A Simple Notion of Parallel Graph Transformation and Its Perspectives

Hans-Jörg Kreowski, Sabine Kuske, and Aaron Lye(✉)

Department of Computer Science, University of Bremen,
P.O. Box 33 04 40, 28334 Bremen, Germany
{kreo,kuske,lye}@informatik.uni-bremen.de

Abstract. In this paper, we reconsider an old and simple notion of parallel graph transformation and point out various perspectives concerning the parallel generation of graph languages, the parallelization of graph algorithms, the parallel transformation of infinite graphs, and parallel models of computation.

1 Introduction

In 1976, Hartmut Ehrig and the first author introduced an approach to parallel graph transformation in [1]. Parallel computation is obtained by the application of parallel rules which are composed of component rules by means of disjoint union. This is a particular simple and intuitive operation on graphs (and the graphs within graph transformation rules) because it places graphs – in our case directed edge-labeled graphs – next to each other without overlap and extra connections. The application of parallel rules has some significant properties. The component rules can be applied in arbitrary order yielding the same result as a given application of a parallel rule (sequentialization theorem). Conversely, if the component rules can be applied and their applications are independent of each other in a certain sense, then their parallel rule can also be applied (parallelization theorem). Parallel graph transformation has been one of the major research topics within the area of graph transformation in the last four decades. In Sect. 3, we revisit the starting point by recalling the basic notions and results introduced in 1976. In the rest of the paper, we point out how parallel graph transformation can be used in the context of some significant issues of parallelism: parallel modes of language generation, parallelization of algorithm, infinite graph theory, and transformation of other parallel models into graph transformation models. In all cases, the sequentialization and parallelization theorems play an important role. All explicit examples are new. The parallelization of algorithm and the use of graph transformation in the theory of infinite graphs are novel attempts as far as we know. In more detail, the aspects considered in the Sects. 4 to 7 are the following.

In proper context, parallel generation may provide more generative power than sequential generation. Consider, for example, Lindenmeyer systems with

R. Heckel and G. Taentzer (Eds.): Ehrig Festschrift, LNCS 10800, pp. 61–82, 2018.
https://doi.org/10.1007/978-3-319-75396-6_4

context-free rules applied with maximum parallelism. If one requires in addition that in each step one set of rules out of several possibilities is used, then these TOL-systems can generate languages that are not context-free. In Sect. 4, we consider hyperedge replacement grammars as a counterpart to context-free grammars and show that they behave in the same way if they are used in the TOL-mode of transformation.

Frequently and in many contexts, parallelism is expected to allow more efficient computations than sequential ones. In Sect. 5, we demonstrate that this idea works also in the framework of graph transformation by analyzing and parallelizing a shortest-path algorithm.

In contrast to the usual approaches, our notion of parallel rules is not restricted to a finite number of component rules. In Sect. 6, we discuss the effect of the application of infinite parallel rules and exemplify that infinite graph transformation may contribute to infinite graph theory.

Parallel graph transformation provides a suitable framework for the modeling and analysis of parallel and concurrent processes. In particular, other approaches to parallel processing can be transformed into and interpreted as graph-transformational parallelism. This is demonstrated for the well-known cellular automata in Sect. 7.

Related work is discussed in the respective sections.

2 Preliminaries

In this section, we recall some basic notions and notations of graphs and graph transformation. In particular, we define the disjoint union of sets and graphs and consider its elementary properties as prerequisites for the introduction of parallel rules.

2.1 Disjoint Union of Sets

Let $F = (X_i)_{i \in I}$ be a family of sets for some index set I. Then a set X together with injective mappings $in_i \colon X_i \to X$ for all $i \in I$ is a *disjoint union* of F if $in_i(X_i) \cap in_j(X_j) = \emptyset$ for all $i \neq j$ and $\bigcup_{i \in I} in_i(X_i) = X$. X may be denoted by $\sum_{i \in I} X_i$. For $I = \{1, 2\}$, one may denote X also by $X_1 + X_2$. A disjoint union can be constructed as $\bigcup_{i \in I} (\{i\} \times X_i)$ with $in_i \colon X_i \to \bigcup_{i \in I} (\{i\} \times X_i)$ defined by $in_i(x) = (i, x)$ for all $i \in I$ and $x \in X_i$.

A disjoint union X with $(in_i)_{i \in I}$ of $F = (X_i)_{i \in I}$ has the following (universal) property: If Y is a set and $(f_i \colon X_i \to Y)_{i \in I}$ is a family of mappings, then there is a unique mapping $f \colon X \to Y$ with $f \circ in_i = f_i$ for all $i \in I$. It is defined by $f(x) = f_i(\overline{x})$ for the unique $\overline{x} \in X_i$ with $in_i(\overline{x}) = x$. It may be denoted by $\langle f_i \rangle_{i \in I}$. The property means that a disjoint union of F is a categorial coproduct in the category of sets. Using the property, one can easily show that Y with $(f_i)_{i \in I}$ is a disjoint union of F if and only if $\langle f_i \rangle_{i \in I}$ is bijective. Given a set Y

and a bijection $f\colon X \to Y$, then Y with the injections $(f \circ in_i)_{i\in I}$ is a disjoint union of F provided that X with $(in_i)_{i\in I}$ is one. In other words, the construction of disjoint unions of sets is unique up to bijection.

We use two further nice properties of disjoint unions. The first property is a (de-)composition property: $\sum_{i\in I} X_i = \sum_{i\in I'} X_i + \sum_{i\in I\setminus I'} X_i$ for $I' \subseteq I$. This means in particular that the $+$-operator is commutative and associative. The second property is that inclusions are preserved. Let $F = (X_i)_{i\in I}$ and $F' = (Y_i)_{i\in I}$ be two families of sets and Y with $(in_i^Y)_{i\in I}$ a disjoint union of F'. Let $(g_i\colon X_i \to Y_i)_{i\in I}$ be a family of mappings. Then $\langle in_i^Y \circ g_i\rangle_{i\in I}$ is also denoted by $\sum_{i\in I} g_i$. It is injective if all g_i are injective. It can be chosen as inclusion of $\sum_{i\in I} X_i$ into $Y = \sum_{i\in I} Y_i$ if the g_i are inclusions, i.e. $X_i \subseteq Y_i$ for all $i \in I$.

2.2 Basic Notions of Graphs

Let Σ be a set of labels. A (directed edge-labeled) *graph* over Σ is a system $G = (V, E, s, t, l)$ where V is a set of *nodes*, E is a set of *edges*, $s, t\colon E \to V$ are mappings assigning a *source* $s(e)$ and a *target* $t(e)$ to every edge in E, and $l\colon E \to \Sigma$ is a mapping assigning a label to every edge in E. An edge e with $s(e) = t(e)$ is also called a *loop*. The components V, E, s, t, and l of G are also denoted by V_G, E_G, s_G, t_G, and l_G, respectively. The set of all graphs over Σ is denoted by $\mathcal{G}_\Sigma$.

This notion of graphs is flexible enough to cover other types of graphs. *Simple graphs* form a subclass consisting of those graphs two edges of which are equal if their sources and their targets are equal respectively. A label of a loop can be interpreted as a label of the node to which the loop is attached so that *node-labeled graphs* are covered. We assume a particular label $*$ which is omitted in drawings of graphs. In this way, graphs where all edges are labeled with $*$ may be seen as *unlabeled graphs*. Moreover, *undirected graphs* can be represented by directed graphs if one replaces each undirected edge by two directed edges attached to the same two nodes, but in opposite directions. Finally, *hypergraphs* can be handled by the introduced type of graphs as done explicitly in Sect. 4.

A graph $G \in \mathcal{G}_\Sigma$ is a *subgraph* of a graph $H \in \mathcal{G}_\Sigma$, denoted by $G \subseteq H$, if $V_G \subseteq V_H$, $E_G \subseteq E_H$, $s_G(e) = s_H(e)$, $t_G(e) = t_H(e)$, and $l_G(e) = l_H(e)$ for all $e \in E_G$. In drawings of graphs and subgraphs, shapes, colors, and names are used to indicate the identical nodes and edges.

Given a graph, a subgraph is obtained by removing some nodes and edges subject to the condition that the removal of a node is accompanied by the removal of all its *incident* edges. More formally, let $G = (V, E, s, t, l)$ be a graph and $X = (V_X, E_X) \subseteq (V, E)$ be a pair of sets of nodes and edges. Then $G \setminus X = (V \setminus V_X, E \setminus E_X, s', t', l')$ with $s'(e) = s(e)$, $t'(e) = t(e)$, and $l'(e) = l(e)$ for all $e \in E \setminus E_X$ is a subgraph of G if and only if there is no $e \in E \setminus E_X$ with $s(e) \in V_X$ or $t(e) \in V_X$. This condition is called *dangling condition* of X in G.

For graphs $G, H \in \mathcal{G}_\Sigma$ a *graph morphism* $g\colon G \to H$ is a pair of mappings $g_V\colon V_G \to V_H$ and $g_E\colon E_G \to E_H$ that are structure-preserving, i.e.

$g_V(s_G(e)) = s_H(g_E(e))$, $g_V(t_G(e)) = t_H(g_E(e))$, and $l_H(g_E(e)) = l_G(e)$ for all $e \in E_G$. We may write $g(v)$ and $g(e)$ for nodes $v \in V_G$ and edges $e \in E_G$ since the indices V and E can be reconstructed easily from the type of the argument. If g_V and g_E of a graph morphism $g\colon G \to H$ are bijective, then g is called a *graph isomorphism*. In this case G and H are *isomorphic*, denoted by $G \simeq H$.

For a graph morphism $g\colon G \to H$, the image of G in H is called a *match* of G in H, i.e. the match of G with respect to the morphism g is the subgraph $g(G) \subseteq H$ which is induced by $(g(V), g(E))$.

Given $F \subseteq G$, then the two inclusions of the sets of nodes and edges define a graph morphism. It is also easy to see that the (componentwise) sequential composition of two graph morphisms $f\colon F \to G$ and $g\colon G \to H$ yields a graph morphism $g \circ f\colon F \to H$. Consequently, if f is the inclusion w.r.t. $F \subseteq G$, then $g(F)$ is the match of F in H w.r.t. g restricted to F.

Instead of removing nodes and edges, one may add some nodes and edges to extend a graph such that the given graph is a subgraph of the extension. The addition of nodes causes no problem at all, whereas the addition of edges requires the specification of their labels, sources, and targets, where the latter two may be given or new nodes. Let $G_1 = (V_1, E_1, s_1, t_1, l_1)$ be a graph and $(V_2, E_2, s_2, t_2, l_2)$ be a structure consisting of two sets V_2 and E_2 and three mappings $s_2\colon E_2 \to V_1 + V_2$, $t_2\colon E_2 \to V_1 + V_2$, and $l_2\colon E_2 \to \Sigma$. Then $H = G_1 + (V_2, E_2, s_2, t_2, l_2) = (V_1 + V_2, E_1 + E_2, \langle \hat{s}_1, s_2 \rangle, \langle \hat{t}_1, t_2 \rangle, \langle l_1, l_2 \rangle)$ is a graph with $G_1 \subseteq H$ where $\hat{s}_1, \hat{t}_1\colon E_1 \to V_1 + V_2$ with $\hat{s}_1(e) = s_1(e), \hat{t}_1(e) = t_1(e)$ for all $e \in E_1$.

Let $G = (G_i)_{i \in I}$ be a family of graphs. Then the *disjoint union* of G is defined by $\sum_{i \in I} G_i = (\sum_{i \in I} V_{G_i}, \sum_{i \in I} E_{G_i}, \sum_{i \in I} s_{G_i}, \sum_{i \in I} t_{G_i}, \langle l_{G_i} \rangle_{i \in I})$. The construction has all the properties summarized in Sect. 2.1 for the disjoint union of sets if one replaces the term *set* by *graph* (with the exception of the index set), *subset* by *subgraph* and *mapping* by *graph morphism*.

2.3 Rule-Based Graph Transformation

Formally, a *rule* $r = (L \supseteq K \subseteq R)$ consists of three graphs $L, K, R \in \mathcal{G}_\Sigma$ such that K is a subgraph of L and R. The components L, K, and R of r are called *left-hand side*, *gluing graph*, and *right-hand side*, respectively.

The application of a graph transformation rule to a graph G consists of replacing a match of the left-hand side in G by the right-hand side such that the match of the gluing graph is kept. Hence, the application of $r = (L \supseteq K \subseteq R)$ to a graph $G = (V, E, s, t, l)$ comprises the following three steps.

First, a graph morphism $g\colon L \to G$ called *matching morphism* is chosen to establish a match of L in G subject to the *gluing condition* consisting of two parts: (a) the dangling condition of $g(L) \setminus g(K) = (g(V_L) \setminus g(V_K), g(E_L) \setminus g(E_K))$ in G; and (b) the identification condition requesting that two nodes or edges of L must be in K if they are identified in the match of L.

Second, the match of L up to $g(K)$ is removed from G, resulting is the intermediate graph $Z = G \setminus (g(L) \setminus g(K))$.

Third, the right-hand side R is added to Z by gluing Z with R in $g(K)$ yielding the graph $H = Z + (R \setminus K, g)$ where $(R \setminus K, g) = (V_R \setminus V_K, E_R \setminus E_K, s', t', l')$ with $s'(e') = s_R(e')$ if $s_R(e') \in V_R \setminus V_K$ and $s'(e') = g(s_R(e'))$ otherwise, $t'(e') = t_R(e')$ if $t_R(e') \in V_R \setminus V_K$ and $t'(e') = g(t_R(e'))$ otherwise, and $l'(e') = l_R(e')$ for all $e' \in E_R \setminus E_K$.

The extension of Z to H is properly defined because s' and t' map the edges of $E_R \setminus E_K$ into nodes of $V_R \setminus V_K$ or $g(V_K)$ which is part of V_Z. As the disjoint union is only unique up to isomorphism, the resulting graph is only unique up to isomorphism. Due to the construction, g can be restricted to $d\colon K \to Z$, and d can be extended to a right matching morphism $h\colon R \to H$ by the identity on $R \setminus K$.

Hence a rule application of r can be depicted by the following diagram.

$$\begin{array}{ccccc} L & \supseteq & K & \subseteq & R \\ \downarrow g & & \downarrow d & & \downarrow h \\ G & \supseteq & Z & \subseteq & H \end{array}$$

It is worth noting that both squares of the diagram are pushouts in the category of graphs if the subgraph relations in the diagram are interpreted as inclusion morphisms. Therefore, an equivalent definition based on double pushouts in category theory can be found in, e.g., [2]. Here the identification condition is significant because the left diagram is not a pushout if g does not obey the identification condition.

The application of a rule r to a graph G is denoted by $G \underset{r}{\Longrightarrow} H$. A rule application is called a *direct derivation*, and an iteration of direct derivations $G \simeq G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} G_n \simeq H$ $(n \in \mathbb{N})$ is called a *derivation* from G to H. As usual, the derivation from G to H can also be denoted by $G \overset{n}{\underset{P}{\Longrightarrow}} H$ where $\{r_1, \ldots, r_n\} \subseteq P$, or by $G \overset{*}{\underset{P}{\Longrightarrow}} H$ if the number of direct derivations is not of interest. The subscript P may be omitted.

As the disjoint union is only uniquely defined up to isomorphism, derived graphs are also only uniquely constructed up to isomorphism. But without loss of generality, one can assume that nodes and edges, which are not removed, keep their identity. We make use of this fact in all our explicit examples.

A *graph class expression* may be any syntactic entity X that specifies a class of graphs $SEM(X) \subseteq \mathcal{G}_\Sigma$ like expressions or formula. A *control condition* may be any syntactic entity that restricts the derivation process. Explicit examples are introduced where they are needed.

A *graph transformation unit* is a system $gtu = (I, P, C, T)$ where I and T are graph class expressions to specify the *initial* and the *terminal* graphs respectively, P is a set of rules, and C is a control condition. Such a transformation unit specifies a binary relation $SEM(gtu) \subseteq \mathcal{G}_\Sigma \times \mathcal{G}_\Sigma$ that contains a pair (G, H) if and only if $(G, H) \in SEM(I) \times SEM(T)$ and there is a derivation $G \overset{*}{\underset{P}{\Longrightarrow}} H$ permitted by C.

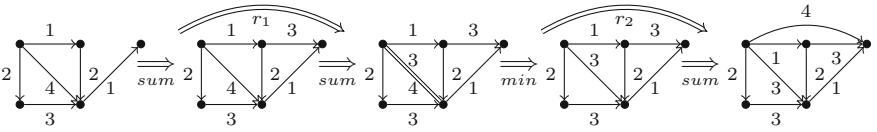
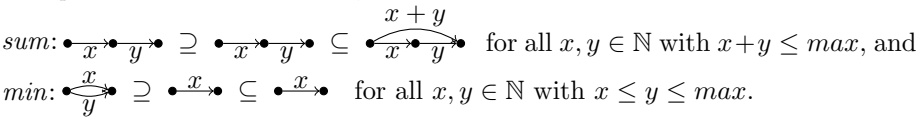


Fig. 1. A derivation based on the *shortest_paths*(*max*) graph transformation unit

Example 1. For some $max \in \mathbb{N}$, consider all rules of the form:



Given a graph with labels in $\mathbb{N}$, the application of a *sum*-rule adds an edge bridging a path of length 2 and summing up the labels of the path. A *min*-rule is applicable to each two parallel edges, keeping the edge with the smaller label or one of the two if the labels are equal. A sample derivation can be seen in Fig. 1. (The derivation applying r_1 and r_2 is explained in Sect. 3.1.) If the two edges of the left-hand side match a single edge, then the identification condition is not satisfied. The dangling condition is always satisfied because nodes are never removed. But if one modifies the *sum*–rule in such a way that the middle node and the edges of the gluing graph are omitted, then the dangling condition is not satisfied whenever the middle node is attached to more than two edges.

The *sum*–rule can be applied to each path of length 2 arbitrarily often. This can be avoided if one requires that there is no bridging edge with a label $z \leq x+y$ in the accessed graph. Such a negative context condition is an example of a control condition. The rules together with this control condition specify a graph transformation unit if one chooses proper initial and terminal graphs in addition. The constant expressions *loop-free*, *strictly-simple* and *max-labeled* denote the classes of graphs without loops; with at most one edge between every two nodes; and with labels in $\mathbb{N}$ whose sum does not exceed *max*, respectively. Combined by &, one gets the intersection of the three classes. Then the expression 0-*looped*(*max-labeled* & *strictly-simple* & *loop-free*) defines the graphs in the intersection with a 0-loop at each node in addition. Starting with these graphs as initial graphs and applying the rules according to the control condition as long as possible, results in graphs where each edge between nodes v and v' is labeled with the shortest distance between v and v' in the respective initial graph, i.e. the minimum label sum of all paths from v to v'. The terminal graphs can be specified by the expressions $\{sum, min\}$-*reduced*. In summary, it is justified to call the unit *shortest_paths*(*max*). It is schematically given in Fig. 2. The components of the unit are preceded with respective keywords, the negative context condition of the *sum*-rule is denoted by the dashed edge.

The example is further discussed in Sect. 5.

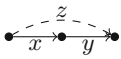
shortest_paths(*max*)
initial: 0-*looped*(*max-labeled* & *strictly-simple* & *loop-free*)
rules: *sum*, *min*
control:  with $z \leq x + y$ wrt *sum*
terminal: {*sum*, *min*}-*reduced*

Fig. 2. The graph transformation unit *shortest_paths*(*max*)

3 Parallel Graph Transformation

In this section, we recall the notion of parallel graph transformation and its fundamental properties of sequentialization and parallelization as introduced in [1] only slightly modified. Our rules consist of two inclusions each while a rule in [1] consists of an injective graph morphism from the gluing graph to the left-hand side graph and an arbitrary graph morphism from the gluing graph into the right-hand side graph. Moreover, the parralel rule in [1] is a disjoint union of two rules while we consider parallel rules composed of an arbitrary family of component rules.

Definition 1. Let $F = (r_i)_{i \in I} = (L_i \supseteq K_i \subseteq R_i)_{i \in I}$ be a family of rules. Then the *parallel rule* of F is defined by $r(F) = \sum_{i \in I} r_i = (\sum_{i \in I} L_i \supseteq \sum_{i \in I} K_i \subseteq \sum_{i \in I} R_i)$.

Due to the properties of disjoint unions, the parallel rule is an ordinary rule so that parallel derivations are just derivations applying parallel rules.

3.1 Sequentialization and Parallelization Theorems

Let $G \underset{r_1+r_2}{\Longrightarrow} X$ be a direct parallel derivation, let $g\colon L_1 + L_2 \to G$ be the corresponding matching morphism, and let $in_1\colon L_1 \to L_1 + L_2$ be the inclusion of L_1 into $L_1 + L_2$. Then $g_1 = g \circ in_1$ defines a matching morphism of L_1 into G. It is easy to see that g_1 satisfies the gluing condition using the satisfaction of the gluing condition of g. This yields a direct derivation $G \underset{r_1}{\Longrightarrow} H_1$. Let Z_1 be its intermediate graph. Then the identification condition satisfied by g yields $g(L_2) \subseteq Z_1$. This allows one to define a matching morphism g_2' of L_2 into H_1 by $g_2'(x) = g(x)$ for all x of L_2. Using again the gluing condition satisfied by g, it turns out that g_2' satisfies the gluing condition and yields a direct derivation $H_1 \underset{r_2}{\Longrightarrow} X_1$. Finally, one can show by the construction of direct derivations and some basic properties of union and difference of sets that X and X_1 are isomorphic. Altogether, the reasoning yields the following result.

Theorem 1 (Sequentialization of parallel derivations). *Let r_1, r_2 be rules and $G \underset{r_1+r_2}{\Longrightarrow} X$ be a direct derivation. Then there is a derivation $G \underset{r_1}{\Longrightarrow} H_1 \underset{r_2}{\Longrightarrow} X$.*

The resulting derivation is called the *sequentialization* of $G \underset{r_1+r_2}{\Longrightarrow} X$. We also get $G \underset{r_2}{\Longrightarrow} H_2 \underset{r_1}{\Longrightarrow} X$ as $r_1 + r_2 = r_2 + r_1$. The identification condition satisfied by the given matching morphism $g\colon L_1 + L_2 \to G$ implies for the two matching morphims g_1 and g_2 which restrict g to the components L_1 and L_2 that $g_1(L_1) \cap g_2(L_2) \subseteq g_1(K_1) \cap g_2(K_2)$, i.e. the two matches may overlap, but only in common gluing elements. A further analysis yields for the right matching morphism $h_1\colon R_1 \to H_1$ of $G \underset{r_1}{\Longrightarrow} H_1$ and the matching morphism $g_2'\colon L_2 \to H_1$ constructed above: $h_1(R_1) \cap g_2'(L_2) \subseteq h_1(K_1) \cap g_2'(K_2)$. The two properties are called parallel and sequential independence respectively. Independence refers to the fact that the application of one of the two rules does not prevent or influence the application of the other one. Nicely enough, independence is sufficient for parallelization.

Definition 2. Let $r_i = (L_i \supseteq K_i \subseteq R_i)$ for $i = 1, 2$ be rules.

1. Two direct derivations $G \underset{r_i}{\Longrightarrow} H_i$ with the matching morphism $g_i\colon L_i \to G$ respectively are *parallel independent* if $g_1(L_1) \cap g_2(L_2) \subseteq g_1(K_1) \cap g_2(K_2)$.
2. Successive direct derivations $G \underset{r_1}{\Longrightarrow} H_1 \underset{r_2}{\Longrightarrow} X$ with the right matching morphism $h_1\colon R_1 \to H_1$ and the (left) matching morphism $g_2'\colon L_2 \to H_1$ are *sequentially independent* if $h_1(R_1) \cap g_2'(L_2) \subseteq h_1(K_1) \cap g_2'(K_2)$.

Theorem 2 (Parallelization of independent direct derivations). *Let $r_i = (L_i \supseteq K_i \subseteq R_i)$ for $i = 1, 2$ be rules.*

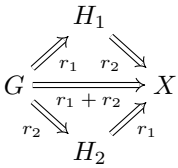
1. *Let $G \underset{r_i}{\Longrightarrow} H_i$ for $i = 1, 2$ be parallel independent direct derivations with matching morphisms $g_i\colon L_i \to G$. Then there is a direct parallel derivation $G \underset{r_1+r_2}{\Longrightarrow} X$ for some $X \in \mathcal{G}_\Sigma$ with matching morphism $\langle g_1, g_2\rangle\colon L_1 + L_2 \to G$.*
2. *Let $G \underset{r_1}{\Longrightarrow} H_1 \underset{r_2}{\Longrightarrow} X$ be sequentially independent direct derivations. Then there is a parallel derivation $G \underset{r_1+r_2}{\Longrightarrow} X$.*

The first direct derivations of the two possible sequentializations of the constructed direct parallel derivation in Point 1 coincide with the given direct derivations.

Let $g_1\colon L_1 \to G$ and $g_2'\colon L_2 \to H_1$ be the matching morphisms of the given direct derivation in Point 2. Then the sequential independence and the construction of direct derivations yield $g_2'(L_2) \subseteq G$. This allows to define a matching morphism $g_2\colon L_2 \to G$. Then the direct derivation $G \underset{r_1+r_2}{\Longrightarrow} X$ is given by the matching morphims $\langle g_1, g_2\rangle\colon L_1 + L_2 \to G$. $G \underset{r_1+r_2}{\Longrightarrow} X$ is called *parallelization* of $G \underset{r_1}{\Longrightarrow} H_1 \underset{r_2}{\Longrightarrow} X$.

Example 2. Look at the derivation in Fig. 1. The first two steps are sequentially independent as the second step does not match the edge generated by the first step. Moreover, the last two steps are also sequentially independent so that the two possible parallelizations yield the derivation applying $r_1 = sum + sum$ and $r_2 = min + sum$ in Fig. 1.

The sequentialization and parallelization theorems involve the following derivations from G to X where the two direct derivations $G \underset{r_1}{\Longrightarrow} H_1$ and $G \underset{r_2}{\Longrightarrow} H_2$ are parallel independent and both derivations $G \underset{r_1}{\Longrightarrow} H_1 \underset{r_2}{\Longrightarrow} X$ and $G \underset{r_2}{\Longrightarrow} H_2 \underset{r_1}{\Longrightarrow} X$ are sequentially independent.



The whole diagram is obtained (a) from the direct parallel derivation $G \underset{r_1+r_2}{\Longrightarrow} X$, (b) from the two parallel independent direct derivations $G \underset{r_1}{\Longrightarrow} H_1$ and $G \underset{r_2}{\Longrightarrow} H_2$, or (c) from each of the sequentially independent derivations from G to X.

As pointed out in, e.g., [3], the diagram reflects the concurrency of two events: One may happen after the other or the other way round or both may happen simultaneously. All three ways to move from G to X are equally possible. There is neither a causal dependence nor any mutual influence.

The results are particularly significant with respect to the construction of matching morphisms which is the most time-consuming part of a rule application. Whether a graph morphism from L to G exists, is a well-known NP-complete problem if L and G are finite input graphs of arbitrary size. Hence, all known algorithms that find graph morphisms for finite, but arbitrary large L and G are exponential. In contrast to that, the search for matching morphisms becomes polynomial in the size of G if L is fixed or the size of L is bounded by a constant. This is the case if one assumes finite sets of finite rules. The number of mappings from a set with k elements to a set with n elements is n^k so that one can check all possible matchings in polynomial time even in an exhaustive search. This applies in particular to the case of finite sets of finite rules. But it does not apply to parallel rules because their left-hand sides may become arbitrary large so that one would have to face the problem of NP-completeness without further knowledge. Fortunately, we know that the matching morphism of a parallel rule is composed of matching morphisms of the atomic component rules so that matching morphism for parallel rules can be found in polynomial time if the number of components is polynomial or the components can be processed in parallel.

3.2 Shifts and Canonical Derivations

The three derivations from G to X in the diagram above may be considered as equivalent from a concurrency point of view. Further, this view can be extended to arbitrary derivations so that the equivalence classes represent concurrent processes. But the equivalence classes may be exponentially large. In order to give an efficient representation a shift operation can be defined as a certain combination of sequentialization and parallelization such that shifting as long as possible yields unique canonical representatives.

Let $s = G_0 \overset{*}{\Longrightarrow} G_i \underset{r_1+r_2}{\Longrightarrow} G_{i+2} \overset{*}{\Longrightarrow} G_n$ and $s' = G_0 \overset{*}{\Longrightarrow} G_i \underset{r_1}{\Longrightarrow} G_{i+1} \underset{r_2}{\Longrightarrow} G_{i+2} \overset{*}{\Longrightarrow} G_n$ be two derivations where $G_i \underset{r_1}{\Longrightarrow} G_{i+1} \underset{r_2}{\Longrightarrow} G_{i+2}$ is a sequentialization of $G_i \underset{r_1+r_2}{\Longrightarrow} G_{i+2}$. Then s is *seq-related* to s' denoted by $s \underset{seq}{\rightarrow} s'$. The equivalence closure is denoted by $\sim$.

Let us restrict the consideration to parallel derivations where only parallel rules with a finite number of component rules are applied. Then the equivalence classes are always finite, but they may have an exponential number of elements. Let s, s' and s'' be three derivations with $s \underset{seq}{\rightarrow} s'$ and $s'' \underset{seq}{\rightarrow} s'$ of the form

$$G_0 \overset{*}{\Longrightarrow} G_{i-1} \underset{r_1}{\Longrightarrow} G_i \underset{r_2}{\Longrightarrow} G_{i+1} \underset{r_3}{\Longrightarrow} G_{i+2} \overset{*}{\Longrightarrow} G_n$$

with $r_1 + r_2$ spanning G_{i-1} to G_{i+1} and $r_2 + r_3$ spanning G_i to G_{i+2}

where s is the derivation with $r_1 + r_2$, s'' is the derivation with $r_2 + r_3$, and s' is the derivation in the middle. Then s'' is *shift-related* to s, denoted by $s'' \underset{shift}{\rightarrow} s$. Moreover, if only s and s' are given with $s \underset{seq}{\rightarrow} s'$, then s' is also *shift-related* to s, denoted by $s' \underset{shift}{\rightarrow} s$. A rule applied in step $i + 1$ can only be shifted if its direct derivation is sequentially independent of the preceding direct derivation, and it can be shifted i times at most. Therefore, *shift*-sequences are never longer than $n(n-1)/2$ if n is the number of applications of atomic rules. In particular, one gets always *shift*-reduced derivations if one shifts as long as possible. These *shift*-reduced derivations are called *canonical* because they are unique representatives of their equivalence classes.

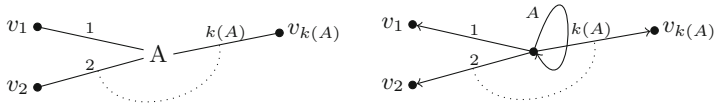
Theorem 3. *Let s and s' be two canonical derivations with $s \sim s'$, then $s = s'$.*

The proof is based on the fact, that the *shift*-relation is locally Church-Rosser: Given $s \underset{shift}{\rightarrow} s'$ and $s \underset{shift}{\rightarrow} s''$, then there is $\overline{s}$ with $s' \overset{*}{\underset{shift}{\rightarrow}} \overline{s}$ and $s \overset{*}{\underset{shift}{\rightarrow}} \overline{s}$.

Example 3. The derivation applying r_1 and r_2 in Fig. 1 is canonical.

3.3 Related Work

In [1] and in the present paper, the proofs of the stated results are only roughly sketched. The full proofs can be found in [2,4]. In the last 40 years, the topic of parallel graph transformation has been further studied by many researchers in various respects modifying and generalizing the approach. As it is impossible to refer to all related publications – there are too many – the reader may consult Volume 3 of the Handbook of Graph Grammars and Computing by Graph Transformation [5] and the two monographs [6,7] where much of the work is systematically presented in the context of the double-pushout approach, and the important references are given in the introductions of the books and of the respective chapters. This covers nicely typed attributed graphs, high-level replacement systems in adhesive categories as well as concurrent and amalgamated rules. Concerning the single- and the sesqui-pushout approaches, the reader is referred to [8,9].



(a) Hyperedge (b) Its graph representation

Fig. 3. The graph representation of a hyperedge

4 Parallelism of Hyperedge Replacement

Hyperedge replacement (see, e.g., [10–13]) is a kind of context-free hypergraph transformation. Hyperedges of hypergraphs may be incident to arbitrary sequences of nodes rather than to two nodes as ordinary edges. But there is a straightforward way to formulate hyperedge replacement within the graph setting introduced in Sect. 2 (cf. [14]). With respect to parallelism, hyperedge replacement is of interest in at least two ways.

First, the context-freeness lemma provides a decomposition of derivations into a set of fibers that corresponds to the decomposition of a direct parallel derivation into direct derivations applying the rule components.

Second, due to the sequentialization theorem, sequential and parallel hyperedge replacement have the same generative power. But if one applies hyperedge replacement rules in the mode of TOL-systems, then one can get quite different languages.

4.1 Hyperedge Replacement and Its Context-Freeness Lemma

We assume some subset $N \subseteq \Sigma$ of *nonterminals* which are typed, i.e. there is an integer $k(A) \in \mathbb{N}$ for each $A \in N$. Moreover, we assume that Σ contains the numbers $1, \ldots, max$ for some $max \in \mathbb{N}$ with $k(A) \leq max$ for all $A \in N$. A *hyperedge* with label $A \in N$ is meant to be an atomic item which is attached to a sequence of nodes $v_1 \cdots v_{k(A)}$. It can be represented by a node with an A-labeled loop and $k(A)$ edges the labels of which are $1, \ldots, k(A)$, respectively, and the targets of which are $v_1, \ldots, v_{k(A)}$, respectively, as depicted in Fig. 3. Accordingly, we call such a node with its incident edges an *A-hyperedge*. A graph is said to be *N-proper* if each occurring nonterminal and each occuring number between 1 and max belong to some hyperedge. Each $A \in N$ induces a particular N-proper graph $A^\bullet$ with the nodes $\{0, \ldots, k(A)\}$ and a single hyperedge where the A-loop is attached to 0 and i is the target of the edge labeled with i for $i = 1, \ldots, k(A)$. Let $[k(A)]$ denote the discrete graph with the nodes $\{1, \ldots, k(A)\}$. Using these notations, a rule of the form $A^\bullet \supseteq [k(A)] \subseteq R$ for some N-proper graph R is a *hyperedge replacement rule*, which can be denoted by $A ::= R$ for short. A *hyperedge replacement grammar* is a system $HRG = (N, T, P, S)$ with $S \in N$, $T \subseteq \Sigma$ with $T \cap N = \emptyset$, and a set of hyperedge replacement rules P with finite right-hand sides. Its generated language contains all terminal graphs that are derivable from $S^\bullet$, i.e. $L(HRG) = \{H \in \mathcal{G}_T \mid S^\bullet \overset{*}{\underset{P}{\Longrightarrow}} H\}$.

In this way, hyperedge replacement is just a special case of graph transformation, but with some very nice properties.

1. Let $r = (A ::= R)$ be a hyperedge replacement rule and G an N-proper graph with an A-hyperedge y. Then there is a unique graph morphism $g\colon A^\bullet \to G$ mapping $A^\bullet$ to the A-hyperedge y such that the gluing condition is satisfied and therefore r is applicable to G.
2. The directly derived graph H is N-proper and is obtained by removing y, i.e. by removing the node with the A-loop and all other incident edges, and by adding R up to the nodes $1, \ldots, k(A)$ where edges of R incident to $1, \ldots, k(A)$ are redirected to $g(1), \ldots, g(k(A))$, respectively. Due to this construction, H may be denoted by $G[y/R]$.
3. Two direct derivations $G \underset{r_1}{\Longrightarrow} H_1$ and $G \underset{r_2}{\Longrightarrow} H_2$ are parallel independent if and only if they replace distinct hyperedges.
4. A parallel rule $r = \sum_{i \in I} r_i$ of hyperedge replacement rules $r_i = (A_i ::= R_i)$ for $i \in I$ is applicable to G if and only if there are pairwise distinct A_i-hyperedges y_i for all $i \in I$. In analogy to the application of a single rule, the resulting graph may be denoted by $G[y_i/R_i \mid i \in I]$.
5. If $I = I_1 + I_2$, then $G[y_i/R_i \mid i \in I] = (G[y_i/R_i \mid i \in I_1])[y_i/R_i \mid i \in I_2]$.
6. Two successive direct derivations $G \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} H$ are sequentially independent if and only if the hyperedge replaced by the second step is not created by the first one.

Altogether, the direct derivations through hyperedge replacement rules can be ordered arbitrarily as long as they deal with different hyperedges. This observation leads to the following result.

Theorem 4 (Context-Freeness Lemma). *Let $HRG = (N, T, P, S)$ be a hyperedge replacement grammar and let $A^\bullet \overset{n+1}{\underset{P}{\Longrightarrow}} H$ be a derivation. Then there are some rule $A ::= R$ and a derivation $A(y)^\bullet \overset{n(y)}{\underset{P}{\Longrightarrow}} H(y)$ for each hyperedge y of R with label $A(y)$ such that $H = R[y/H(y) \mid y \in Y_R]$ and $\sum_{y \in Y_R} n(y) = n$, where Y_R is the set of hyperedges of R.*

A derivation $A^\bullet \overset{n+1}{\Longrightarrow} H$ has $A^\bullet \Longrightarrow R$ as the first step. The tail $R \overset{n}{\Longrightarrow} H$ can be decomposed into fibers $A(y)^\bullet \overset{n_i}{\Longrightarrow} H(y)$ for $y \in Y_R$. The fibres induce rules $A(y) ::= H(y)$ for $y \in Y_R$. They can be applied to R in parallel yielding $R \Longrightarrow H$. In this way, hyperedge replacement allows to generalize the sequentialization and parallelization of direct derivations to derivations.

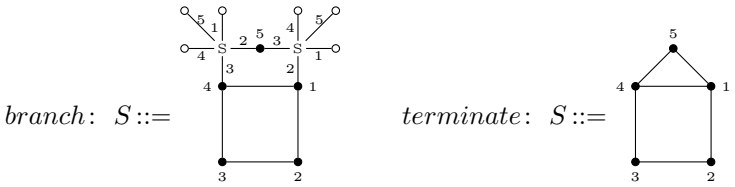
4.2 Maximum Parallel Hyperedge Replacement

Given a hyperedge replacement grammar $HRG = (N, T, P, S)$ and $H \in L(HRG)$. Then there is a derivation $S^\bullet \overset{*}{\Longrightarrow} H$ which can be transformed into a canonical derivation. As the replacements of two hyperedges are parallel independent and

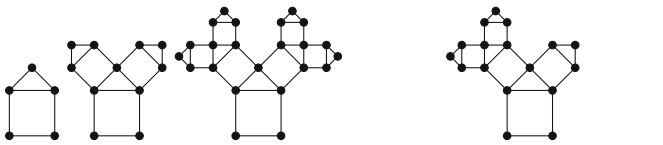
as H is terminal, each direct derivation of the canonical derivation replaces all hyperedges in parallel. This means that canonical derivations are maximum parallel normal forms to generate $L(HRG)$, but maximum parallelism does not extend or vary the generative power.

This changes if the set of rules is partitioned into subsets $P_1, \ldots, P_k$ for some $k \geq 1$ and each direct derivation takes one of them and applies the rules with maximum parallelism, i.e. in the style of TOL- and ETOL-systems (see, e.g., Chap. 5 in [15]). The TOL-mode of hyperedge replacement is a further example of a control condition. It allows to generate all languages that are generated by ordinary hyperedge replacement grammars because one can choose $P_1 = P$. But it also increases the generative power which is proved by a separating example.

Example 4. Consider the hyperedge replacement grammar $KOCHTREE = (\{S\}, \{*\}, P, S)$ where S has type 5 and P contains two rules:



If one decomposes P into $\{branch\}$ and $\{terminate\}$ and applies one or the other with maximum parallelism, one gets very regular finite approximations of the Koch tree (depicted in Fig. 4a). If one applies the rules arbitrarily, then one can also get asymmetric trees (like the one depicted in Fig. 4b). As long as the rule *branch* is used, the number of hyperedges doubles and each hyperedge replacement produces 4 (undirected) edges such that the language of regular Koch trees grows exponentially. On the other hand, it is a well-known fact that the languages generated by ordinary hyperedge replacement grammars have a sublinear growth so that they cannot generate regular Koch trees. Altogether, this shows that hyperedge replacement grammars with a TOL-mode of transformation are more powerful than without.



(a) Regular Koch trees (b) Asymmetric Koch tree

Fig. 4. Approximations of the Koch tree

4.3 Related Work

While hyperedge replacement is a well-studied area of graph transformation, we are not aware of much work on parallel generation of graph languages. But we

would like to mention that we introduced recently fusion grammars in [16] that display quite strong parallelization properties and extend the generative power of hyperedge replacement grammars.

5 Parallelization of Graph Algorithms

A major promise of parallelism is that parallel computation can be faster than sequential computation. Consider, for example, a totally balanced binary tree of height n. It has 2^n leaves, and therefore, a full traversal needs at least this many steps whereas traversing the tree from the root to the leaves with maximum parallelism takes n steps. So it seems worthwhile to look into graph transformation whether the use of parallel derivations can produce similar effects. To demonstrate the potential of this line of research, we look into the well-known search for shortest paths.

5.1 The Case of Shortest Paths

Most shortest-path algorithms like the prominent ones by Floyd/Warshall [17,18] and by Dijkstra [19] are based on two elementary operations: the sequential composition of paths summing up the distances and keeping the path with minimum distance out of some parallel paths (i.e. paths with the same source and target nodes). The algorithms differ from each other by the order in which the two basic operations are applied.

Let us reconsider the graph transformation unit *shortest_paths*(max) in Example 1. To make sure that the unit computes shortest distances, the following correctness properties can be proved. The distance of a path p in a graph G is the sum of the distances of the edges on p and is denoted by $dist_G(p)$.

Proposition 1 (Correctness). Let $G \stackrel{*}{\Longrightarrow} H$ be a derivation where G is initial and H is terminal. Then the following hold:

1. For every shortest path p from v to v' in G, there is some $e \in E_H$ with $s_H(e) = v$, $t_H(e) = v'$, and $l_H(e) = dist_G(p)$.
2. For every $e \in E_H$, there is a shortest path p from $s_H(e)$ to $t_H(e)$ in G with $l_H(e) = dist_G(p)$.

The first statement can be proved by induction on the length of shortest paths, the second one by induction on the length of derivations. The details are omitted for reasons of space limitations.

Now we consider the parallelization of the algorithm. The graph transformation unit *shortest_paths_in_parallel*(max) extends the unit *shortest_paths*(max) by the control condition

$$(sum[double{-}free\ maxpar]; min[largest\ maxpar])^*.$$

It requires that, repeatedly, the *sum*-rule is applied with double-free maximum parallelism followed by the largest maximum parallel application of the *min*-rule.

In a double-free parallel rule application of *sum*, no two matches of left-hand sides may overlap entirely. A largest parallel rule application of *min* must involve as many *min*-rules as possible.

As the left-hand side of the *sum*-rule coincides with the gluing graph, each two applications of *sum* are parallel independent. Therefore, there are at most $n \cdot (n-1) \cdot (n-2)$ double-free applications of *sum* where n is the number of nodes in the initial graph. The following largest maximum parallel *min*-step makes sure that no two parallel edges are left. More precisely, two *min*-applications are parallel independent if they match four different edges or intersect in the edge that is kept. Therefore, whenever there are m parallel edges between two nodes, the largest parallel step removes $m-1$ of them, and this happens if all applications of *min* choose the same edge to be kept.

That the unit computes the shortest distances between each two nodes can be seen as follows. The initial and terminal graphs are the same as in the unit *shortest_paths*(*max*) above. A parallel derivation from an initial to a terminal graph can be sequentialized due to the sequentialization theorem. In this sequential derivation, a *sum*-application may occur that does not obey the negative application condition. But then there is already an edge as good as or better than the edge generated by *sum*. Hence, this step as well as the *min*-step that removes this superfluous edge later on can be omitted without changing the result. If the sequential derivation is modified in this way as long as possible, then we end up with a derivation from an initial to a terminal graph in *shortest_paths*(*max*). Hence the correctness of *shortest_paths_in_parallel*(*max*) follows from the correctness of *shortest_paths*(*max*).

A closer look reveals that the edges after k rounds of a parallel *sum*-step followed by a parallel *min*-step represent the shortest paths of the initial graph of lengths up to 2^k. This implies that after a logarithmic number of parallel steps the terminal graph is reached.

Proposition 2 (Correctness and derivation length). Let $G \overset{2k}{\Longrightarrow} H$ be a derivation in *shortest_paths_in_parallel*(*max*) from an initial graph to a terminal graph with alternating parallel *sum*- and *min*-steps according to the control condition. Then Points 1 and 2 of Proposition 1 hold as well, and the length of the derivation has a logarithmic bound, i.e., $2^k \leq n-1$ where n is the number of nodes in G.

In a similar way, well-known shortest paths algorithms can be parallelized. For example, the parallelization of Mahr's algorithm [20] (which originally is of the order $n^3 \cdot \log n$) yields a logarithmic number of parallel steps and the parallelization of the Floyd/Warshall algorithm (which originally is a cubic algorithm) yields a linear number of parallel steps. But it should be noted that the short parallel derivations do not improve the complexity automatically, but only if the matching of the parallel rules can be found in a time bound that is – multiplied by the logarithmic length of the derivations – still smaller than the complexity of the corresponding sequential algorithms.

5.2 Related Work

There is not much work on parallel and distributed algorithms employing graph transformation. A noteworthy exception is the modeling of distributed algorithms by means of graph relabelling (see, e.g., [21]). Moreover, we would like to mention graph-multiset transformation (see [22]) that can be interpreted as a special case of the parallel graph transformation of Sect. 3 and allows to solve NP-complete graph problems by parallel computations of polynomial lengths. On the other hand, there is a realm of literature on parallel graph algorithm and very much interest in this topic so that the area seems to invite further and deeper studies by means of graph transformation.

6 Infinity

The definition of parallel graph transformation in Sect. 3 includes the case of parallel rules of an infinite family of component rules. In this section, we indicate that such infinite parallel rules may have some potential in the context of infinite graph theory (see, e.g., [23]) but only if one applies them to infinite graphs.

6.1 Application to Finite Graphs

Let $F = (r_i)_{i \in I} = (L_i \supseteq K_i \subseteq R_i)_{i \in I}$ be a family of rules for an infinite index set I. Let G be a finite graph and $G \underset{r(F)}{\Longrightarrow} H$ be an application of the parallel rule of F to G with the matching morphism $g = \langle g_i \rangle_{i \in I} \colon \sum_{i \in I} L_i \to G$. Then the definition of rule application reveals the following facts:

1. Let $I' = \{i \in I \mid K_i \neq L_i\}$ be the set of indices of erasing rules. Then I' is finite because otherwise g would not obey the identification condition. Therefore, $K_i = L_i$ for almost all $i \in I$.
2. Let $I'' = \{i \in I \mid K_i \neq R_i\}$ be the set of indices of adding rules. Then H is finite if and only if I'' is finite.
3. Let I'' be infinite. Then H contains an infinite number of finite subgraphs that are pairwise disjoint or H has a node with infinite degree or both is the case.

Infinite graphs consisting of infinitely many finite disjoint components or with nodes of infinite degree are considered as less interesting in finite graph theory. Therefore, the application of parallel rules of an infinite family of rules can make more sense only if one applies them to infinite graphs.

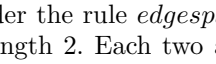
6.2 Application to Infinite Graphs

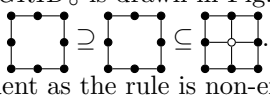
We are not going to study the application of parallel rules of an infinite family of rules to infinite graphs in any depth. But we would like to give an example that displays an interesting property and nourishes the hope that infinite graph transformation may be of interest.

(a) A finite section of *GRID* (b) A finite section of $GRID_\circ$

Fig. 5. Infinite plane grids

Example 5. Consider the infinite plane grid a finite section of which looks like the structure in Fig. 5a. Nodes are the points in the plane with integer coordinates. Each node has four outgoing edges to its Northern, Eastern, Southern, and Western neighbor respectively. As each node is neighbor of its four neighbors, the edges can be drawn as undirected edges. Formally, it can be defined as follows: $GRID = (\mathbb{Z} \times \mathbb{Z}, \mathbb{Z} \times \mathbb{Z} \times \{N, E, S, W\}, s_{GRID}, t_{GRID}, l_{GRID})$ with $s_{GRID}((x, y, D)) = (x, y)$, $t_{GRID}((x, y, N)) = (x, y + 1)$, $t_{GRID}((x, y, E)) = (x + 1, y)$, $t_{GRID}((x, y, S)) = (x, y - 1)$, $t_{GRID}((x, y, W)) = (x - 1, y)$, and $l_{GRID}((x, y, D)) = *$ for all $(x, y) \in \mathbb{Z} \times \mathbb{Z}$ and $D \in \{N, E, S, W\}$.

Consider the rule *edgesplit*:  that splits an edge into a path of length 2. Each two applications of the rule are parallel independent if they match different edges. Therefore, one can apply the rules to all edges in parallel. A finite section of the result $GRID_\circ$ is drawn in Fig. 5b.

Consider now the rule *squaresplit*:  . Each two applications of this rule are parallel independent as the rule is non-erasing. Hence, one can apply the rule to all smallest squares (the cycles of length 8) of $GRID_\circ$ in parallel – one rule per square. Then the result is obviously isomorphic to *GRID*.

This kind of self-replication is remarkable as the applied rules are strictly monotonously growing in that they add more than they erase. Such a property is impossible in the context of finite graphs. Hence it may be worthwhile to study infinite graph transformation in more detail and depth.

7 Parallel Models of Computation

Parallel graph transformation is well-suited for modeling and analyzing parallel processes and, in particular, as a domain into which other visual approaches to parallel processing can be transformed. To demonstrate this, we model the well-known cellular automata as graph transformation units with massive parallelism.

7.1 The Case of Cellular Automata

Cellular automata are computational devices with massive parallelism known for many decades (see, e.g., [24–27]). A cellular automaton is a network of cells where each cell has got certain neighbor cells. A configuration is given by a mapping that associates a local state with each cell. A current configuration can change into a follow-up configuration by the simultaneous changes of all local states. The local transitions are specified by an underlying finite automaton where the local states of the neighbor cells are the inputs. If the network is infinite, one assumes a particular sleeping state that cannot change if all input states of neighbor cells are also sleeping. Consequently, all follow-up configurations have only a finite number of cells that are not sleeping if one starts with such a configuration.

To keep the technicalities simple, we consider 2-dimensional cellular automata the cells of which are the unit squares in the Euclidean plane *GRID* and can be identified by their left lower corner. The neighborhood is defined by a vector $N = (N_1, \ldots, N_k) \in (\mathbb{Z} \times \mathbb{Z})^k$ where the neighbor cells of (i, j) are given by the translations $(i, j)+N_1, \ldots, (i, j)+N_k$. If one chooses the local states as colors, a cell with a local state can be represented by filling the area of the cell with the corresponding color. Accordingly, the underlying finite automaton is specified by a finite set of colors, say *COLOR*, and its transition $d\colon COLOR \times COLOR^k \to COLOR$. Without loss of generality, we assume $white \in COLOR$ and use it as sleeping state, i.e. $d(white, white^k) = white$. Under these assumptions, a configuration is a mapping $S\colon \mathbb{Z} \times \mathbb{Z} \to COLOR$ and the follow-up configuration S' of S is defined by $S'((i, j)) = d(S((i, j)), (S((i, j) + N_1)), \ldots, S((i, j) + N_k)))$.

If one starts with a configuration S_0 which has only a finite number of cells the colors of which are not *white*, then only these cells and those that have them as neighbors may change the colors. Therefore, the follow-up configuration has again only a finite number of cells with other colors than *white*. Consequently, the simultaneous change of colors of all cells can be computed. Moreover there is always a finite area of the Euclidean plane that contains all changing cells. In other words, a sequence of successive follow-up configurations can be depicted as a sequence of pictures by filling the cells with their colors.

Example 6. The following instance of a cellular automaton may illustrate the concept. It is called *SIER*, has two colors, $COLOR = \{white, black\}$, and the neighborhood vector is $N = ((-1, 0), (0, 1))$ meaning that each cell has the cell to its left and the next upper cell as neighbors. The transition of *SIER* changes *white* into *black* if exactly one neighbor is *black*, i.e. $d\colon COLOR \times COLOR^2 \to COLOR$ with $d(white, (black, white)) = d(white, (white, black)) = black$ and $d(c, (c_1, c_2)) = c$ otherwise. Let S_0 be the start configuration with $S_0((0, 0)) = black$ and $S_0((i, j)) = white$ otherwise. Then one gets the configuration S_{30} in Fig. 6 after 30 transitions. The drawing illustrates that *SIER* iterates the Sierpinski gadget (see, e.g., [28]) if one starts with a single black cell.

Cellular automata can be transformed into graph transformation units. Let *CA* be a cellular automaton with the neighborhood vector $N = (N_1, \ldots, N_k) \in (Z \times \mathbb{Z})^k$, the set of colors *COLOR* and the transition function

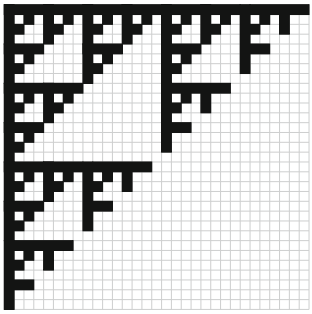
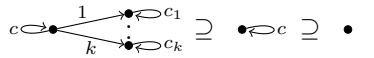


Fig. 6. A pictorial representation of the configuration S_{30} of *SIER*

$d\colon COLOR \times COLOR^k \to COLOR$. Then a configuration $S\colon \mathbb{Z} \times \mathbb{Z} \to COLOR$ can be represented by a graph $gr(N, S)$ with the cells as nodes, with k edges from each cell to its neighbors labeled with $1, \ldots, k$ in the order of the neighborhood, and a loop at each cell labeled with the color of the cell. The set of all these graphs is denoted by $\mathcal{G}(CA)$. If the color of a cell (i, j) changes, i.e. $d(S((i, j)), (S((i, j) + N_1), \ldots, S((i, j) + N_k))) \neq S(i, j)$, then the following rule with positive context  can be applied to the node (i, j) in $gr(N, S)$ provided that $c = S(i, j)$ and $c_p = S((i, j) + N_p)$ for $p = 1, \ldots, k$. Here the rule consists of the two inclusions to the left. The inclusion to the right is the positive context which serves as a control condition: The rule is applicable if the left-hand size is matched and the matching can be extended to the context. The set of all those rules is denoted by $\mathcal{R}(CA)$. A rule application removes a loop so that two rule applications are independent if and only if their matches do not overlap. Consequently, all applicable rules can be applied in parallel yielding $gr(N, S')$ where S' is the follow-up configuration of S. In other words, $gr(N, S) \Longrightarrow gr(N, S')$ with maximum parallelism is a direct derivation in the graph transformation unit $gtu(CA) = (\mathcal{G}(CA), \mathcal{R}(CA), \mathit{maxpar}, \mathcal{G}(CA))$.

Conversely, a derivation step $gr(N, S) \Longrightarrow H$ in $gtu(CA)$ changes a c-loop into a $d(c, (c_1, \ldots, c_k))$-loop at the node (i, j) if and only if, for $l = 1, \ldots, k$, the neighbor $(i, j) + N_l$ has a c_l-loop. All other c-loops are kept. This means that $H = gr(N, S')$. Summarizing, each cellular automaton can be transformed into a graph transformation unit such that the following correctness result holds.

Theorem 5. Let *CA* be a cellular automaton with neighborhood vector N and let *gtu(CA)* be the corresponding graph transformation unit. Then there is a transition from S to S' in *CA* if and only if $gr(N, S)) \Longrightarrow gr(N, S')$ in *gtu(CA)*.

Therefore, cellular automata behave exactly as their corresponding graph transformation units up to the representation of configurations as graphs. We have considered cellular automata over the 2-dimensional space $\mathbb{Z} \times \mathbb{Z}$. It is not difficult to see that all our constructions also work for the d-dimensional space

$\mathbb{Z}^d$ in a similar way. One may even replace the quadratic cells by triangular or hexagonal cells or use completely other networks.

7.2 Related Work

Like cellular automata, other approaches to parallel processing have been transformed into parallel graph transformation like Petri nets (cf. [3,29,30]), production networks (cf. [31]), artificial-ant colonies and particle swarms (cf. [32]). Moreover, parallel graph transformation provides a semantic domain for the graph-transformational modeling of various kinds of parallelism like for autonomous units (cf. [33]) and graph-transformational swarm computing (cf. [32]). Besides these examples that are closely related to the kind of parallel graph transformation considered in this paper, one encounters many further subjects in the literature where parallel rule application on and parallel evaluation of graph-like structures play an important rule like interaction nets, multi-agent systems, parallel and reversible circuits, various kinds of diagrams and networks. It may be worthwhile to look into the diverse research topics from a graph-transformational point of view.

8 Conclusion

In this paper, we have recalled the approach to parallel graph transformation that was introduced in [1], and have discussed some of its perspectives including the parallel generation of graph languages, the parallelization of graph algorithms, the infinite parallel graph transformation, and parallel graph transformation as a framework for the modeling of parallel processes. Further research on these topics can shed more light on their significance.

The theory of graph languages and those obtained by parallel generation in particular is not at all far developed. It may be worthwhile to study decidability and closure properties and to compare the various classes.

Given a specification of a graph algorithm by sequential graph transformation, one can always analyze the independence of rule applications to get a parallelized solution. The use of proper control conditions may lead to further improvement. Alternatively, graph algorithms may be modeled directly by means of parallel graph transformation. So far, not much work is done in this direction, but it may help to prove correctness and to analyze the complexity in a systematic way.

One can handle infinite graphs by the application of parallel rule with infinitely many finite component rules. There is a good chance that this machinery can contribute to infinite graph theory.

Parallel graph transformation has proven to provide a framework for the modeling of parallel processes and a domain into which other approaches to parallel-process modeling can be transformed. Therefore, it may be desirable to develop parallel graph transformation further into a visual modeling languages with suitable tool support.

Acknowledgment. We are grateful to the four reviewers for their helpful comments that lead to various improvements.

References

1. Ehrig, H., Kreowski, H.-J.: Parallelism of manipulations in multidimensional information structures. In: Mazurkiewicz, A. (ed.) MFCS 1976. LNCS, vol. 45, pp. 284–293. Springer, Heidelberg (1976). https://doi.org/10.1007/3-540-07854-1_188
2. Corradini, A., Ehrig, H., Heckel, R., Löwe, M., Montanari, U., Rossi, F.: Algebraic approaches to graph transformation part I: basic concepts and double pushout approach. In: Rozenberg [34], pp. 163–245
3. Baldan, P., Corradini, A., Ehrig, H., Löwe, M., Montanari, U., Rossi, F.: Concurrent semantics of algebraic graph transformations. In: Ehrig et al. [5], pp. 107–185
4. Kreowski, H.-J.: Manipulationen von Graphmanipulationen. Ph.D. thesis, Technische Universität Berlin (1978). Fachbereich Informatik
5. Ehrig, H., Kreowski, H.-J., Montanari, U., Rozenberg, G. (eds.): Handbook of graph grammars and computing by graph transformation, concurrency, parallelism, and distribution, vol. 3. World Scientific, Singapore (1999)
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of algebraic graph transformation. monographs in theoretical computer science. An EATCS Series. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
7. Ehrig, H., Ermel, C., Golas, U., Hermann, F.: Graph and model transformation: general framework and applications. monographs in theoretical computer science. An EATCS Series. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47980-3
8. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci. **109**, 181–224 (1993)
9. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer, Heidelberg (2006). https://doi.org/10.1007/11841883_4
10. Habel, A., Kreowski, H.-J.: Some structural aspects of hypergraph languages generated by hyperedge replacement. In: Brandenburg, F.J., Vidal-Naquet, G., Wirsing, M. (eds.) STACS 1987. LNCS, vol. 247, pp. 207–219. Springer, Heidelberg (1987). https://doi.org/10.1007/BFb0039608
11. Habel, A.: Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Berlin (1992)
12. Drewes, F., Habel, A., Kreowski, H.-J.: Hyperedge replacement graph grammars. In: Rozenberg [34], pp. 95–162
13. Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 125–213. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59126-6_3
14. Kreowski, H.-J., Klempien-Hinrichs, R., Kuske, S.: Some essentials of graph transformation. In: Esik, Z., Martin-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications. Studies in Computational Intelligence, vol. 25, pp. 229–254. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-540-33461-3_9
15. Rozenberg, G., Salomaa, A.: The Mathematical Theory of *L* Systems. Pure and Applied Mathematics: A Series of Monographs and Textbooks, vol. 90. Academic Press, Orlando (1980)

16. Kreowski, H.-J., Kuske, S., Lye, A.: Fusion grammars: a novel approach to the generation of graph languages. In: de Lara, J., Plump, D. (eds.) ICGT 2017. LNCS, vol. 10373, pp. 90–105. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61470-0_6
17. Floyd, R.W.: Algorithm 97 (shortest path). Commun. ACM **5**(6), 345 (1962)
18. Warshall, S.: A theorem on Boolean matrices. J. ACM **9**(1), 11–12 (1962)
19. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)
20. Mahr, B.: Algebraic complexity of path problems. RAIRO Theor. Inf. Appl. **16**(3), 263–292 (1982)
21. Litovski, I., Métivier, Y., Sopena, É.: Graph relabelling systems and distributed algorithms. In: Ehrig et al. [5], pp. 1–56
22. Kreowski, H.-J., Kuske, S.: Graph multiset transformation - a new framework for massively parallel computation inspired by DNA computing. Nat. Comput. **10**(2), 961–986 (2011). https://doi.org/10.1007/s11047-010-9245-6
23. Diestel, R. (ed.): Directions in Infinite Graph Theory and Combinatorics. Topics in Discrete Mathematics, vol. 3. Elsevier, North Holland (1992)
24. Codd, E.F.: Cellular Automata. Academic Press, New York (1968)
25. Kari, J.: Theory of cellular automata: a survey. Theoret. Comput. Sci. **334**, 3–33 (2005)
26. von Neumann, J.: The General and Logical Theory of Automata, pp. 1–41. Wiley, Pasadena (1951)
27. Wolfram, S.: A New Kind of Science. Wolfram Media Inc., Champaign (2002)
28. Peitgen, H.-O., Jürgens, H., Saupe, D.: Chaos and Fractals: New Frontiers of Science. Springer, New York (1992). https://doi.org/10.1007/978-1-4757-4740-9
29. Kreowski, H.-J.: A comparison between Petri nets and graph grammars. In: Noltemeier, H. (ed.) WG 1980. LNCS, vol. 100, pp. 306–317. Springer, Heidelberg (1981). https://doi.org/10.1007/3-540-10291-4_22
30. Corradini, A., Montanari, U., Rossi, F.: Graph processes. Fundam. Inform. **26**(3/4), 241–265 (1996)
31. Dashkovskiy, S., Kreowski, H.-J., Kuske, S., Mironchenko, A., Naujuk, L., von Totth, C.: Production networks as communities of autonomous units and their stability. Int. Electron. J. Pure Appl. Math. **2**, 17–42 (2010)
32. Abdenebaoui, L., Kreowski, H.-J., Kuske, S.: Graph-transformational swarms. In: Bensch, S., Drewes, F., Freund, R., Otto, F., (eds.) Proceedings of the Fifth Workshop on Non-Classical Models for Automata and Applications (NCMA 2013), pp. 35–50. Österreichische Computer Gesellschaft (2013)
33. Hölscher, K., Kreowski, H.-J., Kuske, S.: Autonomous units to model interacting sequential and parallel processes. Fundam. Inform. **92**, 233–257 (2009)
34. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation. Foundations, vol. 1. World Scientific, Singapore (1997)