Theoretical Computer Science 429 (2012) 193–201

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Polynomial graph transformability☆

Hans-Jörg Kreowski, Sabine Kuske *

University of Bremen, Department of Computer Science, P.O. Box 33 04 40, 28334 Bremen, Germany

ARTICLE INFO

Keywords:
Graph transformation
Polynomial graph transformation unit
Satisfiability problem
NP-completeness
Reduction

ABSTRACT

In this paper, we introduce and study polynomial graph transformability as a graph-transformational counterpart of the satisfiability problem of the propositional calculus.

1. Introduction

Graph transformation is a research area with contributions to formal language theory, visual modeling, model transformation, the theory of concurrency and others as well as with a spectrum of potential applications. The achievements of the first three decades are documented in the three volumes of the Handbook of Graph Grammar and Computing by Graph Transformation edited by Rozenberg et al. [1–3]. The more recent developments of the area are documented in the proceedings of the International Conferences on Graph Transformation and the International Symposium on Applications of Graph Transformations with Industrial Relevance [4–9].

While the research in graph transformation is often related to the areas mentioned above, there are only occasional contributions to complexity theory. In this paper, we undertake an attempt in this direction. We introduce and study polynomial graph transformability (*PGT*) as a graph-transformational counterpart of the satisfiability problem of the propositional calculus. It is based on the notion of a graph transformation unit which comprises a set of rules, specifications of initial and terminal graphs, and a control condition that regulates the application of rules (cf., e.g., [10]). As an instance of *PGT*, a graph transformation unit is accompanied by a polynomial and some initial graph. Then the question is whether there is a derivation from the given initial graph into some terminal graph permitted by the control condition and composed of a polynomial number of steps.

Choosing the components of graph transformation units accordingly, *PGT* turns out to be NP-complete. Moreover, we claim that *PGT* provides a suitable framework for the systematic modeling of decision problems on graphs in the complexity class NP and demonstrate this by two typical examples. If a graph problem is modeled by means of *PGT*, then one can execute it on graph transformation engines. Such a tool can be seen as a PGT solver in analogy to a SAT solver with respect to the satisfiability problem provided that the system has a non-deterministic mode of execution so that successful derivations may be found by repeated trials. We discuss this verification principle and make first experiments by means of the graph transformation engine GrGen.NET [11].

The paper is organized as follows. In Section 2, we recall the concepts of graph transformation and graph transformation units. Section 3 introduces the problem of polynomial graph transformability (*PGT*) and shows that it is in NP. The NP-completeness of *PGT* is shown in Section 4. Moreover, the subgraph isomorphism problem and the shapes partition problem are formulated as instances of the *PGT* in Section 5. Section 6 presents how polynomial graph transformation can be used for verification purposes by carrying over the ideas concerning verification based on SAT solvers. The paper ends with the conclusion.

☆ The authors would like to acknowledge that their research is partially supported by the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).

* Corresponding author.
E-mail addresses: kreo@informatik.uni-bremen.de (H.-J. Kreowski), kuske@informatik.uni-bremen.de (S. Kuske).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.12.039

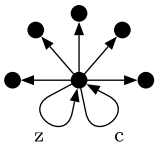


Fig. 1. A sample graph.

Fig. 2. A sample rule.

2. Preliminaries

In this section, the basic notions and notations of graph transformation are recalled as far as needed in the following.

The key concept of graph transformation is the application of a rule to a graph yielding a directly derived graph. In the literature, one encounters quite a variety of notions of graphs and rule applications. We employ a frequently used variant: multiple edge-labeled directed graphs and rule applications in the style of the double-pushout approach (cf., e.g., [12]).

Let Σ be a set of labels. A *graph* over Σ is a system $G = (V, E, s, t, l)$ where V is a finite set of *nodes*, E is a finite set of *edges*, $s, t: E \to V$ are mappings assigning a *source* $s(e)$ and a *target* $t(e)$ to every edge in E, and $l: E \to \Sigma$ is a mapping assigning a *label* to every edge in E. An edge e with $s(e) = t(e)$ is called a *loop*. The sum of the number of nodes and the number of edges is the *size* of G, denoted by *size*(G). 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 $\mathcal{G}_\Sigma$.

The notion is flexible enough to cover other types of graphs. *Simple graphs* form a subclass in which no graph has parallel edges. More formally, a graph $G = (V, E, s, t, l)$ is simple, if for all $e, e' \in E$ the fact $e \neq e'$ implies $s(e) \neq s(e')$ or $t(e) \neq t(e')$ or $l(e) \neq l(e')$. We reserve a specific label $*$ which is omitted in drawings of graphs. In this way, graphs where all edges are labeled with $*$ may be seen as *unlabeled graphs*.*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. The class of simple unlabeled loop-free graphs is denoted by $\mathcal{G}$, its subclass of undirected graphs by $\mathcal{G}_{undir}$.

An example of a directed graph consisting of six nodes, two labeled loops and five unlabeled edges is depicted in Fig. 1.

For graphs $G, H \in \mathcal{G}_\Sigma$, G is a *subgraph* of H, 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 each $e \in E_G$. A *graph morphism* $g: G \to H$ is a pair of mappings $g_V: V_G \to V_H$ and $g_E: E_G \to E_H$ 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 $e \in E_G$. If the mappings g_V and g_E are bijective, then G and H are called *isomorphic*, denoted by $G \cong H$. For a graph morphism $g: G \to H$, the image $g(G) \subseteq H$ of G in H is called a *match* of G in H.

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. An example of a rule is given in Fig. 2. The left-hand side consists if a c-looped node v connected to a node w and a z-looped node x. The gluing graph consists of the nodes u and w together with the c-loop. The right-hand side contains the gluing graph and a new z-looped node from which a new edge points to x.

The application of $r = (L \supseteq K \subseteq R)$ to a graph $G = (V, E, s, t, l)$ yields a directly derived graph H and consists of the following three steps.

1. A match $g(L)$ of L in G is chosen subject to the following conditions.
   - *dangling condition*: $v \in g_V(V_L)$ with $s_G(e) = v$ or $t_G(e) = v$ for some $e \in E_G - g_E(E_L)$ implies $v \in g_V(V_K)$.
   - *identification condition*: $g_V(v) = g_V(v')$ for $v, v' \in V_L$ implies $v = v'$ or $v, v' \in V_K$ as well as $g_E(e) = g_E(e')$ for $e, e' \in E_L$ implies $e = e'$ or $e, e' \in E_K$.
2. Now the nodes of $g_V(V_L - V_K)$ and the edges of $g_E(E_L - E_K)$ are removed yielding the *intermediate graph* $Z \subseteq G$.
3. Let $d: K \to Z$ be the restriction of g to K and Z. Then H is constructed as the disjoint union of Z and $R - K$ where all edges $e \in E_Z + (E_R - E_K)$ keep their labels and their sources and targets except for $s_R(e) = v \in V_K$ or $t_R(e) = v \in V_K$ which is replaced by $d_V(v)$.

The rule of Fig. 2 can be applied to the graph in Fig. 1 by identifying the c-looped with the z-looped node. The construction of a direct derivation where the left-most node of the left-hand side of the rule is mapped to the right-most node of the graph is illustrated in Fig. 3.

After four more rule applications one gets the graph depicted in Fig. 4. To this graph no further rule application is possible.

The application of a rule r to a graph G is denoted by $G \underset{r}{\Longrightarrow} H$ and called a *direct derivation*. The sequential composition of direct derivations

$$d = G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} G_n \quad (n \in \mathbb{N})$$

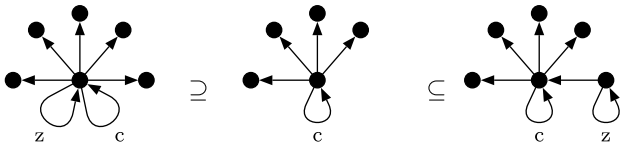


Fig. 3. An application of the above rule to the above graph.

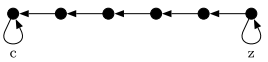


Fig. 4. The result of applying the above rule five times to the above graph.

is called a *derivation* from G_0 to G_n. The string $r_1 \cdots r_n$ is the *application sequence* of the derivation d. For a set P of rules, $G \overset{n}{\underset{P}{\Longrightarrow}} G'$ expresses that there exists a sequence $r_1, \ldots, r_n$ of rules in P and graphs $G_0, \ldots, G_n$ such that $G_0 = G, G_n = G'$ and $G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} G_n$. The subscript P may be omitted if it is clear from the context. The superscript n can be replaced by $*$ if the length of the derivation does not matter.

A *graph class expression* may be any syntactic entity X that specifies a class of graphs $SEM(X) \subseteq \mathcal{G}_\Sigma$. A typical example is a forbidden structure. Let F be a graph; then $SEM(forbidden(F))$ consists of all graphs G such that there is no subgraph in G that is isomorphic to F. Furthermore, we use the expressions $\mathcal{G}_{X+\overline{X}}$, *0-labeled*$(\mathcal{G})$ + *1-labeled*$(\mathcal{G})$, *0-looped*$(\mathcal{G})$ and $\mathbb{N} \cdot loops(\mathbb{N})$. The first expression specifies all graphs over $X + \overline{X}$ where X is some set, $\overline{X} = \{\overline{x} \mid x \in X\}$ is a disjoint copy of X and $X + \overline{X}$ is the disjoint union of X and $\overline{X}$. In Section 4, X is a set of Boolean variables, $\overline{X}$ contains their negations, and $X + \overline{X}$ is the set of all literals. The expression *a-labeled*$(\mathcal{G})$ specifies all graphs *a-labeled*(G) for $G \in \mathcal{G}$ where all edges of G are relabeled by a and all nodes get an a-loop. The expression *0-labeled*$(\mathcal{G})$ + *1-labeled*$(\mathcal{G})$ describes the disjoint union of the 0-labeled and the 1-labeled graphs. The third expression specifies all graphs *0-looped*(G) for $G \in \mathcal{G}$ where each node of G gets an extra 0-loop. Finally, $\mathbb{N} \cdot loops(\mathbb{N})$ describes all graphs $m \cdot loops(n)$ for $m, n \in \mathbb{N}$ consisting of m disjoint copies of $loops(n)$ which is a node with n 0-loops.

A *control condition* may be any syntactic entity that restricts the non-determinism of the derivation process. A typical example is a regular expression over a set of rules. Let C be a regular expression specifying the language $L(C)$. Then a derivation with application sequence s is *permitted* by C if $s \in L(C)$. A derivation $G \overset{*}{\underset{P}{\Longrightarrow}} H$ permitted by C is denoted by $G \overset{*}{\underset{P,C}{\Longrightarrow}} H$. In the following, we consider no other control conditions.

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 finite set of rules, and C is a control condition.

The operational semantics of gtu is given by the set of all *successful derivations* $DER(gtu)$ that derive initial graphs into terminal graphs and are permitted by C.

In the following, we assume that all considered graph class expressions define classes of graphs with polynomial membership problem and that all control conditions can be checked in polynomial time which is the case for all examples above.

3. The problem of polynomial graph transformability

Putting together the ingredients recalled in the preceding section, the problem of polynomial graph transformability (*PGT*) can be introduced as a member of the famous complexity class NP of non-deterministic polynomial decision problems. Each instance of *PGT* consists of a graph transformation unit gtu, a polynomial p, and an initial graph G_0. The question is whether there exists a successful derivation of G_0 in gtu with a length smaller than or equal to $p(size(G_0))$.

POLYNOMIAL GRAPH TRANSFORMABILITY (*PGT*)

instance: (gtu, p, G_0) where $gtu = (I, P, C, T)$ is a graph transformation unit, p is a polynomial and $G_0 \in SEM(I)$

question: Is there a derivation $G_0 \overset{k}{\underset{P,C}{\Longrightarrow}} G_k \in DER(gtu)$ with $k \leq p(size(G_0))$?

Theorem 1. *PGT* $\in$ NP.

Proof. By construction, $SEM(I)$ has a polynomial membership problem so that $G_0 \in SEM(I)$ is established in polynomial time. Let us assume that a derivation $G_0 \overset{i}{\underset{P}{\Longrightarrow}} G_i$ with $i \leq p(size(G_0))$ is also constructed in polynomial time. Then there is a polynomial number of matches in G_i and each match is found in polynomial time. Both numbers are polynomial in $size(G_i)$. because the number of rules is finite. Given a match, the check, whether the dangling and the identification condition hold, and the construction of the directly derived graph are linear in the size of G_i. But as the size of a graph changes only by

a constant if a rule of P is applied and because of $i \leq p(size(G_0))$, the two numbers are also polynomial in $size(G_0)$. In other words, the derivation $G_0 \stackrel{i}{\Longrightarrow} G_i$ can be prolonged to $G_0 \stackrel{k}{\underset{P}{\Longrightarrow}} G_k$ in polynomial time with respect to $size(G_0)$ as long as $k \leq p(size(G_0))$. Moreover, $SEM(T)$ has a polynomial membership problem. And whether the derivation fulfills the control condition C, can be checked in time linear to the length of the derivation which is bounded by the polynomial p. □

Obviously, the theorem remains true for all subproblems of *PGT*. This applies, in particular, to *sizeGT* which denotes the polynomial graph transformability problem with the polynomial fixed as the identity. Therefore, the lengths of derivations to be taken into account are bounded by the size of the respective initial graph. A further restriction is *#nodesGT* where the length bound is the number of nodes of the initial graph.

4. Reducing SAT3 to PGT

In this section, we reduce the satisfiability problem *SAT3* of propositional formulas in conjunctive normal form with three literals per clause to *PGT*. As *SAT3* is known to be NP-complete, *PGT* inherits this property.

4.1. The satisfiability problem SAT3

An instance of *SAT3* consists of a finite set X of Boolean variables and a propositional formula being a conjunction of clauses where each clause is a disjunction of three literals. A literal is a Boolean variable $x \in X$ or its negation $\bar{x} \in \overline{X}$. A formula is satisfied if a truth assignment $a: X \to \{T, F\}$ can be found such that each clause contains a Boolean variable x with $a(x) = T$ or a negation $\bar{y}$ with $a(y) = F$.

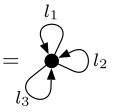
SATISFIABILITY (*SAT3*)

instance: (X, f) where X is a finite set of Boolean variables and $f = \bigwedge_{i\in[m]}(l_{i1} \vee l_{i2} \vee l_{i3})$ with $l_{ij} \in X + \overline{X}$, for all $i \in [m]$ and $j \in [3]$.[1]

question: Is there a truth assignment $a: X \to \{T, F\}$ such that, for each $i \in [m]$, a $j(i)$ exists with $l_{ij(i)} = x$ and $a(x) = T$ or $l_{ij(i)} = \bar{y}$ and $a(y) = F$?

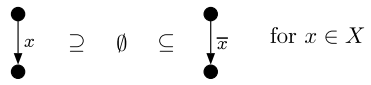
4.2. Reduction of SAT3 to PGT

To reduce *SAT3* to *PGT*, we construct a mapping r that maps a *SAT3*-instance (X, f) into a *PGT*-instance $r(X, f) = (sat3(X), p(X, f), G(X, f))$. The initial graph $G(X, f)$ consists of disjoint subgraphs of the form $\bullet \xrightarrow{x} \bullet$ for each $x \in X$ and disjoint subgraphs of the form

$loops(c) =$ 

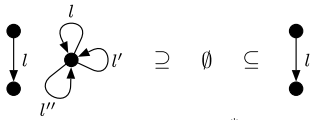
for each clause $c = l_1 \vee l_2 \vee l_3$ in f. The graph transformation unit $sat3(X)$ is defined as follows:

sat3(X)
initial: $\mathcal{G}_{X+\overline{X}}$
rules:
$negate(x)$



for $x \in X$

$evaluate(\{l, l', l''\})$



for $l, l', l'' \in X + \overline{X}$

control: $negate^*$; $evaluate^*$

terminal: $forbidden($  $\mid l \in X + \overline{X})$

[1] $[m]$ abbreviates the set $\{1, \ldots, m\}$.

It provides two types of rules. The negation rule allows to replace a disjoint subgraph $\bullet \xrightarrow{x} \bullet$ by $\bullet \xrightarrow{\overline{x}} \bullet$. The evaluation rule allows to remove a disjoint subgraph of the form *loops*(c) if a disjoint subgraph $\bullet \xrightarrow{l_j} \bullet$ for some $j \in [3]$ is around. The control condition requires that an arbitrary number of negation rule applications is followed by an arbitrary number of evaluation rule applications. The initial graphs are arbitrary graphs labeled in $X + \overline{X}$, the terminal graphs have no loops.

As the gluing graph of each rule is empty, the rules can only be applied to disjoint subgraphs of the forms of the left-hand sides. The application of an evaluation rule removes a node and the application of a negation rule concerns two nodes that cannot occur in matchings afterward. Therefore, no derivation can be longer than the number of nodes. In other words, no derivation is excluded if the polynomial $p(X, f)$ is chosen as the identity.

The construction of *sat3*(X) is polynomial in the number $\#X$ of Boolean variables because one must build $\#X$ negation rules and $(2 \cdot \#X)^3$ evaluation rules. Moreover, $2 \cdot \#X$ forbidden structures must be built and the graph $G(X, f)$ is constructed proportionally to $\#X$ and the number of clauses. The choice of the control condition and the polynomial $p(X, f)$ are constant actions. Altogether, the construction of $r(X, f)$ is polynomial in the number of Boolean variables and clauses.

Moreover, one can show that the construction is correct with respect to *SAT3* and *PGT*, i.e., for all *SAT3*-instances (X, f):

$$SAT3(X, f) = PGT(r(X, f)).$$

Let $a: X \to \{T, F\}$ be a truth assignment. Then one can apply the rules *negate*(x) for all $x \in X$ with $a(x) = F$ starting with the initial graph $G(X, f)$. This derives a graph with a disjoint subgraph $\bullet \xrightarrow{x} \bullet$ if $a(x) = T$ and a disjoint subgraph $\bullet \xrightarrow{\overline{y}} \bullet$ if $a(y) = F$. If f is satisfied through a, then each clause $c = l_1 \vee l_2 \vee l_3$ of f contains a literal x with $a(x) = T$ or a literal $\overline{y}$ with $a(y) = F$. This allows to apply the rule *evaluate*$(\{l_1, l_2, l_3\})$ removing the node with three loops corresponding to the clause c. Therefore, all nodes with loops can be removed ending up with a terminal graph. This means that $PGT(r(X, f))$ holds if $SAT3(X, f)$ holds. Conversely, if there is a successful derivation of $G(X, f)$ into a graph without loops, then f turns out to be satisfiable. The derivation decomposes into two sections $G(X, f) \stackrel{*}{\Longrightarrow} G$ and $G \stackrel{*}{\Longrightarrow} H$ where only negation rules are applied in the initial section and only evaluation rules in the terminal section. The intermediate graph induces a truth assignment $a: X \to \{T, F\}$ by $a(x) = T$, if G contains the subgraph $\bullet \xrightarrow{x} \bullet$ and $a(y) = F$, if G contains $\bullet \xrightarrow{\overline{y}} \bullet$. The graphs $G(X, f)$ and G contain a disjoint subgraph *loops*(c) for each clause $c = l \vee l' \vee l''$ of f. The only way to derive the loop-free H from G is by applying the rule *evaluate*$(\{l, l', l''\})$ eventually. But this is only possible if one of the three literals holds with respect to a. Therefore, all clauses of f hold.

All the considerations together prove the following theorem if the existence of a reduction of an NP-problem D to an NP-problem D' is denoted by $D \leq D'$.

Theorem 2. *SAT3* $\leq$ *PGT*.

Actually, the result can be formulated a bit stronger. As the construction of the graph transformation unit *sat3*(X) for any finite set X shows, no derivation in *sat3*(X) is ever longer than the number of nodes of its initial graph. Hence, the target problem of the reduction can be restricted to *#nodesGT*.

Corollary 1. *SAT3* $\leq$ *#nodesGT*.

It is a known fact that an NP-problem is NP-complete if an NP-complete problem can be reduced to it. Hence, the two theorems above and the corollary yield the NP-completeness of polynomial graph transformability.

Corollary 2. *#nodesGT*, *sizeGT and PGT are* NP*-complete.*

5. Modeling graph problems by means of PGT

Many well-known NP-problems are graph problems like the Hamilton-path problem, the clique problem and the coloring problem (cf., e.g., [13]). In most cases, they are formally described by some kind of second order monadic logic like "there exists a simple path visiting all nodes", " there exists a complete subgraph of a given size" and "there exists a mapping on the nodes into a set of a given size such that adjacent nodes get different values". To show that such problems belong to the class NP, the nondeterministic and polynomial algorithm that solves such a problem is often given in a rather informal way or is some kind of pseudo-code. Polynomial graph transformability provides an alternative modeling approach. A graph problem belongs to NP if it can be reduced to *PGT*. Several examples indicate that graph-transformational solutions of graph problems can be modeled in a quite natural way. This is illustrated by graph-transformational models of the subgraph isomorphism problem and the shapes partition problem where the latter generalizes the triangles partition problem (cf., e.g., [13]). Both problems are not yet considered in the context of graph transformation as far as we know.

To construct a reduction from a decision problem D to *PGT* is not only interesting as a way to prove that D belongs to NP. In the next section, we discuss tools that solve the polynomial graph transformability as SAT solvers solve the satisfiability problem. If a decision problem D is reduced to *PGT*, then D can be run on PGT solvers.

5.1. Reduction into PGT

A decision problem on graphs turns out to be in NP if it can be reduced to *PGT* (or *sizeGT* or *#nodesGT* likewise). This is made precise by the following observation.

Observation 1. *Let $INST \subseteq \mathcal{G}_\Sigma$ be a set of graphs with a polynomial membership problem and $D: INST \to \{T, F\}$ be a decision problem on $INST$. Let $red: INST \to INST(PGT)$ be a mapping from $INST$ to the set $INST(PGT)$ of instances of PGT that can be computed in polynomial time and is correct, i.e., for all $G \in INST$:*

$$D(G) = PGT(red(G)).$$

Then $D \in$ NP.

Proof. The correctness states that D can be computed by the sequential composition of the polynomial mapping *red* and an NP-solution of *PGT* yielding an NP-solution of D. □

Obviously, all the arguments remain true if *PGT* is replaced by *sizeGT* or *#nodesGT*.

5.2. Subgraph isomorphism problem

The subgraph isomorphism problem (*SUB*) checks for two graphs whether the first one is a subgraph of the second one (up to the naming of nodes and edges). We restrict the consideration to the class $\mathcal{G}$ of simple, unlabeled and loop-free graphs.

SUBGRAPH ISOMORPHISM (*SUB*)

instance: $(G, H) \in \mathcal{G} \times \mathcal{G}$

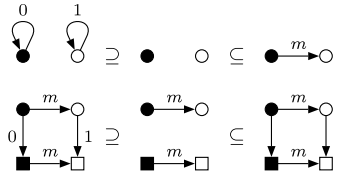
question: Is there an injective graph morphism $m: G \to H$?

We reduce *SUB* to *sizeGT* so that the reduction must associate a graph transformation unit and an initial graph to each *SUB*-instance (G, H) while the polynomial bound is fixed by the size of the initial graphs. As initial graph, we choose *0-labeled*(G) + *1-labeled*(H). As graph transformation unit, we use the following one in all cases:

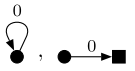
subgraph

initial: *0-labeled*$(\mathcal{G})$ + *1-labeled*$(\mathcal{G})$

rules:



terminal: *forbidden*(



)

As each rule application removes a 0-label (without creating new ones), the length of derivations is bounded by the number of edges of the 0-labeled graph, which is bounded by its size. Moreover, a corresponding 1-label is removed in each case so that rule applications establish one-to-one relations. Edges are only related if sources and targets are also related. If the derivation is successful, then all 0-labels are removed so that the relation defines an injective morphism $m: G \to H$.

5.3. Shapes partition problem

The shapes partition problem is parameterized by a finite set of non-empty shapes $SHAPES \subseteq \mathcal{G} - \{\emptyset\}$. It checks for each input graph whether there is a set of disjoint subgraphs such that all nodes are covered and every subgraph is isomorphic to one of the shapes.

SHAPES PARTITION

instance: $G \in \mathcal{G}$.

question: Is there a set $P \subseteq \mathcal{G}$ such that
(1) $S \subseteq G$ for each $S \in P$,
(2) $S \cap S' = \emptyset$ for each $S, S' \in P$ with $S \neq S'$,
(3) $V_G = \bigcup_{S \in P} V_S$ and
(4) for each $S \in P$, there exists an $S' \in SHAPES$ with $S' \cong S$.

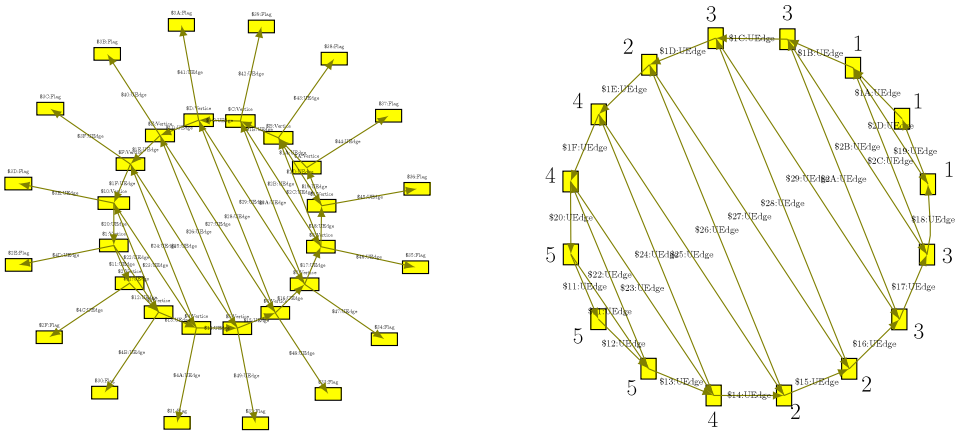


Fig. 5. A graph and its partition into 4 triangles and 1 quadrangle.

SHAPES PARTITION can be reduced to *#nodesGT* mapping an input graph G to the *#nodesGT*-instance *shapes-partition*(*SHAPES*) and 0-*looped*(G) where the graph transformation unit is defined as follows.

shapes-partition(*SHAPES*)
initial: *0-looped*(G)
rules: *0-looped*(S) $\supseteq S \subseteq S$ for $S \in$ *SHAPES*
terminal: *forbidden*()

Considering a derivation of the initial graph *0-looped*(G), a rule application identifies one of the given shapes as a subgraph of G. As the 0-loops can only be removed once and a terminal graph is loop-free, every node of G belongs to exactly one shape. Therefore, the reduction is linear in the number of nodes and correct.

6. Toward PGT solvers

The satisfiability problem plays two very important roles. On one hand, it is a distinguished member of the class of NP-complete problems in the center of many theoretical considerations. On the other hand, it is the basis of SAT solvers that get propositional formulas – mainly in conjunctive normal form – as inputs and check their satisfiability. In the positive case, a satisfying truth assignment is yielded in addition. Although SAT solvers run exponentially in general, they provide proper results in due time in many practical cases. In particular, they have turned out to be helpful in chip design and verification (cf., e.g., [14,15]).

We hope that graph transformation engines may play a similar role with respect to *PGT* as SAT solvers with respect to satisfiability. We demonstrate this idea by means of the graph transformation engine GrGen.NET [11]. The instances of *PGT* can be adapted in such a way that they can serve as inputs of the GrGen.NET system which tries to find a successful derivation. GrGen.NET can be run nondeterministically so that one can try the same input repeatedly and the chance grows with the number of attempts to get a successful derivation (provided that there is one at all). This allows one to verify properties that are equivalent to the existence or non-existence of successful derivations in *PGT*. In the following subsections, the principle is illustrated.

6.1. Finding shapes partitions

We start with a toy example. Assume that one would like to find some shapes partitions for certain graphs and certain shapes. Consider, for instance, the graph on the left-hand side of Fig. 5 which is a maximum outer-planar graph with 16 nodes where the additional flags at the nodes on the cycle represent 0-loops so that this graph corresponds to an initial graph of the graph transformation unit *shapes-partition*(*SHAPES*). For a triangle and a quadrangle as shapes, GrGen.NET yielded proper results. To achieve this, we made 30 experiments with 25 runs each. In each run, the system builds a derivation non-deterministically. 23 experiments yielded successful derivations. A partition consisting of quadrangles only was constructed three times, i.e., the remaining 20 successful derivations generated partitions consisting of 4 triangles and 1 quadrangle. One of these is depicted on the right of Fig. 5 where the nodes belonging to a shape have the same number.

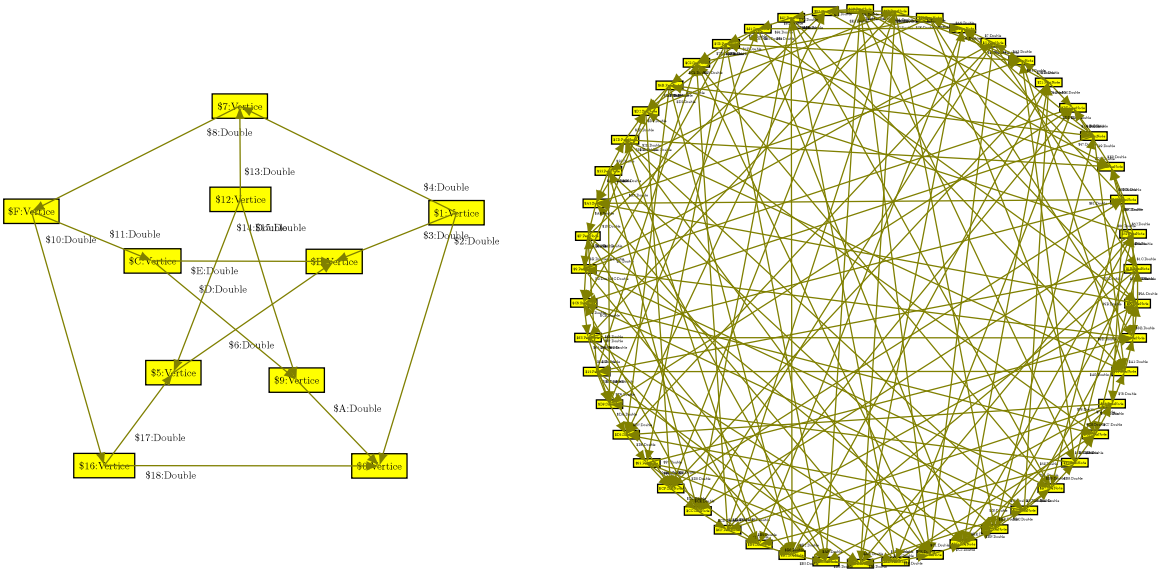
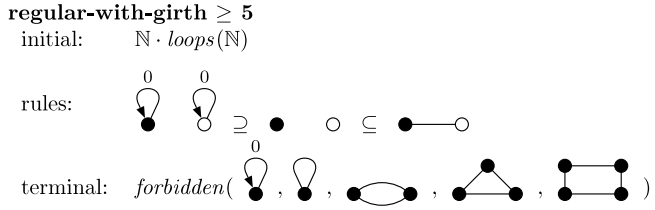


Fig. 6. Petersen graph (left) and Hoffman-Singleton graph (right).

6.2. Regular graphs with girth 5

Let $d \in \mathbb{N}$. Then A simple and loop-free undirected graph is d-regular if d is the degree of each node. This means that each node is incident with d (undirected) edges. The girth of a graph is the minimum length of a cycle. The diameter of a graph is the greatest distance between any pair of vertices where the distance is the length of the shortest path between the two nodes. It is known that a d-regular graph with girth 5 and diameter 2 has at least $d^2 + 1$ nodes. Therefore, one may pose the problem whether there is, for $d \in \mathbb{N}$, a d-regular graph with girth 5, diameter 2 and $d^2 + 1$ nodes. In [16], it is shown that the answer can only be positive if $d \in \{2, 3, 7, 57\}$. The pentagon solves the problem for $d = 2$, the Petersen graph for $d = 3$ and the Hoffman–Singleton graph for $d = 7$ (cf. Fig. 6). For $d = 57$, the problem is unsolved.

Executing *PGT* on GrGen.NET, we can confirm the results for $d = 3$ using the following graph transformation unit, the size, and the initial graphs $10 \cdot loops(3)$ as instance.



Starting with the initial graph $m \cdot loops(n)$ for $m, n \in \mathbb{N}$, each rule application consumes two 0-loops without generating new ones so that all lengths of derivations are bounded by the size of the initial graph. Moreover, each rule application increases the degree of the two processed nodes by 1 so that all nodes have degree n if no 0-loop is left. Finally, the resulting graphs of successful derivations have no cycles of length 1, 2, 3 or 4, so that they have a girth ≥ 5.

For $10 \cdot loops(3)$, we made 30 experiments with 10 runs each and 30 with 20 runs. A third of the experiments failed in the first case, and only one of 30 experiments was not successful with 20 runs. In all other cases, the Petersen graph on the left of Fig. 6 was found.

Trying the same for the initial graph $50 \cdot loops(7)$, all our experiments failed even with 600.000 runs. This is not very surprising because the graph transformation unit is highly non-deterministic so that the number of derivation grows exponentially and the successful part gets smaller and smaller.

Fortunately, a modified approach works confirming the case $d = 7$. First, we allow arbitrary graphs as initial ones adding 0-loops to each node so that the resulting graph becomes regular. Second, we use an additional control condition to cut down the non-determinism. In the actual case, we start with 5 copies of the Petersen graph and add 4 loops to each of the 50 nodes. And the control condition makes sure that new edges connect always outer nodes with inner nodes. For this setting, we made 100 experiments with 10.000 runs each. An experiment took about 10 seconds on an ordinary laptop. 12 experiments failed; all others found the Hoffman-Singleton graph. In the best case, it took 18 runs; in the worst case, success needed 9704 runs. The experienced difficulties to find the Hoffman-Singleton graph by means of GrGen.NET may sound somewhat discouraging to solve the open problem with respect to degree 57. We must admit that we did not try, yet. But it may not be hopeless if one develops a proper strategy.

7. Conclusion

In this paper, we have introduced polynomial graph transformability (*PGT*) as a decision problem that checks for a graph transformation unit, a polynomial, and an initial graph whether the latter can be derived successfully into a terminal graph in a polynomial number of steps. *PGT* is shown to be NP-complete. Moreover, it provides a systematic way to show that decision problems on graphs belong to the complexity class NP. And we have demonstrated that graph transformation engines like GrGen.NET may be used to prove the existence of certain graphs or certain successful derivations in a similar way as SAT solvers are employed to check the satisfiability of propositional formulas.

To shed more light on the significance of this tentative approach, further research should be done in the following respects at least:

1. Our first hope and claim is that polynomial graph transformability (*PGT*) provides a suitable formal framework to model NP-solutions for graph problems in a systematic and natural way. In Section 5, the subgraph isomorphism problem and the shapes partition problem exemplify the idea. It may be interesting to find further examples – maybe more complex or more surprising – that emphasize the appropriateness.
2. Our second hope and claim is that graph transformation engines may be used as PGT solvers and may play a similar role with respect to *PGT* as SAT solvers with respect to the satisfiability problem. The very first experiments of this kind look somewhat promising although we used the GrGen.NET system in a quite naive way. Therefore, better results may be obtained if GrGen.NET or another graph transformation tool is optimized for the task of PGT solving.

Acknowledgements

We are grateful to the anonymous reviewers for their helpful comments.

References

[1] G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations, World Scientific, Singapore, 1997.
[2] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Eds.), Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools, World Scientific, Singapore, 1999.
[3] H. Ehrig, H.-J. Kreowski, U. Montanari, G. Rozenberg (Eds.), Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution, World Scientific, Singapore, 1999.
[4] A. Corradini, H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Proc. First International Conference on Graph Transformation, ICGT 2002, in: Lecture Notes in Computer Science, vol. 2505, Springer, 2002.
[5] H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), Proc. Second International Conference on Graph Transformation, ICGT 2004, in: Lecture Notes in Computer Science, vol. 3256, Springer, 2004.
[6] A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg (Eds.), Third International Conference on Graph Transformation, ICGT 2006, in: Lecture Notes in Computer Science, vol. 4178, Springer, 2006.
[7] A. Schürr, M. Nagl, A. Zündorf (Eds.), Proc. Third International Symposium on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2007, in: Lecture Notes in Computer Science, vol. 5088, Springer, 2008.
[8] H. Ehrig, R. Heckel, G. Rozenberg, G. Taentzer (Eds.), Proc. 4th International Conference on Graph Transformation, ICGT 2008, in: Lecture Notes in Computer Science, vol. 5214, Springer, 2008.
[9] H. Ehrig, A. Rensink, G. Rozenberg, A. Schürr (Eds.), Proc. 5th International Conference on Graph Transformation, ICGT 2010, in: Lecture Notes in Computer Science, vol. 6372, Springer, 2010.
[10] H.-J. Kreowski, S. Kuske, G. Rozenberg, Graph transformation units—An overview, in: P. Degano, R.D. Nicola, J. Meseguer (Eds.), Concurrency, Graphs and Models, in: Lecture Notes in Computer Science, vol. 5065, 2008, pp. 57–75.
[11] R. Geiß, M. Kroll, GrGen.NET: a fast, expressive, and general purpose graph rewrite tool, in: [7], pp. 568–569.
[12] A. Corradini, H. Ehrig, R. Heckel, M. Löwe, U. Montanari, F. Rossi, Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach, in: [1], pp. 163–245.
[13] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.
[14] M. Ganai, A. Gupta, SAT-Based Scalable Formal Verification Solutions (Series on Integrated Circuits and Systems), Springer, 2007.
[15] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, in: Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.
[16] A. Hoffman, R. Singleton, On Moore graphs with diameters 2 and 3, IBM Journal of Research and Development 4 (1960) 497–504.