle_\varrho^\varphi \text{ is an atom in } \mathbb{G}_\Sigma\} \cup \mathcal{B} \\
\mathcal{B} &= \{\varepsilon_\varrho^{(n)} \mid \langle \varepsilon\rangle_\varrho^{(n)} \text{ is a blank in } \mathbb{G}_\Sigma\} \\
\mathit{type}(a_\varrho^\varphi) &= (|\varphi|, |\varrho|) \quad \text{for all } a_\varrho^\varphi \in \mathbf{\Theta}_\Sigma \\
\mathit{type}(\varepsilon_\varrho^{(n)}) &= (n, |\varrho|) \quad \text{for all } \varepsilon_\varrho^{(n)} \in \mathcal{B}.
\end{aligned}$$

The *canonical interpretation* of $\mathbf{\Theta}_\Sigma$ is given by $[\![a_\varrho^\varphi]\!] = \langle a\rangle_\varrho^\varphi$ for all $a_\varrho^\varphi \in \mathbf{\Theta}_\Sigma \setminus \mathcal{B}$ and $\left[\!\!\left[\varepsilon_\varrho^{(n)}\right]\!\!\right] = \langle \varepsilon\rangle_\varrho^{(n)}$ for all $\varepsilon_\varrho^{(n)} \in \mathcal{B}$. We call all symbols in $\mathcal{B}$ *blank symbols* (or just *blanks*), and abbreviate $\varepsilon_\varrho^{(n)}$ with $\varrho = 1 \cdots n$ as $\varepsilon^{(n)}$. Note that $\mathcal{B} \subseteq \mathbf{\Theta}_\Sigma$ is infinite and independent of Σ.

We note the following immediate consequences of the definitions above:

Fact 3.1. *The following holds for every ranked alphabet Σ and its canonical alphabet $\mathbf{\Theta}_\Sigma$:*

1. *Let $a, b \in \mathbf{\Theta}_\Sigma$ with $\mathit{type}(a) = (m,n)$ and $\mathit{type}(b) = (n,k)$ (for some $m,n,k \in \mathbb{N}$) be such that $\{a,b\} \cap \mathcal{B} \neq \varnothing$. Then $\mathbf{\Theta}_\Sigma$ contains a symbol c such that $ab \sim c$. In particular, $\mathit{type}(c) = (m,k)$. (Note that $c \in \mathcal{B}$ if both a and b are blanks, and $c \in \mathbf{\Theta}_\Sigma \setminus \mathcal{B}$ otherwise.)*

2. For all $m,n \in \mathbb{N}$ and $a \in \mathbf{\Theta}_\Sigma$ with $type(a) = (n,m)$, we have $\varepsilon^{(n)} a \sim a \sim a\varepsilon^{(m)}$.

By requiring that finite automata respect types, we obtain finite automata over a finite typed alphabet Θ.

Definition 3.2 (Finite Automaton)**.** Let Θ be a finite typed alphabet. A *finite automaton* over Θ is a tuple $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ such that Q is a ranked alphabet of *states*, $\Delta \subseteq (Q \times \Theta \times Q)$ is a set of *transitions* (q,a,q') such that $type(a) = (i,j)$ implies that $rank(q) = i$ and $rank(q') = j$, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q \setminus \{q_0\}$ is a non-empty set of *final states* such that $rank(q) = rank(q')$ for all $q, q' \in F$.

We let $type(\mathfrak{A}) = (m,n)$ where $m = rank(q_0)$ and $n = rank(q)$ for all $q \in F$. $\mathfrak{A}$ is *deterministic* if, for each pair of transitions $(p,a,q), (p',b,q') \in \Delta$, $p = p'$ and $a = b$ implies $q = q'$.

A *configuration* of $\mathfrak{A}$ is a pair $(q,w) \in Q \times \Theta^*$ consisting of the *current state* q and the *remaining input* w. It is *initial* if $q = q_0$, and *accepting* if $q \in F$ and $w = \varepsilon$. Note that configurations are defined for arbitrary $w \in \Theta^*$, i.e., w is not necessarily typed.

A transition $\delta = (q,a,q') \in \Delta$, where a is a symbol in Θ, defines *moves* $(q,aw) \vdash_\delta (q',w)$ for all strings $w \in \Theta^*$. We write $(q,w) \vdash_{\mathfrak{A}} (q',w')$ if $(q,w) \vdash_\delta (q',w')$ for some transition $\delta \in \Delta$.

The *language* accepted by $\mathfrak{A}$ is defined as usual:

$$\mathcal{L}(\mathfrak{A}) = \{w \in \Theta^* \mid \exists q \in F : (q_0,w) \vdash_{\mathfrak{A}}^* (q,\varepsilon)\}.$$

Note that $\mathcal{L}(\mathfrak{A}) \subseteq \Theta^\oplus$ because transitions are typed and $q_0 \notin F$. In fact, we have $type(w) = type(\mathfrak{A})$ for all $w \in \mathcal{L}(\mathfrak{A})$.

Associating a graph interpretation operator with Θ, we can recognize graph languages in the obvious way: a graph G is accepted by an automaton $\mathfrak{A}$ if $G \cong [\![w]\!]$ for some string $w \in \mathcal{L}(\mathfrak{A})$. To enable our automata to work directly on graphs, we use *graph configurations* and the corresponding moves:

Definition 3.3 (Graph configurations and moves)**.** Let $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ be a finite automaton with $\Theta \subseteq \mathbf{\Theta}_\Sigma$. A pair (q,G) consisting of a state $q \in Q$ and a graph $G \in \mathbb{G}_\Sigma$ is a *graph configuration*, or simply *configuration* if the context prevents confusion.

A transition $\delta = (q,a,q') \in \Delta$ can be applied to a graph $G \in \mathbb{G}_\Sigma$ if there are graphs $G', G_a \in \mathbb{G}_\Sigma$ such that $G = G_a \odot G'$ and $G_a \cong [\![a]\!]$. We then call $(q,G) \Vdash_\delta (q',G')$ a *move* (using δ). We write $(q,G) \Vdash_{\mathfrak{A}} (q',G')$ if $(q,G) \Vdash_\delta (q',G')$ for some transition $\delta \in \Delta$. The set of *acceptable configurations* of $\mathfrak{A}$ with $type(\mathfrak{A}) = (m,n)$ is

$$\mathcal{C}_{\mathfrak{A}} = \{(q,G) \in Q \times \mathbb{G}_\Sigma \mid \exists q' \in F, G' \in \mathbb{G}_\Sigma \colon (q,G) \Vdash_{\mathfrak{A}}^* (q',G') \text{ and } G' \cong \langle \varepsilon \rangle_{1\ldots n}^{(n)}\}.$$

The graph language accepted by $\mathfrak{A}$ is then

$$\mathcal{L}_G(\mathfrak{A}) = \{G \in \mathbb{G}_\Sigma \mid (q_0,G) \in \mathcal{C}_{\mathfrak{A}}\}.$$

Obviously (and provable by a straightforward induction), we have $\mathcal{L}_G(\mathfrak{A}) = \{[\![w]\!] \mid w \in \mathcal{L}(\mathfrak{A})\}$. However, given a graph $G \in \mathcal{L}_G(\mathfrak{A})$, it is usually not the case that $w \in \mathcal{L}(\mathfrak{A})$ for all $w \in \Theta^\oplus$ such that $[\![w]\!] = G$, as demonstrated in the following example. This turns efficient graph recognition with finite automata into a nontrivial problem.

Example 3.1. Let us consider the graph language of "stars" where each "star" consists of a center node, which is the only front interface node, and at least two satellite nodes, which are connected by binary edges with the center node. Just one of the edges is labeled with b, the

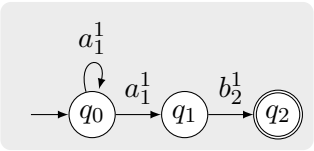


Figure 2: The finite automaton $\mathfrak{S}$ recognizing the graph language of "stars".

others with a, and the rear interface consists of just the satellite node attached to the b-labeled edge. Figure 1 shows such a "star", and Figure 2 the finite automaton $\mathfrak{S}$ recognizing this graph language. As usual, we draw finite automata with circles as states and arrows as transitions. The initial state is indicated by an incoming arrow from nowhere, final states by double borders. $\mathfrak{S}$ is nondeterministic since it contains a transition (q_0, a_1^1, q_0) as well as (q_0, a_1^1, q_1).

One can see that the "star" G in Figure 1 is a member of $\mathcal{L}_G(\mathfrak{S})$ because the string $w = a_1^1\, a_1^1\, b_2^1$ is recognized by $\mathfrak{S}$ and $G \cong [\![w]\!]$. Note that $[\![w]\!] \cong [\![a_1^1]\!] \odot [\![a_1^1]\!] \odot [\![b_2^1]\!] = \langle a \rangle_1^1 \odot \langle a \rangle_1^1 \odot \langle b \rangle_2^1$, which is one of the compositions of G shown in Example 2.2, whereas, e.g., the string $b_{12}^1\, a_{123}^{13}\, a_{1234}^{134}\, \varepsilon_4^{(4)}$ cannot be recognized by $\mathfrak{S}$.

4 Efficient Graph Recognition with Finite Automata

Let us now use a finite automaton for the recognition of graphs. To achieve efficiency, we must avoid nondeterminism. To see this, consider a nondeterministic automaton and a situation where we have reached a configuration with a state that has several outgoing transitions reading the same symbol. This means that all of them are applicable whenever one of them can be applied. Consequently, we must try one of them first, and if this choice leads into a dead end later, we are forced to backtrack and then try the next one. Backtracking usually leads to exponential running times, and should thus be avoided for efficient recognition.

Note that blank transitions, that is, transitions that read a blank, can lead to nondeterminism, even if the automaton is in fact deterministic. To understand this, consider a state with at least two outgoing transitions, one of them a blank transition. Similarly to an epsilon transition in an ordinary finite automaton, a blank transition (of the right type) can always be applied. So whenever one of the other transitions is applicable, one must choose between that transition and the blank transition. If one picks the "wrong" one that leads into a dead end, backtracking is necessary in order to choose the other one.

For efficient graph recognition with finite automata, we first develop an extension of the well-known powerset construction to turn a finite automaton over a subset of the canonical alphabet into a deterministic one without blank transitions in places where they may trigger backtracking.

4.1 Powerset Construction for Finite Automata

General Assumption. Throughout this subsection, we consider a fixed (ranked) alphabet Σ, its canonical alphabet $\boldsymbol{\Theta}_\Sigma$, and a finite automaton $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ over a typed alphabet $\Theta \subset \boldsymbol{\Theta}_\Sigma$. We also let $(m, n) = \mathit{type}(\mathfrak{A})$.

Algorithm 1 extends the well-known powerset construction and computes a new automaton whose states are built from sets of states of the input automaton. As mentioned above, blank

Algorithm 1: Powerset construction for a finite automaton.

Input : Finite automaton $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ with $\Theta \subset \mathbf{\Theta}_\Sigma$.
Output: Powerset automaton $\mathfrak{A}' = (\Theta', Q', \Delta', S_0, F')$.

1 $S_0 \leftarrow Cl(q_0)$ and $S_f \leftarrow \varnothing$
2 $Q' \leftarrow \{S_0, S_f\}$ and $F' \leftarrow \{S_f\}$ where $(rank(S_0), rank(S_f)) = type(\mathfrak{A})$
3 $\Delta' \leftarrow \varnothing$
4 *frontier* $\leftarrow \{S_0\}$
5 **while** *frontier* $\neq \varnothing$ **do**
6 select and remove any X from *frontier*
7 **foreach** $(\beta, q) \in X$ *such that* $q \in F$ **do** add (X, β, S_f) to Δ'
8 $\psi \leftarrow \{(a, q') \in (\mathbf{\Theta} \setminus \mathcal{B}) \times Q \mid (\beta, q) \in X,\ (q, b, q') \in \Delta,$ and $a \sim \beta b\}$
9 **foreach** $a_0 \in dom(\psi)$ **do**
10 let $type(a_0) = (i, j)$
11 $Y \leftarrow Cl(\psi(a_0))$
12 **if** $Y = \beta' Y'$ *for some* $Y' \in Q'$ *and* $\beta' \in \mathcal{B}$ *with* $type(\beta') = (j, j)$ **then**
13 add (X, a_1, Y') to Δ' for some $a_1 \in \Theta \setminus \mathcal{B}$ such that $a_1 \sim a_0 \beta'$
14 **else**
15 let $rank(Y) = j$ and add Y to Q' as well as to *frontier*
16 add (X, a_0, Y) to Δ'
17 $\Theta' \leftarrow \{a \in \mathbf{\Theta}_\Sigma \mid \exists S, S'' \in Q' : (S, a, S') \in \Delta'\}$

transitions resemble epsilon transitions that "read" the empty word. However, blanks are more complicated because they must take nodes into account; remember that the rear interface of a blank can contain a subset of its nodes in any order (cf. Figure 1). Therefore, Algorithm 1 considers pairs of the form (β, q) where β is a blank and q a state of the input automaton. Informally speaking, such a pair indicates that one can reach state q after reading β. Note that β can be $\varepsilon^{(i)}$ for an appropriate i, that is, an identity without any effect (which is comparable to ε in the string case). The states of the computed automaton then consist of sets of such pairs.

When building these sets, the algorithm must follow any sequence of blank transitions and combine their blanks into a single blank, which is always possible thanks to Fact 3.1. We call the set of all states (together with these blanks) that can be reached from a state $q \in Q$ by a sequence of blank transitions the *closure* of q. It is defined as

$$Cl(q) = \{(\beta, q') \in \mathcal{B} \times Q \mid \exists \sigma \in \mathcal{B}^* \colon (q, \sigma) \vdash_{\mathfrak{A}}^* (q', \varepsilon) \text{ and } \varepsilon^{(i)} \beta \sim \varepsilon^{(i)} \sigma\}$$

where $rank(q) = i$. The condition $\varepsilon^{(i)} \beta \sim \varepsilon^{(i)} \sigma$ in this definition can be simplified to $\beta \sim \sigma$ unless $\sigma = \varepsilon$. If $\sigma = \varepsilon$ then $\varepsilon^{(i)}$ is required because $[\![\varepsilon]\!]$ is undefined. Note also that $(\varepsilon^{(i)}, q) \in Cl(q)$, and that $(\beta, q') \in Cl(q)$ implies $type(\beta) = (i, rank(q'))$.

We extend the definition of closures to sets of states by defining $Cl(M) = \bigcup_{q \in M} Cl(q)$ for all $M \subseteq Q$ as used in Line 11. For every set $C \subseteq \mathcal{B} \times Q$ and $i \in \mathbb{N}$ such that $(\beta', q) \in C$ implies $type(\beta') = (i, rank(q))$, and every blank $\beta \in \mathcal{B}$ of type (i, i), we define

$$\beta C = \{(\beta'', q) \in \mathcal{B} \times Q \mid \exists \beta' \in \mathcal{B} \colon (\beta', q) \in C \text{ and } \beta'' \sim \beta \beta'\}$$

as used in Line 12. Note that a blank β'' with $\beta'' \sim \beta\beta'$ always exists, thanks to Fact 3.1.1.

We now prove the correctness of Algorithm 1 by first showing that it always terminates and produces a proper finite automaton as a result. After that, we will show that the produced powerset automaton is equivalent to the input automaton.

Lemma 4.1. *Algorithm 1 terminates for every finite automaton* $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ *and outputs a finite automaton.*

Proof. We use the notations established in Algorithm 1. In particular, $\mathfrak{A}' = (\Theta', Q', \Delta', S_0, F')$ is the output of the algorithm.

Every state in Q' is a subset of $\mathcal{B} \times Q$. The fact that Q and Θ are finite thus implies that Q' is finite. Since no element of Q' is added to *frontier* more than once, none of the loops of Algorithm 1 can run endlessly.

It remains to be shown that the transitions of $\mathfrak{A}'$ are correctly typed. For this, we consider any transition $\delta = (S, a, S') \in \Delta'$ and show that $\mathit{type}(a) = (\mathit{rank}(S), \mathit{rank}(S'))$. Transition δ must have been added in Lines 7, 13, or 16. Before we consider each of these cases, first note that each state $S \neq S_f$ is a non-empty set of pairs $(\beta, q) \in \mathcal{B} \times Q$ that satisfy $\mathit{type}(\beta) = (\mathit{rank}(S), \mathit{rank}(q))$ by the definition of closures, how ψ is computed in Line 8, and how $\mathit{rank}(S)$ is set in Lines 2 and 15. Further note that we have $\mathit{rank}(X) = i$ when Line 12 is reached, also by the definition of closures and how ψ is computed. We now distinguish the three cases where δ may have been added to Δ':

- Line 7: Since $(\beta, q) \in X$ and $q \in F$, $\mathit{type}(\beta) = (\mathit{rank}(X), \mathit{rank}(q)) = (\mathit{rank}(X), \mathit{rank}(S_f))$
- Line 13: $\mathit{rank}(Y') = j$ and $\mathit{type}(a_1) = (i, j)$ follow from $Y = \beta' Y'$, $\mathit{type}(\beta') = (j, j)$, and $\mathit{type}(a_0) = (i, j)$, and hence $\mathit{type}(a_1) = (\mathit{rank}(X), \mathit{rank}(Y'))$.
- Line 16: $\mathit{rank}(Y)$ is set to j, and hence $\mathit{type}(a_0) = (\mathit{rank}(X), \mathit{rank}(Y))$,

Hence, the transitions in Δ' are correctly typed, i.e., $\mathfrak{A}'$ is a finite automaton. □

To prove that the powerset automation $\mathfrak{A}'$ produced by Algorithm 1 is equivalent to its input automaton $\mathfrak{A}$, we now show that every accepting sequence of moves in $\mathfrak{A}$ has a corresponding sequence of moves in $\mathfrak{A}'$ (Lemma 4.3) and vice versa (Lemma 4.4). For proving Lemma 4.3, we will need the following auxiliary result. Informally speaking (and ignoring the fact that we are actually dealing with pairs (β, q) instead of just states q), Lemma 4.2 proves that every state reachable from a closure state by a blank transition is also contained in the closure:

Lemma 4.2. *Let* $q, q', q'' \in Q$, $\beta, \beta', \beta'' \in \mathcal{B}$, *and* $\delta = (q', \beta', q'') \in \Delta$. *If* $(\beta, q') \in \mathit{Cl}(q)$ *and* $\beta'' \sim \beta\beta'$ *then* $(\beta'', q'') \in \mathit{Cl}(q)$.

Proof. Let $\mathit{rank}(q) = i$. By the definition of $\mathit{Cl}(q)$, $(\beta, q') \in \mathit{Cl}(q)$ implies $(q, \sigma) \vdash_{\mathfrak{A}}^{*} (q', \varepsilon)$ and $\varepsilon^{(i)}\beta \sim \varepsilon^{(i)}\sigma$ for some $\sigma \in \mathcal{B}^*$. Consequently, $(q, \sigma\beta') \vdash_{\mathfrak{A}}^{*} (q', \beta') \vdash_{\delta} (q'', \varepsilon)$ and $\varepsilon^{(i)}\sigma\beta' \sim \varepsilon^{(i)}\beta\beta' \sim \varepsilon^{(i)}\beta''$, and thus $(\beta'', q'') \in \mathit{Cl}(q)$. □

Lemma 4.3. *Consider* $q \in Q$ *and* $w \in \Theta^*$ *such that* $(q_0, w) \vdash_{\mathfrak{A}}^{*} (q, \varepsilon)$. *Then there exist* $w' \in \Theta^*$, $S \in Q'$, *and* $\beta \in \mathcal{B}$ *such that* $(S_0, w') \vdash_{\mathfrak{A}'}^{*} (S, \varepsilon)$, $(\beta, q) \in S$, *and* $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w' \beta$.[2]

Proof. We prove the proposition by induction over the length ℓ of the move sequence $(q_0, w) \vdash_{\mathfrak{A}}^{*} (q, \varepsilon)$. The proposition is clearly true for $\ell = 0$ (with $w = w' = \varepsilon$, $S = S_0$, and $\beta = \varepsilon^{(m)}$). For $\ell > 0$, we consider the sequence $(q_0, wa) \vdash_{\mathfrak{A}}^{\ell-1} (q, a) \vdash_{\delta} (q', \varepsilon)$ using transition $\delta = (q, a, q') \in \Delta$.

[2] Recall that $\mathit{type}(\mathfrak{A}) = (m, n)$.

By the induction hypothesis, there is a sequence $(S_0, w') \vdash^*_{\mathfrak{A}'} (S, \varepsilon)$ with $(\beta, q) \in S$ and $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w'\beta$. We have to show that there is also a sequence $(S_0, w'a') \vdash^*_{\mathfrak{A}'} (S', \varepsilon)$ with $(\gamma, q') \in S'$ and $\varepsilon^{(m)} wa \sim \varepsilon^{(m)} w'a'\gamma$. We distinguish two cases:

- $a \in \mathcal{B}$: By Lines 2, 11, and 15 of Algorithm 1, $(\beta, q) \in Cl(q'') \subseteq S$ for some $q'' \in Q$. Let $\beta' \in \mathcal{B}$ such that $\beta' \sim \beta a$. We have $(\beta', q') \in Cl(q'') \subseteq S$ by Lemma 4.2 and $\varepsilon^{(m)} wa \sim \varepsilon^{(m)} w'\beta a \sim \varepsilon^{(m)} w'\beta'$ by the induction hypothesis and the definition of β'. This proves the proposition by choosing $a' = \varepsilon$ and $\gamma = \beta'$.
- $a \in \Theta \setminus \mathcal{B}$: Whenever an element is added to Q', it is also added to *frontier* (Line 4 and Line 15). Since $S \in Q'$ and *frontier* $= \varnothing$ when Algorithm 1 terminates, S must have been selected as X in Line 6 at some point. As $(\beta, q) \in S$ and $\delta = (q, a, q') \in \Delta$, there is a $b \in \Theta \setminus \mathcal{B}$ such that $b \sim \beta a$, $(b, q') \in \psi$ after executing Line 8, and $(\varepsilon^{(j)}, q') \in Y$ after executing Line 11, and after selecting b as a_0 in Line 9. Y is handled in one of two sub-cases selected in Lines 12 and 14:
  - Line 12, i.e., Q' already contains a similar nonterminal Y' such that $Y = \beta' Y'$ for some blank $\beta' \in \mathcal{B}$ with $type(\beta') = (j, j)$, and $\delta' = (X, b', Y') = (S, b', Y')$ is added to Δ' where $b' \in \Theta \setminus \mathcal{B}$ and $b' \sim b\beta' = a_0\beta'$. Consequently, $(S_0, w'b') \vdash^*_{\mathfrak{A}'} (S, b') \vdash_{\delta'} (Y', \varepsilon)$. Since $(\varepsilon^{(j)}, q') \in Y$, there exists $\beta'' \in \mathcal{B}$ such that $(\beta'', q') \in Y'$ and $\varepsilon^{(j)} \sim \beta'\beta''$. Therefore,

    $$\varepsilon^{(m)} wa \sim \varepsilon^{(m)} w'\beta a \sim \varepsilon^{(m)} w'b \sim \varepsilon^{(m)} w'b\varepsilon^{(j)} \sim \varepsilon^{(m)} w'b\beta'\beta'' \sim \varepsilon^{(m)} w'b'\beta''$$

    by the induction hypothesis, the construction of b and b', and the fact that $\varepsilon^{(j)} \sim \beta'\beta''$. This proves the proposition by choosing $a' = b'$, $\gamma = \beta''$, and $S' = Y'$.
  - Line 14, i.e., Y is not already in Q', and neither is there a similar one. Thus, Y is added to Q' and $\delta' = (X, b, Y) = (S, b, Y)$ is added to Δ'. Consequently, $(S_0, w'b) \vdash^*_{\mathfrak{A}} (S, b) \vdash_{\delta'} (Y, \varepsilon)$ and $\varepsilon^{(m)} wa \sim \varepsilon^{(m)} w'\beta a \sim \varepsilon^{(m)} w'b$ by the induction hypothesis and the construction of b. This proves the proposition with $a' = b$, $\gamma = \varepsilon$, and $S' = Y$. □

Lemma 4.4. *Let $S \in Q'$, $\beta \in \mathcal{B}$, $q \in Q$, and $w' \in \Theta^*$ be such that $(\beta, q) \in S$ and $(S_0, w') \vdash^*_{\mathfrak{A}'} (S, \varepsilon)$. Then there exists $w \in \Theta^*$ such that $(q_0, w) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ and $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w'\beta$.*

Proof. We prove the proposition by induction over the length ℓ of the move sequence $(S_0, w') \vdash^*_{\mathfrak{A}'} (S, \varepsilon)$. For $\ell = 0$, we have $(S_0, \varepsilon) \vdash^0_{\mathfrak{A}'} (S_0, \varepsilon)$. Consider any $(\beta, q) \in S_0$. $S_0 = Cl(q_0)$ implies $(q_0, \sigma) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ for some $\sigma \in \mathcal{B}^*$ such that $\varepsilon^{(m)} \sigma \sim \varepsilon^{(m)} \beta$. This proves the proposition by choosing $w' = \varepsilon$ and $w = \sigma$.

For $\ell > 0$, the sequence $(S_0, w') \vdash^*_{\mathfrak{A}'} (S, \varepsilon)$ has the form $(S_0, w'_0 a) \vdash^{\ell-1}_{\mathfrak{A}'} (S', a) \vdash_\delta (S, \varepsilon)$, where $w' = w'_0 a$ and $\delta = (S', a, S) \in \Delta'$. By the induction hypothesis, for every $(\gamma, q') \in S'$, there is $w_0 \in \Theta^*$ such that $(q_0, w_0) \vdash^*_{\mathfrak{A}} (q', \varepsilon)$ and $\varepsilon^{(m)} w_0 \sim \varepsilon^{(m)} w'_0 \gamma$. We show that $(q_0, w) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ for some $w \in \Theta^*$ such that $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w'_0 a\beta$. This will complete the proof because $w' = w'_0 a$.

Transition $\delta = (S', a, S)$ must have been added to Δ' in one of the two cases selected in Lines 12 or 14.

- Line 12: $\delta = (S', a, S)$ has been added to Δ' after S' had been selected as X in Line 6, a terminal $a_0 \in \Theta \setminus \mathcal{B}$ had been selected in Line 9, Y had been computed in Line 11, and $S = Y'$ had been identified by $Y = \beta' Y'$ for some blank $\beta' \in \mathcal{B}$ with $type(\beta') = (j, j)$ and $a \sim a_0\beta'$. Since $(\beta, q) \in S = Y'$, $Y = \beta' Y'$ must contain some (γ_0, q) with $\gamma_0 \sim \beta'\beta$. There must be a state $q'' \in Q$ such that $(\gamma_0, q) \in Cl(q'')$ and $q'' \in \psi(a_0)$, where ψ is computed in Line 8, because $(\gamma_0, q) \in Y$. By the definition of $Cl(X)$, $(\gamma_0, q) \in Cl(q'')$ implies $(q'', \sigma) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ for

some $\sigma \in \mathcal{B}^*$ such that $\varepsilon^{(j)}\sigma \sim \varepsilon^{(j)}\gamma_0$. By the definition of ψ, there must be a transition $(q', b, q'') \in \Delta$ and $(\gamma, q') \in S' = X$ such that $a_0 \sim \gamma b$ and thus $(q_0, w_0 b\sigma) \vdash^*_{\mathfrak{A}'} (q', b\sigma) \vdash_{\mathfrak{A}} (q'', \sigma) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ by the induction hypothesis for some $w_0 \in \Theta^*$ such that $\varepsilon^{(m)} w_0' \gamma \sim \varepsilon^{(m)} w_0$. Therefore,

$$\varepsilon^{(m)} w_0 b\sigma \sim \varepsilon^{(m)} w_0' \gamma b\sigma \sim \varepsilon^{(m)} w_0' a_0 \sigma \sim \varepsilon^{(m)} w_0' a_0 \gamma_0 \sim \varepsilon^{(m)} w_0' a_0 \beta' \beta \sim \varepsilon^{(m)} w_0' a\beta.$$

This proves the statement of the lemma in this case by choosing $w = w_0 b\sigma$.

- Line 14: $\delta = (S', a, S)$ has been added to Δ' after S' had been selected as X in Line 6 and $S = Y$ had been computed in Line 11 after selecting a as a_0 in Line 9. There must be a state $q'' \in Q$ such that $(\beta, q) \in Cl(q'')$ and $(a, q'') \in \psi$ computed in Line 8 because $(\beta, q) \in S = Y$. Similar to the previous case, $(\beta, q) \in Cl(q'')$ implies $(q'', \sigma) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ for some $\sigma \in \mathcal{B}^*$ such that $\varepsilon^{(j)}\sigma \sim \varepsilon^{(j)}\beta$. By the definition of ψ, there must be a transition $(q', b, q'') \in \Delta$ and $(\gamma, q') \in S' = X$ with $a \sim \gamma b$ and thus $(q_0, w_0 b\sigma) \vdash^*_{\mathfrak{A}} (q', b\sigma) \vdash_{\mathfrak{A}} (q'', \sigma) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ by the induction hypothesis for some $w_0 \in \Theta^*$ such that $\varepsilon^{(m)} w_0' \gamma \sim \varepsilon^{(m)} w_0$. Therefore,

$$\varepsilon^{(m)} w_0 b\sigma \sim \varepsilon^{(m)} w_0' \gamma b\sigma \sim \varepsilon^{(m)} w_0' a\sigma \sim \varepsilon^{(m)} w_0' a\beta.$$

This proves the statement in this case, and thus completes the proof of the lemma, by setting $w = w_0 b\sigma$. □

We are now ready to prove the equivalence of a finite automaton and its powerset automaton using the following notion of equivalence:

Definition 4.5 (Equivalent automata)**.** Two finite automata of the same type (m, n) are *equivalent* if the following holds: for every $w \in \mathcal{L}(\mathfrak{A})$ there exists $w' \in \mathcal{L}(\mathfrak{A}')$ such that $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w'$, and for every $w' \in \mathcal{L}(\mathfrak{A}')$ there exists $w \in \mathcal{L}(\mathfrak{A})$ such that $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w'$.

Theorem 4.6. *$\mathfrak{A}$ and the powerset automaton $\mathfrak{A}'$ computed from $\mathfrak{A}$ by Algorithm 1 are equivalent.*

Proof. We have to prove both implications in Definition 4.5. To prove the first implication, let $w \in \Theta^*$ and consider a sequence $(q_0, w) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ with $q \in F$. By Lemma 4.3, there exist $w'' \in \Theta^*$, $S \in Q'$, and $\beta \in \mathcal{B}$ such that $(S_0, w'') \vdash^*_{\mathfrak{A}'} (S, \varepsilon)$, $(\beta, q) \in S$, and $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w''\beta$. Since $S \in Q'$, S must have been selected as X in Line 6 at some point. Since $q \in F$ and $(\beta, q) \in S$, a transition (S, β, S_f) has been added to Δ' in Line 7, and thus $(S_0, w''\beta) \vdash^*_{\mathfrak{A}'} (S, \beta) \vdash_{\mathfrak{A}'} (S_f, \varepsilon)$, i.e., $w''\beta \in \mathcal{L}(\mathfrak{A}')$, which proves the first implication by choosing $w' = w''\beta$.

Now consider any $w' \in \Theta^*$ and a sequence $(S_0, w') \vdash^*_{\mathfrak{A}'} (S_f, \varepsilon)$. Since S_f can only be reached by transitions added in Line 7, this sequence must have the form $(S_0, w''\beta) \vdash^*_{\mathfrak{A}'} (S, \beta) \vdash_{\mathfrak{A}'} (S_f, \varepsilon)$ for some $S \in q'$, $w'' \in \Theta^*$, $q \in F$, and $\beta \in \mathcal{B}$ such that $w' = w''\beta$ and $(\beta, q) \in S$. By Lemma 4.4, there is a $w \in \Theta^*$ such that $(q_0, w) \vdash^*_{\mathfrak{A}} (q, \varepsilon)$ and $\varepsilon^{(m)} w \sim \varepsilon^{(m)} w''\beta = \varepsilon^{(m)} w'$, which proves the other implication. □

4.2 Making the Powerset Automaton Deterministic

Since Algorithm 1 resembles the classical powerset construction, one might assume that the resulting powerset automaton is deterministic, but this is not always the case, demonstrated by the following example:

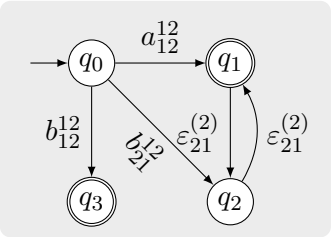


Figure 3: Automaton $\mathfrak{B}$ of Example 4.1.

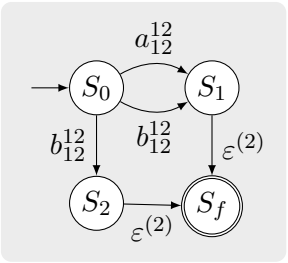


Figure 4: Nondeterministic automaton $\mathfrak{B}'$ produced from $\mathfrak{B}$ by Algorithm 1.

Example 4.1. Let a and b be symbols of rank 2, and consider the finite automaton $\mathfrak{B}$ shown in Figure 3 as input to Algorithm 1. Before the first iteration of the while loop beginning at Line 5 starts, we have $S_0 = \{(\varepsilon_{12}^{(2)}, q_0)\}$, $Q' = \{S_0, S_f\}$, $\Delta' = \varnothing$, and *frontier* $= \{S_0\}$. The first iteration of the while loop then selects $X = S_0$ and computes $\psi = \{(a_{12}^{12}, q_1), (b_{21}^{12}, q_2), (b_{12}^{12}, q_3)\}$. The nested foreach loop has the following iterations:

1. $a_0 = a_{12}^{12}$ and $Y = \{(\varepsilon_{12}^{(2)}, q_1), (\varepsilon_{21}^{(2)}, q_2)\}$. Lines 15 and 16 add Y as a new state S_1 to Q' and (S_0, a_{12}^{12}, S_1) to Δ'.
2. $a_0 = b_{21}^{12}$ and $Y = \{(\varepsilon_{12}^{(2)}, q_2), (\varepsilon_{21}^{(2)}, q_1)\}$. Since $Y = \varepsilon_{21}^{(2)} S_1$, Line 13 computes $a_1 = b_{12}^{12}$ because $b_{12}^{12} \sim b_{21}^{12}\varepsilon_{21}^{(2)}$, and adds (S_0, b_{12}^{12}, S_1) to Δ'.
3. $a_0 = b_{12}^{12}$ and $Y = \{(\varepsilon_{12}^{(2)}, q_3)\}$. Lines 15 and 16 add Y as a new state S_2 to Q' and (S_0, b_{12}^{12}, S_2) to Δ'.

Algorithm 1 terminates after adding $(S_1, \varepsilon_{12}^{(2)}, S_f)$ and $(S_2, \varepsilon_{12}^{(2)}, S_f)$ to Δ' with $Q' = \{S_0, S_1, S_2, S_f\}$. Figure 4 shows the resulting automaton $\mathfrak{B}'$. We have $(S_0, b_{12}^{12}, S_1), (S_0, b_{12}^{12}, S_2) \in \Delta'$, that is, $\mathfrak{B}'$ is nondeterministic. □

The reason for the nondeterminism of the resulting automaton is apparently the use of the atoms b_{12}^{12} and b_{21}^{12}, which is problematic because $b_{12}^{12} \sim b_{21}^{12}\varepsilon_{21}^{(2)}$. Let us call automata that use such symbols *ambiguous*:

Definition 4.7 (Ambiguous finite automata)**.** A finite automaton $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ is *ambiguous* if there are symbols $a, a' \in \Theta \setminus \mathcal{B}$ such that $a \neq a'$ and $a\beta \sim a'\beta'$ for some $\beta, \beta' \in \mathcal{B}$ (i.e., $[\![a]\!]$ and $[\![a']\!]$ differ only in their rears). $\mathfrak{A}$ is called *unambiguous* if it is not ambiguous.

Note that the automaton defined in Example 4.1 is ambiguous by this definition because $b_{21}^{12}\varepsilon_{21}^{(2)} \sim b_{12}^{12} \sim b_{12}^{12}\varepsilon_{12}^{(2)}$.

Theorem 4.8. *The powerset automaton $\mathfrak{A}'$ computed by Algorithm 1 is deterministic if $\mathfrak{A}$ is unambiguous.*

Proof. Consider any unambiguous automaton $\mathfrak{A}$, and assume that Algorithm 1 produces a nondeterministic automaton $\mathfrak{A}'$, i.e., there are two transitions $\delta = (S, a, S') \in \Delta'$ and $\delta' = (S, a, S'') \in \Delta'$ such that $S' \neq S''$. Note that we have target state S_f and $\beta \in \mathcal{B}$ for all transitions added to Δ' in Line 7, and $Y \neq S_f$, $Y' \neq S_f$, $a_0 \notin \mathcal{B}$, as well as $a_1 \notin \mathcal{B}$ for all transitions added in Lines 13 and 16, respectively. Consequently, neither δ nor δ' can have been added by Line 7, and both of them must have been added within the same iteration of the while loop using the same set ψ computed in Line 8, but in different iterations of the foreach loop that have selected, say, c_1 and

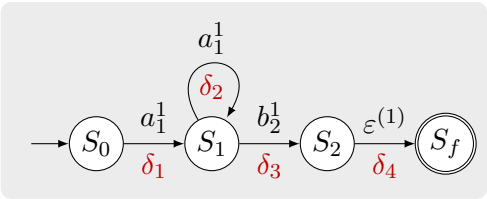


Figure 5: The deterministic automaton $\mathfrak{S}'$ obtained from $\mathfrak{S}$ (Figure 2) by Algorithm 1.

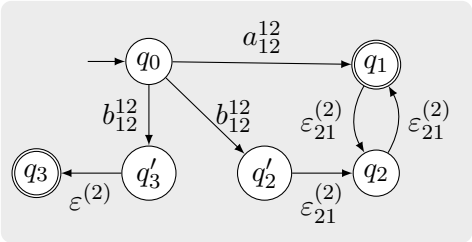


Figure 6: Unambiguous automaton $\mathfrak{B}''$ obtained from $\mathfrak{B}$ (Example 4.1).

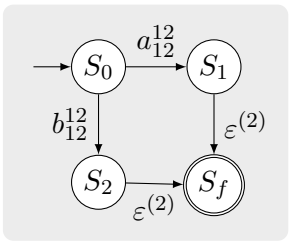


Figure 7: Automaton $\mathfrak{B}_\mathrm{d}$ obtained from $\mathfrak{B}''$ by Algorithm 1.

c_2 as a_0, respectively. If δ has been added to Δ' in Line 13, we have $a \sim c_1\beta_1$ for some $\beta_1 \in \mathcal{B}$, and $a = c_1$ if δ has been added in Line 16. Similarly, $a \sim c_2\beta_2$ for some $\beta_1 \in \mathcal{B}$, or $a = c_2$. We have $c_1 \neq c_2$ because c_1 and c_2 have been selected in different foreach loop iterations. Hence, either $c_1 \sim c_2\beta_2$, $c_1\beta_1 \sim c_2$, or $c_1\beta_1 \sim c_2\beta_2$, in contradiction to $\mathfrak{A}$ being unambiguous. □

Example 4.2. The automaton $\mathfrak{S}$ shown in Figure 2 is unambiguous, and Algorithm 1 thus computes a deterministic finite automaton (DFA) $\mathfrak{S}'$ shown in Figure 5. The states of $\mathfrak{S}'$ are $S_0 = \{(\varepsilon^{(1)}, q_0)\}$, $S_1 = \{(\varepsilon^{(1)}, q_0), (\varepsilon^{(1)}, q_1)\}$, $S_2 = \{\varepsilon^{(1)}, q_2)\}$, and $S_f = \varnothing$.

The restriction to unambiguous automata is in fact insignificant for automata over subsets of the canonical alphabet $\mathbf{\Theta}_\Sigma$ (and its canonical interpretation). One can easily transform an automaton $\mathfrak{A}$ into an equivalent unambiguous one using the following iterative process:

Consider an automaton $\mathfrak{A}$ that has $a, a' \in \Theta \setminus \mathcal{B}$ such that $a \neq a'$ and $a\beta \sim a'\beta'$ for some $\beta, \beta' \in \mathcal{B}$. Since we use the canonical interpretation of symbols in $\mathbf{\Theta}_\Sigma$, a and a' must have the form $a = b_\varrho^\varphi$ and $a' = b_{\varrho'}^\varphi$ using the same label $b \in \bar{\Sigma}$ and front interface φ, that is, a and a' differ only in their rear interfaces. Hence, one can identify a more general rear interface $\hat{\varrho}$ such that $a \sim b_{\hat{\varrho}}^\varphi \hat{\beta}$ and $a' \sim b_{\hat{\varrho}}^\varphi \hat{\beta}'$ for some blanks $\hat{\beta}$ and $\hat{\beta}'$. $\mathfrak{A}'$ is then obtained from $\mathfrak{A}$ by removing a and a' from Θ and adding $b_{\hat{\varrho}}^\varphi, \hat{\beta}$, and $\hat{\beta}'$. Furthermore, every transition (q, a, q') in Δ is replaced by two transitions $(q, b_{\hat{\varrho}}^\varphi, q'')$ and $(q'', \hat{\beta}, q')$ where q'' is a new state with $\mathit{rank}(q'') = |\hat{\varrho}|$ that is also added to Q, and proceeds similarly for each transition referring to a'. This process is continued until $\mathfrak{A}'$ is finally unambiguous.

Example 4.3. Figure 6 shows the unambiguous automaton $\mathfrak{B}''$ obtained from $\mathfrak{B}$ in Example 4.1 using the procedure sketched above. The ambiguous transitions (q_0, b_{21}^{12}, q_2) and (q_0, b_{12}^{12}, q_3) of $\mathfrak{B}$ have been split into four transitions with new intermediate states q_2' and q_3'. Algorithm 1 now produces the DFA $\mathfrak{B}_\mathrm{d}$ shown in Figure 7 when applied to $\mathfrak{B}''$.

4.3 Selecting Promising Transitions

Let us now consider the DFA $\mathfrak{A}_\mathrm{d}$ that has been produced by Algorithm 1, and use it for recognizing a valid graph $G \in \mathcal{L}_G(\mathfrak{A}_\mathrm{d})$ using graph moves. First note that $\mathfrak{A}_\mathrm{d}$ does not have any blank transitions except those that go into the final state, which does not have any outgoing transitions at all. Consequently, blank transitions are only applied when the remaining input is isomorphic to the corresponding blank graph, that is, there is no choice when such transitions have to be applied. Despite determinism, this is not necessarily the case for the other transitions. Consider the situation where we have reached a configuration (s,G) and G contains some edges. Although $\mathfrak{A}_\mathrm{d}$ is deterministic, several transitions may be applicable because a graph can be composed from basic graphs in different ways (see Examples 2.2 and 3.1). So we are again forced to choose and possibly to backtrack. To avoid this inefficient backtracking procedure, we would like to identify a unique transition, if it exists, that does not lead into a dead end, by inspecting only local information. To motivate such a procedure, let us first consider the simplest non-trivial case where s has two outgoing transitions, say $\delta, \delta' \in \Delta$ with $\delta \neq \delta'$. Now assume that we can prove that every edge read by a move using δ can never be read by any sequence of moves starting with δ'. It is then clear that any move using δ' instead of δ must inevitably lead into a dead end, and we know for sure that δ must be picked for continuing the recognition process. Moreover, backtracking later and trying δ' is meaningless because we already know that it will fail. It is thus crucial to find out whether an edge read by a move using δ can also be read by a sequence of moves starting with δ'. This is discussed next.

Again consider the situation where recognition has reached configuration (s,G), and two transitions $\delta, \delta' \in \Delta$, $\delta \neq \delta'$, can be used for two competing moves

$$(s,G) \Vdash_\delta (s',G') \tag{1}$$
$$(s,G) \Vdash_{\delta'} (s'',G'') \tag{2}$$

with $\delta = (s, a^{\varphi}_{\varrho}, s')$, $\delta' = (s, b^{\varphi'}_{\varrho'}, s'')$, $G = G_\delta \odot G' = G_{\delta'} \odot G''$ for two graphs G_δ and $G_{\delta'}$ with $\bar{G}_\delta = \{e\}$, $\bar{G}_{\delta'} = \{e'\}$, $G_\delta \cong [\![a^{\varphi}_{\varrho}]\!]$, and $G_{\delta'} \cong \left[\!\!\left[b^{\varphi'}_{\varrho'}\right]\!\!\right]$. The sequences φ and φ' specify how e and e' must be connected to front nodes of G such that (1) and (2) are a valid moves, respectively. To be more precise, let $\xi, \xi' \colon [\mathit{rank}(a)] \rightharpoonup [\mathit{rank}(s)]$ be partial functions defined by

$$\xi = \{(k,i) \in [\mathit{rank}(a)] \times [\mathit{rank}(s)] \mid k = \varphi(i)\}. \tag{3}$$
$$\xi' = \{(k,i) \in [\mathit{rank}(b)] \times [\mathit{rank}(s)] \mid k = \varphi'(i)\}. \tag{4}$$

Once more, we use the notation $\varphi(i)$ to access the i-th number within the sequence φ. Thus, ξ and ξ' assign the index of a front node (in the sequence of all front nodes of G) to its index in the sequence of all nodes attached to e or e', respectively. If a node is attached to e or e' as its k-th node, but is not a front node of G, $\xi(k)$ or $\xi'(k)$ are undefined, respectively.

We now assume that e is also read in the competing move (2) or a subsequent move. If e is read in (2), we have $e = e'$ as well as $a = b$, and $\xi = \xi'$ follows.

If $e \neq e'$, e must be read later, that is, there must be a sequence of moves

$$(s'',G'') \Vdash^*_{\mathfrak{A}_\mathrm{d}} (q,H) \Vdash_{\delta''} (q',H') \tag{5}$$

with $\delta'' = (q, a^{\varphi''}_{\varrho''}, q')$, $H = H_{\delta''} \odot H'$, and $\bar{H}_{\delta''} = \{e\}$, and $H_{\delta''} \cong \left[\!\!\left[a^{\varphi''}_{\varrho''}\right]\!\!\right]$.

Some front nodes of G may also be front nodes of H in (5). Let us indicate this situation by a partial function $\mu\colon [rank(s)] \rightharpoonup [rank(q)]$ defined by

$$\mu = \{(i,j) \in [rank(s)] \times [rank(q)] \mid front_G(i) = front_H(j)\} \tag{6}$$

Consequently, $\{front_G(i) \mid i \in dom(\mu)\}$ is the subset of those front nodes of G that are also front nodes of H. Similar to ξ and ξ', let us define the partial function $\xi''\colon [rank(a)] \rightharpoonup [rank(s)]$ by

$$\xi'' = \{(k,i) \in [rank(a)] \times dom(\mu) \mid k = \varphi''(\mu(i))\}. \tag{7}$$

Function μ is used to refer to front nodes of G instead of front nodes of H.

We now show that we have $\xi = \xi''$, similar to the case $e = e'$. To see this, consider any node v attached to e, that is, there is $k \in [rank(a)]$ such that $v = att_G(e,k)$.

- If v is not a front node of G, $k \notin [\varphi]$ follows from $G_\delta \cong [\![a^{\varphi}_{\varrho}]\!]$, and thus $k \notin dom(\xi)$. Now assume that $k \in dom(\xi'')$ and, hence, $k = \varphi''(j)$ as well as $j = \mu(i)$ for some $i \in [rank(s)]$ and $j \in [rank(q)]$. This implies $front_G(i) = front_H(j) = att_G(e,k) = v$ in contradiction to $v \notin [front_G]$. Hence, $k \notin dom(\xi'')$ as well.
- If v is a front node of G, there is an index $i \in [rank(s)]$ such that $v = front_G(i)$ and $k = \varphi(i)$, because of $G_\delta \cong [\![a^{\varphi}_{\varrho}]\!]$. Consequently, $k \in dom(\xi)$ and $i = \xi(k)$. When e is read in the last move (using δ'') in (5), it must also be a front node of H (otherwise, H could not contain v as a node because moves cannot turn front nodes into non-front nodes). Hence, $i \in dom(\mu)$ and $v = front_H(\mu(i))$. $H_{\delta''} \cong \left[\!\!\left[a^{\varphi''}_{\varrho''}\right]\!\!\right]$ also implies $k = \varphi''(\mu(i))$, and thus $k \in dom(\xi'')$ as well as $i = \xi''(k)$.

Note that the definition of ξ'' in fact subsumes the definition of ξ' when using the identity on $[rank(s)]$ as μ. Suppose we could compute the set $\Xi(\delta', a)$ of all functions ξ'' defined as in (7) for any graph G where μ represents any move sequence starting with δ' applied to G and finally reading an a-labelled edge (that is, exactly the situation as in (2) and (5)). Then we can summarize the discussion above as follows: If e can be read also in move (2) or later in (5), then $\xi \in \Xi(\delta', a)$. Since ξ and $\Xi(a)$ are independent of the specific choice of G and e, we can state the following observation:

Observation 4.9. *Let $\delta, \delta' \in \Delta$, ξ, and $\Xi(\delta', a)$ defined as described above, $\xi \notin \Xi(\delta', a)$, and $G \in \mathcal{G}_\Sigma$ any graph such that δ can be applied to G. Then δ' must not be tried for a move because it either cannot be applied to G, or its application leads inevitably into a dead end eventually.*

In fact, $\Xi(\delta', a)$ can be computed in a rather straightforward way using Algorithm 2. To simplify things, we use the notation $\Delta_q = \{\delta \in \Delta \mid \exists a \in \Theta, q' \in Q : \delta = (q, a, q')\}$ for indicating the set of transitions leaving a state q. Let us further define

$$follow(\delta) = \{(a, \xi) \mid \xi \in \Xi(\delta, a)\}. \tag{8}$$

The following lemma states that Algorithm 2 computes $follow(\delta)$.

Lemma 4.10. *Let Σ be a ranked alphabet, Θ a finite set of canonical graph symbols for Σ, $\mathfrak{A}_\mathrm{d} = (\Theta, Q, \Delta, q_0, F)$ a finite automaton produced by Algorithm 1, and $\delta_0 \in \Delta$ a transition. Called with $\mathfrak{A}_\mathrm{d}$ and δ_0, Algorithm 2 returns $follow(\delta_0)$ for $\mathfrak{A}_\mathrm{d}$.*

Proof Sketch. Consider any $\mathfrak{A}_\mathrm{d}$ and δ_0 as in the lemma. We first show that Algorithm 2 terminates. To see this, first note that $dom(\mu') \subseteq dom(\mu)$ and $\mu'(x) \in [rank(q')]$ for each $x \in dom(\mu')$

Algorithm 2: Determining $follow(\delta_0)$.

Input : Finite automaton $\mathfrak{A}_\mathrm{d} = (\Theta, Q, \Delta, q_0, F)$ produced by Algorithm 1 and a transition $\delta_0 \in \Delta$.

Output: Follow set *follow*.

```
1  let δ0 ∈ Δs
2  frontier ← {(δ0, ν)} where ν = {(i, i) | i ∈ [rank(s)]}
3  done ← ∅, follow ← ∅
4  while frontier ≠ ∅ do
5      select and remove any (δ, μ) from frontier
6      add (δ, μ) to done
7      let δ = (q, α, q′)
8      if α ∉ B then
9          let α = a_ϱ^φ
10         ξ = {(i, j) ∈ [rank(a)] × dom(μ) | i = φ(μ(j))}
11         add (a, ξ) to follow
12         μ′ ← {(i, j) ∈ dom(μ) × [rank(q′)] | φ(μ(i)) = ϱ(j)}
13         foreach δ′ ∈ Δ_q′ such that (δ′, μ′) ∉ done do
14             add (δ′, μ′) to frontier
```

of function μ' defined in Line 12, and thus $dom(\mu') \subseteq [rank(s)]$ for every pair (δ', μ') ever added to *frontier* in Line 14. Consequently, only finitely many pairs (δ, μ) can be added to *frontier* and *done*. And because no pair is added to *frontier* again after it has been selected in Line 5 (and adding it to *done* in Line 6), Algorithm 2 terminates.

Now consider any $a \in \Sigma$, $\xi \in \Xi(\delta_0, a)$ as for Observation 4.9, and any graph $G \in \mathbb{G}_\Sigma$. Consequently, there is a move sequence starting with δ_0 and finally reading an a-labeled edge using some transition δ such that ξ is defined like ξ' or ξ'' in (4) and (7), respectively, and using an appropriate partial function μ. By induction on the move sequence, one can show that *frontier* eventually will contain (δ, μ). Consequently, (a, ξ) is added to *follow* in Line 11, that is, (a, ξ) is contained in the result of Algorithm 2 as required.

For the other direction, consider any pair (a, ξ) in the result of Algorithm 2. By induction on the number of iterations of the while-loop starting at Line 4, one can show that the set *done* only contains pairs (δ, μ) such that there exists a move sequence starting with δ_0 and ending with δ so that μ is exactly as in (6). Since $(a, \xi) \in follow$, it must have been added to *follow* in Line 11 at some point after selecting (δ, μ) in Line 5. Using (4) and (7), one can show that $\xi \in \Xi(\delta_0, a)$ as required. □

We now use Observation 4.9 for identifying the unique transition (if it exists) that does not lead into a dead end, provided that $\mathfrak{A}_\mathrm{d}$ satisfies certain conditions. To this end, let us define

$$next(\delta) = (a, \xi) \tag{9}$$

for each transition $\delta = (s, a_\varrho^\varphi, s') \in \Delta$ where ξ is defined as in (3). Moreover, let $\prec_q \subseteq \Delta_q \times \Delta_q$ be such that

$$\delta \prec_q \delta' \quad \text{if} \quad next(\delta') \in follow(\delta) \tag{10}$$

for each $q \in Q$. According to Observation 4.9, $\delta' \not\prec_q \delta$ then indicates that δ' may only be tried if δ cannot be applied.

Consider any state $q \in Q$. If $\prec_q^+$ is irreflexive, one can extend $\prec_q$, by topological sorting, to a strict total order $\sqsubset_q \subseteq \Delta_q \times \Delta_q$ such that $\prec_q \subseteq \sqsubset_q$. Such an order $\sqsubset_q$ has the following nice property with respect to Observation 4.9:

Lemma 4.11. *Let $q \in Q$, $\sqsubset_q \subseteq \Delta_q \times \Delta_q$ a strict total order such that $\prec_q \subseteq \sqsubset_q$, and $\delta, \delta' \in \Delta_q$. Then $\delta \sqsubset_q \delta'$ implies $next(\delta) \notin follow(\delta')$.*

Proof. Consider any $q \in Q$ and $\delta, \delta' \in \Delta_q$ with $\delta \sqsubset_q \delta'$. Then $next(\delta) \in follow(\delta')$ would imply $\delta' \prec_q \delta$ and thus $\delta' \sqsubset_q \delta$, contradicting the irreflexivity of $\sqsubset_q$. □

Such a total strict order $\sqsubset_q$ only exists if $\prec_q^+$ is irreflexive, which motivates the following:

Definition 4.12 (Transition Selection Property)**.** A DFA $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ has the *transition selection* (*TS*) *property* if $\prec_q^+$ as defined in (10) is irreflexive for each state $q \in Q$.

An immediate consequence of Lemma 4.11 is the following:

Lemma 4.13. *For every DFA $\mathfrak{A}$ with the TS property, there is a constant time procedure which, given a configuration of $\mathfrak{A}$, selects the unique transition that allows to continue the recognition process without running into a dead end, provided that such a transition exists.*

Proof. Assume that a configuration with state q has been reached during recognition. There exists an order $\sqsubset_q$ as in Lemma 4.11 because of the TS property. Now try all outgoing transitions of q in ascending order of $\sqsubset_q$ and pick the first one, say δ, that can be applied. Then $\delta \sqsubset_q \delta'$ holds for every outgoing transition δ' that has not yet been tried, and thus $next(\delta) \notin follow(\delta')$ by Lemma 4.11, that is, δ' must not be tried by Observation 4.9 because δ can be applied.

The relation $\sqsubset_q$ depends only on $\mathfrak{A}$ and q, and can thus be precomputed for every state q. Checking whether a given transition applies takes constant time, because all that is required is to test whether an edge with the relevant label is attached to the front nodes of the graph as determined by the symbol to be read by the transition. Using appropriate data structures, this can straightforwardly be implemented to run in constant time. □

Example 4.4. Only state S_1 of the DFA $\mathfrak{S}'$ shown in Figure 5 has two outgoing transitions, δ_2 and δ_3. We obtain

$$next(\delta_2) = (a, \{(1,1)\}) \qquad follow(\delta_2) = \{(a, \{(1,1)\}), (b, \{(1,1)\})\}$$
$$next(\delta_3) = (b, \{(1,1)\}) \qquad follow(\delta_3) = \{(b, \{(1,1)\})\}$$

by applying (9) with (3) and using Algorithm 2, that is, $\delta_2 \prec_{S_1} \delta_3$ and $\delta_3 \not\prec_{S_1} \delta_2$, that is, $\mathfrak{S}'$ has the TS property. We can thus choose $\delta_2 \sqsubset_{S_1} \delta_3$, that is, one must try δ_2 first when one has reached S_1, and δ_3 only if δ_2 is not applicable, that is, if there is no remaining outgoing edge with label a.

4.4 Free Edge Choice

Unfortunately, the transition selection property does not necessarily prevent the recognition process from running into dead ends, even for valid graphs. This is the case if one has the choice between different edges to be read by a selected transition, but not all of them are equally suited. This is demonstrated in the following example:

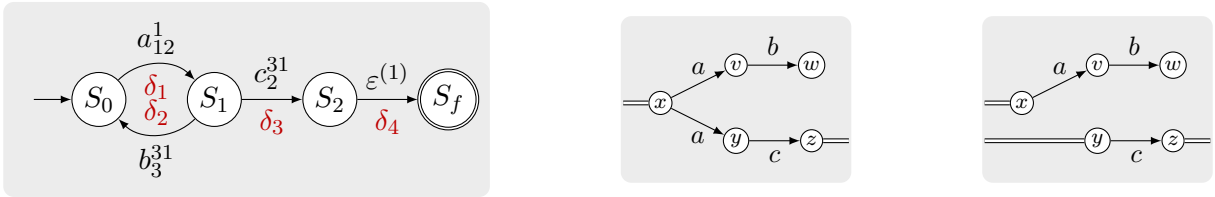


Figure 8: Finite automaton $\mathfrak{F}$ and two graphs used in Example 4.5.

Example 4.5. Let a, b, and c be symbols of rank 2. The DFA $\mathfrak{F}$ shown in Figure 8 always allows to select a unique transition: δ_1 in state S_0, δ_2 in S_1 if the second front interface node has an outgoing b-edge, and δ_3 if it has an outgoing c-edge. The left graph in Figure 8 can thus be recognized by first reading the edge from x to v in state S_0 and then following the uniquely determined moves.

However, the recognition process will run into a dead end if it reads the edge from x to y first. This move then yields a configuration (S_1, G') where G' is the right graph in Figure 8. The second front interface node (y) has an outgoing c-edge, but no b-edge, that is, δ_3 is selected. This transition tries to find a composition $G' = C \odot G''$ with $C \cong [\![c^{31}_2]\!]$, that is, $\mathit{front}_C = xy$, $\mathit{rear}_C = z$, and thus $x \notin \dot{G}''$, which would leave the edge from x to v dangling. Hence, δ_3 cannot be applied which leaves the recognition process stuck in a dead end. □

Such a situation cannot happen if every edge that can be chosen to be read next allows to continue the recognition process until an accepting configuration is reached, or none of them does. We say that the next edge can be freely chosen:

Definition 4.14 (Free Edge Choice Property)**.** A finite automaton $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ has the *free edge choice (FEC) property* if $(q', H') \in \mathcal{C}_\mathfrak{A}$ implies $(q', H'') \in \mathcal{C}_\mathfrak{A}$ for all move sequences $(q_0, G) \Vdash^*_\mathfrak{A} (q, H) \vdash_\delta (q', H')$ and $(q, H) \vdash_\delta (q', H'')$ where $\delta \in \Delta$.[3]

By the discussion above, the recognition process using an automaton with the TS property and the FEC property will never run into a dead end when applied to a valid graph. It remains to show how the FEC property can be checked for a given automaton. We present a sufficient condition that is easy to test. Thus, an automaton is guaranteed to have the FEC property if it passes the test, but the converse is not necessarily true.

Consider a situation during the recognition of a valid graph when a transition has been selected for the next move, and there are several edges one can choose from. All the edges that are not being chosen now must be read later during the recognition process. This is only possible for certain transitions. Let us call these transitions *deferrable* because one edge can be read now whereas reading of the others is deferred:

A transition $\delta \in \Delta$ is called *deferrable* if and only if there are move sequences

$$(q, G) \vdash_\delta (q', G') \tag{11}$$

$$\text{and} \quad (q, G) \vdash_\delta (q', G'') \Vdash^*_\mathfrak{A} (s, H) \vdash_\mathfrak{A} (s', H') \tag{12}$$

such that the edge read in (11) and the one read in the last move of (12) are the same, that is $\bar{G}' \setminus \bar{G} = \bar{H}' \setminus \bar{H}$.

[3] Recall that $\mathcal{C}_\mathfrak{A}$ is the set of all acceptable graph configurations of $\mathfrak{A}$, that is, those configurations from where an accepting configuration can be reached (see Definition 3.3).

One can easily check with an algorithm similar to Algorithm 2 whether a transition is deferrable. With this information, one can then check whether $\mathfrak{A}$ has the FEC property:

Lemma 4.15. *A finite automaton $\mathfrak{A}$ has the free edge choice property if $[\varrho] \subseteq [\varphi]$ holds for every transition $(q, a^{\varphi}_{\varrho}, q')$ of $\mathfrak{A}$ that is deferrable.*

Proof. Consider a finite automaton $\mathfrak{A} = (\Theta, Q, \Delta, q_0, F)$ that satisfies the condition in the lemma. Let us assume that $\mathfrak{A}$ does not have the free edge choice property, that is, there are move sequences $(q_0, G) \vdash^*_{\mathfrak{A}} (q, H) \vdash_{\delta} (q', H') \in \mathcal{C}_{\mathfrak{A}}$ and $(q, H) \vdash_{\delta} (q', H'') \notin \mathcal{C}_{\mathfrak{A}}$. The two moves using δ must have read two different edges, say e and e', that is, $H = F \odot H' = F' \odot H''$, $\bar{F} = \{e\}$, and $\bar{F}' = \{e'\}$. H' then still contains e', which is read later, because $(q', H') \in \mathcal{C}_{\mathfrak{A}}$. Consequently, there is a move sequence $(q', H') \vdash^*_{\mathfrak{A}} (s, I) \vdash_{\mathfrak{A}} (s', I') \in \mathcal{C}_{\mathfrak{A}}$ such that $\bar{I} \setminus \bar{I}' = \{e'\}$, that is, δ is deferrable. Now let $\delta = (q, a^{\varphi}_{\varrho}, q')$. We have $a = lab_G(e) = lab_G(e')$ since e and e' can both be read by δ. And we have $[\varrho] \subseteq [\varphi]$ since δ is deferrable, and $\mathfrak{A}$ satisfies the condition in the lemma. Note that $front_H = front_F = front_{F'}$. Thus $[\varrho] \subseteq [\varphi]$ implies $rear_F = rear_{F'} = front_{H'} = front_{H''}$. Consequently, all nodes of H that are attached to e or e', but that are not in $front_H$, cannot be attached to any other edge, and thus $H' \cong H''$, where the isomorphism maps e to e' (and vice versa). Then $(q', H'') \in \mathcal{C}_{\mathfrak{A}}$ follows from $(q', H') \in \mathcal{C}_{\mathfrak{A}}$, contradicting the assumption. $\mathfrak{A}$ hence has the free edge property. □

Example 4.6. Transition δ_1 is the only deferrable transition of automaton $\mathfrak{F}$ in Figure 8, but it violates the condition of Lemma 4.15 as it reads a^1_{12}, and $[\varrho] = \{1, 2\} \not\subseteq \{1\} = [\varphi]$. Consequently, $\mathfrak{F}$ does not pass the FEC test, which leaves the question open whether $\mathfrak{F}$ has the FEC property or not. In fact, we have already seen in Example 4.5 that $\mathfrak{F}$ does not have the FEC property.

The automaton $\mathfrak{S}'$ in Figure 5, however, has the FEC property according to Lemma 4.15 since δ_2 is its only deferrable transition, and $[\varrho] = \{1\} \subseteq \{1\} = [\varphi]$ for the graph symbol a^1_1 read by δ_2. Since $\mathfrak{S}'$ also has the TS property (see Example 4.4), one can use $\mathfrak{S}'$ for recognizing star graphs without any backtracking.

4.5 Results

The following theorem summarizes the findings of this paper.

Theorem 4.16. *1. Every finite automaton over the canonical alphabet can effectively be turned into a DFA.*[4]

2. *Given a DFA $\mathfrak{A}$ (of the type produced by Algorithm 1), it can be decided in polynomial time (in the number of transitions of $\mathfrak{A}$) whether $\mathfrak{A}$ has the TS and the FEC property.*
3. *If $\mathfrak{A}$ has the TS and the FEC property, the membership problem for $\mathcal{L}(\mathfrak{A})$ can be decided in linear time.*

Proof. The first statement follows from Theorem 4.8 and the previously mentioned fact that ambiguous automata can be made unambiguous. Deciding the TS property requires polynomial time because Algorithm 2 and topological sorting must be used for checking irreflexivity of $\prec^+_q$ for every state q of $\mathfrak{A}$. The second statement then follows from Lemma 4.15 and the fact that it is decidable in polynomial time whether a transition is deferrable (see above). Finally, to decide whether a graph belongs to $\mathcal{L}(\mathfrak{A})$, we can repeatedly apply transitions using Lemma 4.13. □

[4] As usual, and unavoidably, the powerset construction can cause an exponential blow-up in the size of the automaton.

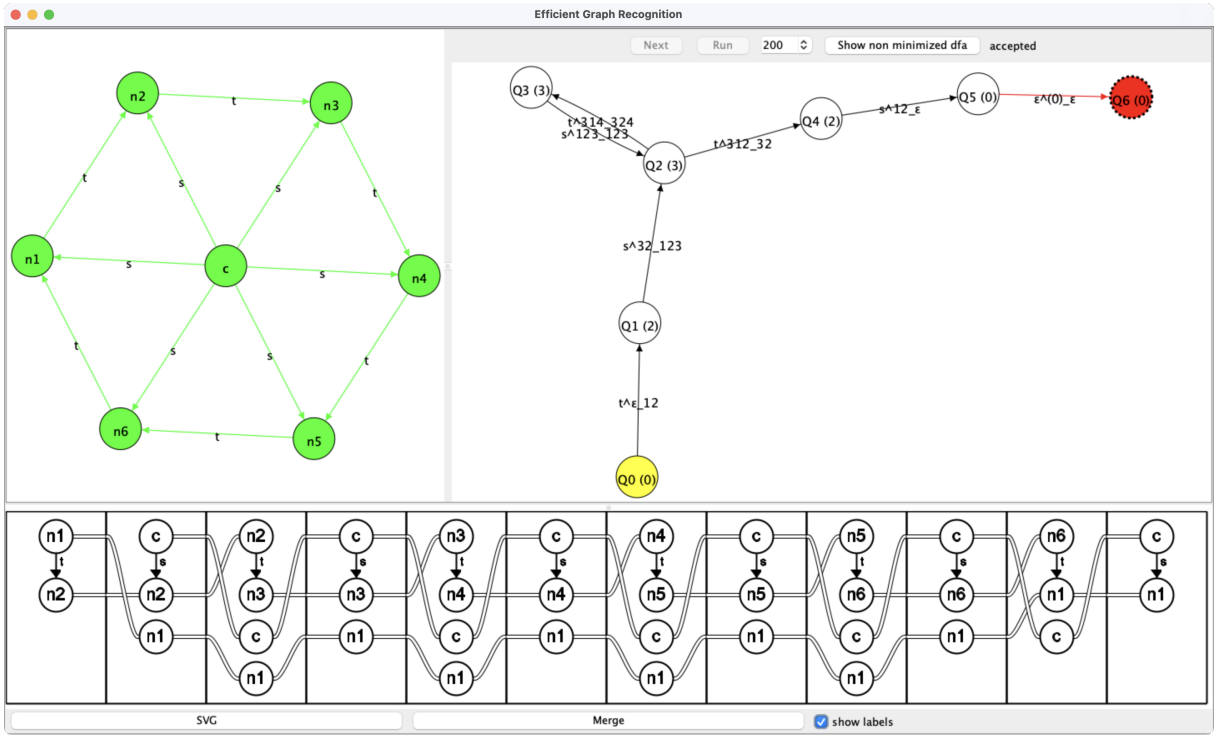


Figure 9: Screenshot of a tool realizing efficient graph recognition based on finite automata.

The concepts described in this paper have been realized in a graphical tool. It reads in a finite automaton over the canonical alphabet, or creates it from a regular expression in the obvious way, turns it into a DFA, and uses it for recognizing input graphs if the DFA has the TS and the FEC property. Figure 9 shows a screenshot with an input graph in the top-left window, and the DFA in the top-right window after the DFA has recognized the input graph. The bottom window shows how the input graph has been composed from basic graphs such that it has been accepted by the DFA.

Table 1 shows some example graph languages that can be specified and efficiently recognized by finite automata: *Stars* is the language of stars used in this paper (Example 3.1), *Wheels* contains wheel graphs introduced in [23, p. 92], *Palindromes* contains all palindromes over $\{a,b\}$, and $a^nb^nc^n$ the language $\{a^nb^nc^n \mid n > 0\}$ as string graphs [23, Ex. 2.4]. Table 1 shows the type of the contained graphs, that is, the size of their front and rear interfaces, and the size of their DFA in terms of the number of states and transitions. Moreover, *max. #nodes* shows the number of nodes of the largest atom used in the DFA. Each of the automata has the TS and FEC property.

Figure 9 shows the DFA for wheels and a wheel with six spokes (`s`-edges for spokes and `t`-edges for the tread of the wheel). The front and rear interfaces of the wheel are empty. As shown in the bottom window, the left-most atom refers to the `t`-edge from `n1` to `n2`, and recognition of the wheel has started with this part of the tread of the wheel. Recognition could have started with any of the `t`-edges because the DFA has the FEC property. However, the FEC test fails for this DFA because of the first transition from `Q0` to `Q1` with label `t^ε_12`, which represents t^{ε}_{12}. This transition is deferrable, yet $\{1,2\} \not\subseteq \varnothing$. Hence, *Wheels* is an example where the simple

Table 1: Example graph languages and their finite automata.

Language	Type	#states	#transitions	max. #nodes	TS	FEC
Stars	$(1,1)$	4	4	2	✓	✓
Wheels	$(1,0)$	7	7	4	✓	✓
Palindromes	$(2,0)$	5	7	3	✓	✓
$a^n b^n c^n$	$(2,0)$	9	10	4	✓	✓

FEC test fails although the automaton does have the FEC property.

5 Conclusions

In this paper we have defined finite automata for graph recognition, devised a modified powerset construction to make them deterministic, and stated two criteria (transition selection and free edge choice) under which these automata work without backtracking.

This paper leaves some questions open. Do (nondeterministic) finite graph automata recognize NP-complete languages, which exist for (context-free) hyperedge replacement grammars? How do the languages accepted by finite graph automata relate to those definable with the "regular" graph grammars of Gilroy *et al.* [20], which have an efficient parsing algorithm (not defined with automata) that also requires free edge choice? (Probably these language classes are incomparable.)

We intend to continue our work in several directions. First, it is easy to define graph expressions defining graph languages by composition, Kleene star, and union. These expressions could be used in the nested graph conditions of Habel and Pennemann [24, 30] to specify global graph properties [19], and check them with automata. Second, we could consider borrowed nodes in graph expressions and graph automata that can be contracted with other nodes in a graph [16]. Then these mechanisms allow to define graphs of unbounded treewidth.

Acknowledgments. We are very grateful to the anonymous reviewers for their helpful comments and suggestions, which have helped to improve the quality of this paper.

References

[1] IJ.J. Aalbersberg, Andrzej Ehrenfeucht & Grzegorz Rozenberg (1986): *On the Membership Problem for Regular DNLC Grammars*. *Discrete Applied Mathematics* 13, pp. 79–85, doi:10.1016/0166-218X(86)90070-3.

[2] Michael A. Arbib & Yehoshafat Give'on (1968): *Algebra Automata I: Parallel Programming as a Prolegomena to the Categorical Approach*. *Information and Control* 12(4), pp. 331–345, doi:10.1016/S0019-9958(68)90374-4.

[3] Christoph Blume, H.J. Sander Bruggink, Martin Friedrich & Barbara König (2013): *Treewidth, Pathwidth and Cospan Decompositions with Applications to Graph-Accepting Tree Automata*. *Journal of Visual Languages & Computing* 24(3), pp. 192–206, doi:10.1016/j.jvlc.2012.10.002.

[4] Symeon Bozapalidis & Antonios Kalampakas (2006): *Recognizability of graph and pattern languages*. *Acta Informatica* 42(8-9), pp. 553–581, doi:10.1007/s00236-006-0006-z.

[5] Symeon Bozapalidis & Antonios Kalampakas (2008): *Graph automata*. *Theoretical Computer Science* 393(1-3), pp. 147–165, doi:10.1016/j.tcs.2007.11.022.

[6] Franz-Josef Brandenburg & Konstantin Skodinis (2005): *Finite graph automata for linear and boundary graph languages*. *Theoretical Computer Science* 332(1-3), pp. 199–232, doi:10.1016/j.tcs.2004.09.040.

[7] H. J. Sander Bruggink & Barbara König (2018): *Recognizable languages of arrows and cospans*. *Math. Struct. Comput. Sci.* 28(8), pp. 1290–1332, doi:10.1017/S096012951800018X.

[8] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones & Kevin Knight (2013): *Parsing Graphs with Hyperedge Replacement Grammars*. In: *Proc. 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, Sofia, Bulgaria, pp. 924–932. Available at https://aclanthology.org/P13-1091.

[9] Bruno Courcelle (1990): *The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs*. *Information and Computation* 85(1), pp. 12–75, doi:10.1016/0890-5401(90)90043-H.

[10] Frank Drewes (1993): *Recognising k–connected hypergraphs in cubic time*. *Theoretical Computer Science* 109, pp. 83–122, doi:10.1016/0304-3975(93)90065-2.

[11] Frank Drewes, Berthold Hoffmann & Mark Minas (2015): *Predictive Top-Down Parsing for Hyperedge Replacement Grammars*. In Francesco Parisi-Presicce & Bernhard Westfechtel, editors: *Graph Transformation - 8th International Conf., ICGT 2015. Proceedings*, *LNCS* 9151, Springer, pp. 19–34, doi:10.1007/978-3-319-21145-9_2.

[12] Frank Drewes, Berthold Hoffmann & Mark Minas (2017): *Predictive Shift-Reduce Parsing for Hyperedge Replacement Grammars*. In Fabio Gadducci & Timo Kehrer, editors: *Graph Transformation - 10th International Conf., ICGT 2017, Proceedings*, *LNCS* 12741, Springer, pp. 106–122, doi:10.1007/978-3-319-61470-0_7.

[13] Frank Drewes, Berthold Hoffmann & Mark Minas (2019): *Extending Predictive Shift-Reduce Parsing to Contextual Hyperedge Replacement Grammars*. In Esther Guerra & Fernando Orejas, editors: *Graph Transformation - 12th International Conf., ICGT 2019*, *LNCS* 11629, Springer, pp. 55–72, doi:10.1007/978-3-030-23611-3_4.

[14] Frank Drewes, Berthold Hoffmann & Mark Minas (2019): *Formalization and Correctness of Predictive Shift-Reduce Parsers for Graph Grammars based on Hyperedge Replacement*. *Journal on Logical and Algebraic Methods for Programming* 104, pp. 303–341, doi:10.1016/j.jlamp.2018.12.006.

[15] Frank Drewes, Berthold Hoffmann & Mark Minas (2021): *Rule-Based Top-Down Parsing for Acyclic Contextual Hyperedge Replacement Grammars*. In Fabio Gadducci & Timo Kehrer, editors: *Graph Transformation - 14th International Conference, ICGT 2021*, *LNCS* 12741, Springer, pp. 164–184, doi:10.1007/978-3-030-78946-6_9.

[16] Frank Drewes, Berthold Hoffmann & Mark Minas (2022): *Acyclic Contextual Hyperedge Replacement: Decidability of Acyclicity and Generative Power*. In Nicolas Behr & Daniel Strüber, editors: *Graph Transformation - 15th International Conference, ICGT 2022*, *LNCS* 13349, Springer, pp. 3–19, doi:10.1007/978-3-031-09843-7_1.

[17] Joost Engelfriet & Jan Joris Vereijken (1997): *Context-Free Graph Grammars and Concatenation of Graphs*. *Acta Informatica* 34(10), pp. 773–803, doi:10.1007/s002360050106.

[18] Fabio Gadducci & Reiko Heckel (1997): *An inductive view of graph transformation*. In Francesco Parisi-Presicce, editor: *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97*, *LNCS* 1376, Springer, pp. 223–237, doi:10.1007/3-540-64299-4_36.

[19] Haim Gaifman (1982): *On local and non-local properties*. In J. Stern, editor: *Proc. of the Herbrand Symposium, Logic Colloquium*, *Studies in Logic and the Foundations of Mathematics* 105, North-Holland, pp. 105–135, doi:10.1016/S0049-237X(08)71879-2.

[20] Sorcha Gilroy, Adam Lopez & Sebastian Maneth (2017): *Parsing Graphs with Regular Graph Grammars*. In: *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, Association for Computational Linguistics, Vancouver, Canada, pp. 199–208, doi:10.18653/v1/S17-1024.

[21] Sorcha Gilroy, Adam Lopez, Sebastian Maneth & Pijus Simonaitis (2017): *(Re)introducing Regular Graph Languages*. In Makoto Kanazawa, Philippe de Groote & Mehrnoosh Sadrzadeh, editors: *Proceedings of the 15th Meeting on the Mathematics of Language, MOL 2017, London, UK, July 13-14, 2017*, ACL, pp. 100–113, doi:10.18653/v1/w17-3410.

[22] Yehoshafat Give'on & Michael A. Arbib (1968): *Algebra Automata II: The Categorical Framework for Dynamic Analysis*. *Information and Control* 12(4), pp. 346–370, doi:10.1016/S0019-9958(68)90381-1.

[23] Annegret Habel (1992): *Hyperedge Replacement: Grammars and Languages*. *LNCS* 643, Springer, doi:10.1007/BFb0013875.

[24] Annegret Habel & Karl-Heinz Pennemann (2005): *Nested Constraints and Application Conditions for High-Level Structures*. In H.-J. Kreowski et al., editors: *Formal Methods in Software and System Modeling*, *LNCS* 3393, Springer, pp. 293–308, doi:10.1007/978-3-540-31847-7_17.

[25] Berthold Hoffmann & Mark Minas (2017): *Generating Efficient Predictive Shift-Reduce Parsers for Hyperedge Replacement Grammars*. In M. Seidl & S. Zschaler, editors: *STAF 2017 Workshops*, *LNCS* 10748, Springer, pp. 76–91, doi:10.1007/978-3-319-74730-9_7.

[26] Antonios Kalampakas (2011): *Graph Automata: The Algebraic Properties of Abelian Relational Graphoids*. In Werner Kuich & George Rahonis, editors: *Algebraic Foundations in Computer Science - Essays Dedicated to Symeon Bozapalidis on the Occasion of His Retirement*, *LNCS* 7020, Springer, pp. 168–182, doi:10.1007/978-3-642-24897-9_8.

[27] Michael Kaminski & Shlomit S. Pinter (1992): *Finite Automata on Directed Graphs*. *Journal of Computer and System Sciences* 44(3), pp. 425–446, doi:10.1016/0022-0000(92)90012-8.

[28] Klaus-Jörn Lange & Emo Welzl (1987): *String Grammars with Disconnecting or a Basic Root of the Difficulty in Graph Grammar Parsing*. *Discrete Applied Mathematics* 16, pp. 17–30, doi:10.1016/0166-218X(87)90051-5.

[29] Clemens Lautemann (1990): *The complexity of graph languages generated by hyperedge replacement*. *Acta Informatica* 27, pp. 399–421, doi:10.1007/BF00289017.

[30] Karl-Heinz Pennemann (2009): *Development of Correct Graph Transformation Systems*. Dissertation, Carl-von-Ossietzky-Universität Oldenburg. Available at http://oops.uni-oldenburg.de/884/1/pendev09.pdf.

[31] Walter Vogler (1991): *Recognizing Edge Replacement Graph Languages in Cubic Time*. In H. Ehrig, H.-J. Kreowski & G. Rozenberg, editors: *Proc. Fourth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, *LNCS* 532, Springer, pp. 676–687, doi:10.1007/BFb0017421.

[32] Kurt-Ulrich Witt (1981): *Finite Graph-Acceptors and Regular Graph-Languages*. *Information and Control* 50(3), pp. 242–258, doi:10.1016/S0019-9958(81)90351-X.