Electronic Communications of the EASST
Volume 61 (2013)

Selected Revised Papers from the 4th International Workshop on Graph Computation Models (GCM 2012)

A Graph Transformational View on Reductions in NP

Marcus Ermler, Sabine Kuske, Melanie Luderer and Caroline von Totth

17 pages

Guest Editors: Rachid Echahed, Annegret Habel, Mohamed Mosbah
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Graph Transformational View on Reductions in NP

Marcus Ermler, Sabine Kuske, Melanie Luderer and Caroline von Totth

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

Abstract: Many decision problems in the famous and challenging complexity class NP are graph problems and can be adequately specified by polynomial graph transformation units. In this paper, we propose to model the reductions in NP by means of a special type of polynomial graph transformation units, too. Moreover, we present some first ideas how the semantic requirements of reductions including their correctness can be proved in a systematic way.

Keywords: Graph transformation units, reductions

1 Introduction

Many famous NP-complete problems involve graphs. Examples of this kind are the problems of finding a clique, a Hamiltonian cycle, a vertex cover, an independent set, etc. Whereas the complexity class NP as well as the notion of reductions between NP problems are usually defined by means of polynomial Turing machines on the general level, explicit problems and reductions are described on some higher level for easier reading and understanding.

Since the algorithms solving decision problems or modeling reductions are often composed of graph transformation steps, polynomial graph transformation units [KKR08, KK12] serve as visual, rule-based and formal descriptions for these algorithms. Consequently, graph transformation units may be helpful not only for specifying and understanding decision problems and reductions but also for obtaining correctness proofs in a systematic way. Graph transformation units contain graph transformation rules for modeling the graph transformation steps in an elegant, formal, visual and intuitive way. Moreover, the control conditions of graph transformation units restrict the set of all derivations induced by the rules to those which solve the problem. Finally, the initial and terminal graph class expressions of graph transformation units allow to specify the input and output types of the algorithms.

In [KK11], it has already been shown that polynomial graph transformation units are a formal computational model for decision problems in NP. To underline the usefulness of this result, we model in this paper the problems of finding a clique, an independent set, a vertex cover and a Hamiltonian cycle as graph transformation units. Moreover, we extend [KK11] by considering also reductions in NP. A reduction from a graph transformation unit to a graph transformation unit transforms the initial graphs of the first unit to the initial graphs of the second unit. Polynomial graph transformation units with stepwise control serve as a computational model for such reductions in NP if they are deadlock-free and correct. The correctness can be split into forward and backward correctness. To stress this, we present reduction units from the clique problem

to the independent set problem and - more sophisticated - from the vertex cover problem to the Hamiltonian cycle problem.

Moreover, we make a first step towards a proof scheme for the correctness of reductions. We show that forward correctness is obtained by induction on the length of computations provided that there are certain auxiliary reductions compatible with the computation steps. We illustrate the principle for the presented examples.

The aim of this paper is to show that graph transformation units provide a uniform, systematic, and high-level framework for the specification of decision problems as well as of reductions between them which is more intuitive than but as formal as Turing machines. Moreover, the paper illustrates how the presented approach may serve as a foundation of a proof scheme for correctness of reductions.

The paper is organized as follows. In Section 2, polynomial graph transformation units with stepwise control are presented. Section 3 shows how graph transformation units can be used as a computational model for the decision problems in NP. Section 4 proposes graph transformation units for modeling reductions in NP. Section 5 deals with correctness of reductions. The paper ends with the conclusion.

2 Polynomial Graph Transformation Units with Stepwise Control

In this section, we briefly recall graph transformation units as far as they are needed in the following sections. We emphasize especially the use of stepwise control conditions in polynomial graph transformation units as they are essential for our approach to reductions in NP.

Graphs. Let A be a set of labels including a special label $*$. A *directed edge-labeled graph* over A is a system $G = (V,E,s,t,l)$ where V is a finite set of *nodes*, E is a finite set of *edges*, $s,t\colon E \to V$ are mappings assigning a *source* $s(e)$ and a *target* $t(e)$ to every edge in E, and $l\colon E \to A$ is a mapping assigning a label to every edge in E. 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 directed edge-labeled graphs over A is denoted by $\mathscr{G}$. An edge e with $s(e) = t(e)$ is called a *loop*. A loop with label a is called an *a-loop*. A node with an a-loop is called an *a-node*. An edge e with $l(e) = *$ is called *unlabeled* and the label $*$ is omitted in drawings. We call a graph $G = (V,E,s,t,l)$ *simple* if for all $e_1,e_2 \in E$, $e_1 \neq e_2$ implies $s(e_1) \neq s(e_2)$ or $t(e_1) \neq t(e_2)$. A pair (e,e') of edges in a simple graph is considered as an *undirected* edge if $s(e) = t(e')$ and $t(e) = s(e')$.

Let $G,H \in \mathscr{G}$. G is called 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 all $e \in E_G$. 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 $e \in E_G$. For a graph morphism $g\colon G \to H$, the image $g(G) \subseteq H$ of G in H is called a *match* of G in H. An *injective match* of G in H is a match where the underlying graph morphism $g\colon G \to H$ is injective.

Let $G = (V,E,s,t,l)$ and $G' = (V',E',s',t',l')$ be graphs in $\mathscr{G}$. Then the graph $G+G' = (V \uplus V', E \uplus E', s'', t'', l'')$[1] with $s''(e) = s(e)$, $t''(e) = t(e)$, $l''(e) = l(e)$ for all $e \in E$ and $s''(e') = s'(e')$,

[1] $\uplus$ denotes the disjoint union of sets.

$t''(e') = t'(e')$, $l''(e') = l'(e')$ for all $e' \in E'$ is called the *disjoint union* of G and G'.

Rules and their application. A *rule* $r = (L \supseteq K \subseteq R)$ consists of three graphs $L, K, R \in \mathscr{G}$ 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 $r = (L \supseteq K \subseteq R)$ to a graph $G = (V, E, s, t, l)$ yields a directly derived graph H and consists of the following three steps. (1) An injective match $g(L)$ of L in G is chosen subject to the dangling condition: $s_G(e) = v$ or $t_G(e) = v$ for some $e \in E_G - g_E(E_L)$ and $v \in g_V(V_L)$ implies $v \in g_V(V_K)$. (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 $d\colon K \to Z$ be the restriction of g to K and Z. Then H is constructed as the componentwise disjoint union for nodes and edges 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 application of a rule r to a graph G is denoted by $G \underset{r}{\Longrightarrow} H$, and called a *direct derivation*. The subscript r may be omitted if it is clear from the context. The sequential composition of direct derivations $d = G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} G_n$ ($n \in \mathbb{N}$) is called a *derivation* from G_0 to G_n. As usual, the derivation from G_0 to G_n can also be denoted by $G_0 \overset{n}{\underset{P}{\Longrightarrow}} G_n$ where $\{r_1, \ldots, r_n\} \subseteq P$, or just by $G_0 \overset{*}{\underset{P}{\Longrightarrow}} G_n$. The sequential composition of two derivations $G_0 \overset{*}{\underset{P}{\Longrightarrow}} G'$, $G' \overset{*}{\underset{P}{\Longrightarrow}} G''$ is a derivation $G_0 \overset{*}{\underset{P}{\Longrightarrow}} G''$. The subscript P may be omitted if it is clear from the context. The string $r_1 \cdots r_n$ is the *application sequence* of the derivation d. The notion of a direct derivation fits into the double-pushout approach (e.g. [CEH+97]).

A rule with a *negative application condition* consists of four graphs $N, L, K, R \in \mathscr{G}$ such that $N \supseteq L \supseteq K \subseteq R$. The application of such a rule to a graph G is defined as above with the additional condition that the graph morphism g cannot be extended to some morphism $g'\colon N \to G$ of which g is the restriction to L. By $\mathscr{R}$ we denote the class of rules consisting of rules $(N \supseteq L \supseteq K \subseteq R)$ with a negative application condition. Negative application conditions are studied in [HHT96].

In the following we consider only alphabets in which equality of labels can be checked in polynomial time. Given a finite set of rules and a graph G, the number of matches is bounded by a polynomial in the size of G because the sizes of left-hand sides and of the negative contexts of rules are bounded by a constant. Given a match, the check, whether the dangling condition holds, and the construction of the directly derived graph is linear in the size of G. Therefore, under the mentioned assumption polynomial time is needed to find a match and to construct a direct derivation, and there is a polynomial number of choices at most. Moreover, the difference of the size of the resulting graph and the host graph is bounded by a constant (cf. [KK12]).

Graph class expressions. A *graph class expression* may be any syntactic entity X that specifies a class of graphs $SEM(X) \subseteq \mathscr{G}$. A typical example is a forbidden structure. Let $\mathscr{F}$ be a set of graphs; then $SEM(forbidden(\mathscr{F}))$ consists of all graphs G such that for each $F \in \mathscr{F}$ there is no injective match of F in G. Furthermore, we use the expressions *all*, *simple&unlabeled&looped* and $gr(\mathbb{N})$. The first expression specifies $\mathscr{G}$. The second expression specifies all graphs that are simple, unlabeled, and have an unlabeled loop at each node. The third expression speci-

fies the set $SEM(gr(\mathbb{N})) = \{gr(k) \mid k \in \mathbb{N}\}$ where $gr(k)$ consists of a single node with k *succ*-loops. The expression *simple&unlabeled&looped* + $gr(\mathbb{N})$ denotes all graphs $G + G'$ such that $G \in SEM(simple\&unlabeled\&looped)$ and $G' \in SEM(gr(\mathbb{N}))$. A graph class expression X is polynomial if for each $G \in \mathscr{G}$ it can be checked in polynomial time whether $G \in SEM(X)$. All graph class expressions above are polynomial. In the following, the class of graph class expressions is denoted by $\mathscr{E}$ and we assume that $\mathscr{E}$ consists of polynomial graph class expressions, only.

Stepwise control conditions. A stepwise control condition directly guides the derivation process, i.e. it provides for each derivation step the next permitted rule application steps. More formally, a *stepwise control condition* $C = (S, J, F, choice)$ consists of a finite set of *control states* S, two subsets $J, F \subseteq S$ of *initial and final control states* resp. and a *choice function* $choice$ with $choice(G,s) \subseteq \mathscr{G} \times S$ for $G \in \mathscr{G}$ and $s \in S$. We denote the class of stepwise control conditions by $\mathscr{C}$.

Stepwise control conditions can be often defined w.r.t. *control conditions* where a control condition is any expression that specifies a binary relation on graphs. Examples of control conditions are the *basic control conditions* $r!$ and $try(r)$ where r is a rule. The expression $r!$ means to apply r as long as possible. The expression $try(r)$ means that if r is applicable to the current graph then apply r once. In the following, *Try&Alap* denotes the class in which each control condition is either a basic control condition or it is the sequential composition of basic control conditions, i.e., it has the form $c_1; \cdots ; c_n$ ($n \geq 1$) where for $i = 1, \ldots, n$ c_i is a basic control condition.

For each control condition $c \in Try\&Alap$ the corresponding stepwise control condition $stw(c)$ is equal to $(S^c, J^c, F^c, choice^c)$ where S^c is defined recursively as $S^c = \{b; lambda, lambda\}$ if $c = b$ and $S^c = \{c; lambda\} \cup S^d$ if $c = b; d$ for some basic control condition b and $d \in Try\&Alap$. (Hence, every state is either equal to $lambda$ or has the form $b; s'$ where s' is a state and b is a basic control condition.) Moreover, $J^c = \{c; lambda\}$ and $F^c = \{lambda\}$. For each $G \in \mathscr{G}$, each $s \in S^c$ and each rule r occurring in c the choice function is given by $choice^c(G, lambda) = \emptyset$ and

$$choice^c(G, r!; s) = \begin{cases} \{(G', r!; s) \mid G \underset{r}{\Longrightarrow} G'\} & \text{if } \exists G' \in \mathscr{G} : G \underset{r}{\Longrightarrow} G' \\ \{(G, s)\} & \text{otherwise} \end{cases}$$

$$choice^c(G, try(r); s) = \begin{cases} \{(G', s) \mid G \underset{r}{\Longrightarrow} G'\} & \text{if } \exists G' \in \mathscr{G} : G \underset{r}{\Longrightarrow} G' \\ \{(G, s)\} & \text{otherwise} \end{cases}$$

In examples, each stepwise control condition $stw(c)$ will be abbreviated by c.

A *configuration* of a stepwise control condition $C = (S, J, F, choice)$ is a pair (G, s) with $G \in \mathscr{G}$ and $s \in S$. $(G, s) \vdash (G', s')$ is a *computational step* if $(G', s') \in choice(G, s)$. A *computation* is a sequence of computational steps $(G_0, s_0) \vdash (G_1, s_1) \vdash \cdots \vdash (G_n, s_n)$ (also denoted by $(G_0, s_0) \vdash^n (G_n, s_n)$). Obviously, each computation induces an underlying derivation

$$G_0 \Longrightarrow G_1 \Longrightarrow \cdots \Longrightarrow G_n.$$

The *semantics* $SEM(C)$ is given by the set of derivations induced by all computations $(G_0, s_0) \vdash^n (G_n, s_n)$ with $s_0 \in J$ and $s_n \in F$.

Graph transformation units. A *graph transformation unit* is a system $gtu = (I, P, C, T)$, where $I, T \in \mathscr{E}$ are graph class expressions to specify the *initial* and the *terminal* graphs respectively, $P \subseteq \mathscr{R}$ is a finite set of rules, and $C \in \mathscr{C}$ is a stepwise control condition. Every graph transformation unit gtu specifies a binary relation $SEM(gtu) \subseteq SEM(I) \times SEM(T)$ that contains a pair (G,H) of graphs if and only if there is a derivation $G \overset{*}{\underset{P}{\Longrightarrow}} H \in SEM(C)$. For each $G \in SEM(I)$ $gtu(G)$ denotes the set $\{H \in \mathscr{G} \mid (G,H) \in SEM(gtu)\}$.

Let $gtu = (I, P, C, T)$ be a transformation unit with a stepwise control condition

$$C = (S, J, F, choice).$$

A configuration (G,s) is *initial* if $G \in SEM(I)$ and $s \in J$. It is *terminal* if $G \in SEM(T)$ and $s \in F$. A *permitted computation* is a sequence of computational steps $(G_0,s_0) \vdash (G_1,s_1) \vdash \cdots \vdash (G_n,s_n)$ if (G_0,s_0) is an initial configuration. The induced derivation of a permitted computation is called *permitted derivation*. Note that the 0-derivation $G \overset{0}{\Longrightarrow} G$ is always permitted if $G \in SEM(I)$. A permitted computation is *successful* if (G_n,s_n) is a terminal configuration. The induced derivation of a successful computation is called *successful derivation*.

Polynomial graph transformation units. A gtu is *polynomial* if the following holds: (1) there is a polynomial p such that for each initial graph $G \in SEM(I)$ and each permitted derivation $G \overset{n}{\Longrightarrow} G'$, $n \leq p(size(G))$, where $size(G)$ is the sum of the number of nodes and the number of edges of G. (2) the membership problems of $SEM(I)$ and $SEM(T)$ are polynomial. (3) to compute a next configuration via the choice function takes polynomial time. Please note that for all graph class expressions and all stepwise control conditions used in this paper, the second and the third condition are satisfied (each next configuration is picked up nondeterministically based on a fixed rule set).

3 Decision Problems of NP as Graph Transformation Units

In the following, it is recalled how polynomial graph transformation units can be used as a computation model for decision problems in the complexity class NP (cf. [KK11]).

3.1 Decision Problems of NP

A decision problem is a mapping $D\colon \Sigma^* \to BOOL$, where Σ is some finite alphabet. D is in the complexity class NP if there exists a nondeterministic Turing machine TM and a polynomial p such that for each input $w \in \Sigma^*$ the following holds. (1) there is a computation of TM starting in the initial state with input w and ending in an accepting state if and only if $D(w) = true$ and (2) no computation of TM starting with input w is longer than $p(|w|)$ (cf., e.g., [HMU07]).

Very often, inputs of decision problems are not strings but they are composed of different data types such as graphs and natural numbers. Describing these instances as words in Σ^* would be very hard to read for human beings. Hence, in the literature, they are usually defined directly. For the same reason, the algorithms solving the decision problems are generally not given as Turing machines but as some higher level algorithmic description, and — based on the Church-Turing thesis — it is assumed that they can be computed by some Turing machine.

3.2 Characterizing NP by Graph Transformation

Whenever inputs of decision problems are graphs, polynomial graph transformation units serve as an intuitive computational model for NP-problems. More explicitly, a polynomial graph transformation unit $gtu = (I,P,C,T)$ *solves* a *graph transformational decision problem* $D\colon SEM(I) \to BOOL$ in the following way. Whenever there is a successful derivation $G \overset{*}{\Longrightarrow} G'$ in gtu, the result of the decision problem applied to G is *true*. Otherwise, it is *false*. Let $\mathrm{NP}_{\mathrm{GT}}$ denote the class of all graph transformational decision problems solvable by some polynomial graph transformation unit. The following statement is shown in [KK11].

Observation 1 $\quad \mathrm{NP}_{\mathrm{GT}} = \mathrm{NP}$.

This means that for each decision problem $D\colon \Sigma^* \to BOOL$ there is a polynomial graph transformation unit gtu that solves $D_{\mathscr{G}}\colon \Sigma^*_{\mathscr{G}} \to BOOL$ with $D_{\mathscr{G}}(w_{\mathscr{G}}) = D(w)$ for each $w \in \Sigma^*$ where $w_{\mathscr{G}}$ denotes the string graph representing w and $\Sigma^*_{\mathscr{G}}$ denotes the set of all string graphs over Σ. Conversely, let $D\colon SEM(I) \to BOOL$ be a graph transformational decision problem solved by some polynomial graph transformation unit $gtu = (I,P,C,T)$. Then there is a polynomial nondeterministic Turing machine that solves $D_{Str}\colon \Sigma^* \to BOOL$ with $D_{Str}(G_{Str}) = D(G)$ for all $G \in SEM(I)$ where $G_{Str} \in \Sigma^*$ is an appropriate string representation of G.

To illustrate how decision problems can be modeled as graph transformation units, we present the decision problem *clique* of NP as a graph transformation unit.

3.3 Example: Cliques

The decision problem *clique* has as input an undirected unlabeled graph G and a natural number k. The result of $clique(G,k)$ returns *true* if G contains a clique of size k, i.e., a complete subgraph with k nodes. Otherwise it returns *false*. For technical simplicity we assume that k is less than or equal to the number of nodes in G.

The problem *clique* can be modeled by the transformation unit *CLIQUE* in Figure 1. Each initial graph of *CLIQUE* is the disjoint union of two graphs. The first is an undirected simple unlabeled graph G in which each node is equipped with a (directed) loop. The second one is the graph $gr(k)$ for some $k \in \mathbb{N}$ consisting of a single node with k *succ*-loops (i.e., k loops each labeled with *succ*). It is worth noting that the unary encoding of k is possible because *clique* is strongly NP-complete.

The rule *select* is applied at first as long as possible selecting in each application a node of G while removing a *succ*-loop. Afterwards the rule $test(CLIQUE)$ is applied once if possible. Its application inserts a *bad*-edge if the selected nodes do not form a clique. The resulting graph is accepted if it does not contain a *bad*-edge. The state transitions of control condition $select!\ ;\ try(test(CLIQUE))$ of *CLIQUE* are visualized in Figure **??** where the nodes represent control states and the edges visualize the choice function. An edge label r means that the rule r is applicable to the current graph and a negated rule means that it is not applicable.

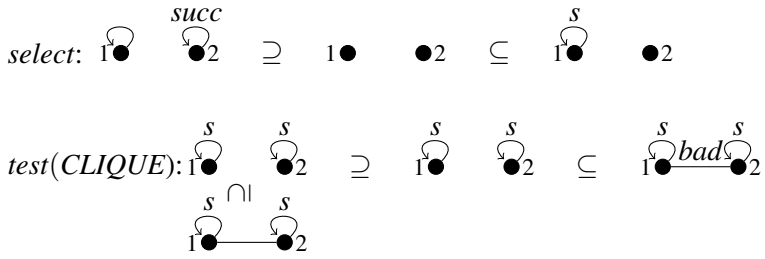
The choice function of the control condition $select!\ ;\ try(test(CLIQUE))$ is defined as

- $choice(G, select!; try(test); lambda) =$
  - $\{(G', select!; try(test); lambda) \mid G \underset{select}{\Longrightarrow} G'\}$ if $\exists G' \in \mathscr{G} : G \underset{select}{\Longrightarrow} G'$

CLIQUE

initial: $simple\&unlabeled\&looped + gr(\mathbb{N})$

rules:



control: $select! \; ; try(test(CLIQUE))$

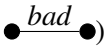
terminal: $forbidden($  $)$

Figure 1: A graph transformation unit for *clique*

- $\{(G, try(test); lambda)\}$ otherwise

- $choice(G, try(test); lambda) =$
  - $\{(G', lambda) \mid G \underset{test}{\Longrightarrow} G'\}$ if $\exists G' \in \mathscr{G} : G \underset{test}{\Longrightarrow} G'$
  - $\{(G, lambda)\}$ otherwise
- $choice(G, lambda) = \emptyset$

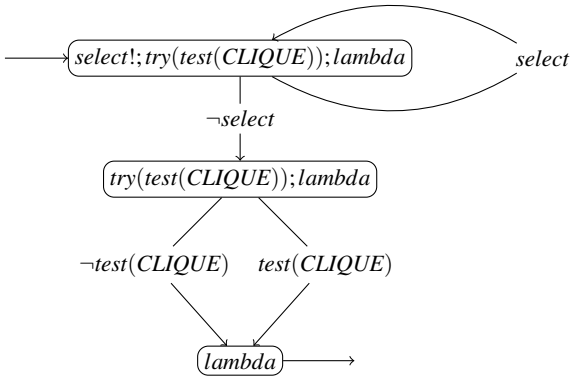


Figure 2: State Transition diagram of $select! \; ; try(test(CLIQUE))$

Each permitted derivation consists of at most $k+1$ rule applications. Hence, according to the considerations of Section 2 concerning polynomial graph transformation units, we get that *CLIQUE* is polynomial.

The semantic relation $SEM(CLIQUE)$ consists of all pairs $(G + gr(k), H + gr(0))$ such that G is simple, unlabeled and looped, and H is obtained from G by inserting an s-loop at each node of

a k-clique. Hence, $(G+gr(k),H+gr(0)) \in SEM(CLIQUE)$ if $clique(G,k) = true$. Otherwise, there is no $\hat{H} \in \mathscr{G}$ such that $(G+gr(k),\hat{H}) \in SEM(CLIQUE)$. This means that *CLIQUE* is correct.

4 Reductions in NP as Graph Transformation Units

In this section, we show how reductions in NP can be modeled by polynomial graph transformation units in a systematic way.

4.1 Reductions in NP

For $i = 1,2$, let $D_i \colon \Sigma^* \to BOOL$ be decision problems. A (polynomial) reduction from D_1 to D_2 is a function $translate \colon \Sigma^* \to \Sigma^*$ such that (1) for each $w \in \Sigma^*$, $D_1(w) = D_2(translate(w))$ and (2) *translate* can be computed by a polynomial Turing machine. Strictly speaking, translate does not need to be a function from $\Sigma^* \to \Sigma^*$. It suffices to require that it associates a nonempty set $red(w)$ with each string w such that $D_1(w) = D_2(w')$ for each $w' \in red(w)$. Hence, the polynomial Turing machine does not need to be deterministic but for each input w all computed outputs must belong to $red(w)$. This assures that if $D_1(w)$ yields *true*, then $D_2(w')$ yields *true* for each computed output w', and vice versa. It is worth noting that such a Turing machine can be easily converted into a deterministic one by ignoring at each state all but one possibilities to proceed. The set of all reductions is denoted by RED. As mentioned before, Turing machines are hard to read and that is why reductions are usually described on a higher level.

4.2 Characterizing Reductions in NP via Graph Transformation

Whenever reductions involve graphs, polynomial graph transformation units are a natural means to specify them. More precisely, a polynomial graph transformation unit $red = (I_1,P,C,I_2)$ models a reduction from $D_1 \colon SEM(I_1) \to BOOL$ to $D_2 \colon SEM(I_2) \to BOOL$ if *red* is *deadlock-free* and *correct*. Deadlock-freeness means that every permitted derivation that is not prolongable is successful. Correctness means that $D_1(G) = D_2(H)$ for each $G \in SEM(I_1)$ and each $H \in red(G)$. In this case, *red* is called a *reduction unit*. Please note that since graph transformation is highly nondeterministic reduction units are not required to be functional, i.e., for every $G \in SEM(I_1)$, there may be more than one H in $red(G)$. If D_1 and D_2 are given as graph transformation units $gtu_1 = (I_1,P_1,C_1,T_1)$ and $gtu_2 = (I_2,P_2,C_2,T_2)$, then the correctness of *red* is implied by its forward and backward correctness defined as follows.

1. *Forward correctness*: If there is a successful derivation from $G \in SEM(I_1)$ in gtu_1, then there is a successful derivation from H in gtu_2, for all $H \in red(G)$.

2. *Backward correctness*: If there is a successful derivation from H in gtu_2, where $H \in red(G)$ for some $G \in SEM(I_1)$, then there is a successful derivation from G in gtu_1.

Let RED_{GT} be the set of all reductions given as graph transformation units. Then it can be shown that RED_{GT} corresponds to RED. This means that for every reduction $translate \colon \Sigma^* \to \Sigma^*$ from D to D' in RED, there is a reduction unit *red* from $D_{\mathscr{G}}$ to $D'_{\mathscr{G}}$ in RED_{GT} where $D_{\mathscr{G}}$ and $D'_{\mathscr{G}}$

are the decision problems in $\mathrm{NP}_{\mathrm{GT}}$ corresponding to D and D', respectively. Conversely, let *red* be a reduction unit from a graph transformational decision problem D to a graph transformational decision problem D'. Then there is a reduction *translate* from D_{Str} to D'_{Str} in RED where D_{Str} and D'_{Str} are the decision problems in NP corresponding to D and D', respectively. The proof is very similar to the proof of the correspondence of $\mathrm{NP}_{\mathrm{GT}}$ and NP (cf. [KK11]) and hence omitted.

Observation 2 $\mathrm{RED}_{\mathrm{GT}} = \mathrm{RED}$.

The first point of the following proposition presents a sufficient condition for deadlock-freeness. The second point relates deadlock-freeness to the example class of stepwise control conditions presented in Section 2. It makes use of the fact that every permitted computation that cannot be prolonged ends in a final state (which is not true in general). Hence to show deadlock-freeness, it is sufficient to show that all those computations end in a terminal graph.

Proposition 1 Let $gtu = (I,P,C,T)$ be a polynomial graph transformation unit with

$$C = (S,J,F,choice).$$

1. Then *gtu* is deadlock-free, if for each permitted computation $(G_0,s_0) \vdash \cdots \vdash (G_n,s_n)$ of *gtu* with $(G_n,s_n) \notin SEM(T) \times F$, $choice(G_n,s_n) \neq \emptyset$.
2. If $C = stw(c)$ for some $c \in Try\&Alap$, then *gtu* is deadlock-free if for each permitted computation $(G_0,s_0) \vdash \cdots \vdash (G_n,s_n)$ of *gtu* $s_n = lambda$ implies $G_n \in SEM(T)$.

A deadlock-free graph transformation unit is *functional* if for every initial graph G every successful derivation from G yields the same terminal graph (up to isomorphism). The next proposition relates the functionality to control conditions. It states that if the control condition of a deadlock-free unit is based on expressions of the form $r!$ only and the rule applications of each r are locally confluent, then the unit is functional. Clearly, in forward correctness proofs of functional reduction units only one derivation has to be checked for each initial graph of the first unit.

Proposition 2 Let $gtu = (I,P,C,T)$ be a deadlock-free polynomial graph transformation unit such that $C = stw(c)$ where c is of the form $r_1!;\cdots;r_n!$ with $\{r_1,\ldots,r_n\} \subseteq P$. Then *gtu* is functional if for all $G,G_1,G_2 \in \mathscr{G}$ where G_1 and G_2 are not isomorphic: $G \underset{r}{\Longrightarrow} G_1$ and $G \underset{r}{\Longrightarrow} G_2$ implies that there is a graph $G_3 \in \mathscr{G}$ such that $G_1 \underset{r}{\Longrightarrow} G_3$ and $G_2 \underset{r}{\Longrightarrow} G_3$.

4.3 Example 1: From Cliques to Independent Sets

The graph transformation unit *CLIQUE*-to-*INDEP* in Figure 3 models a reduction from *clique* to *indep*. The decision problem *indep* gets as inputs a graph G and a natural number k. It returns *true* if and only if G has an independent set of size k, i.e., a set M of k nodes such that no two nodes of M are adjacent in G. The decision problem *indep* can be modeled by the graph transformation unit

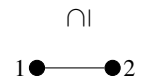
$$INDEP = (I_{CLIQUE}, \{select, test(INDEP)\}, select!;try(test(INDEP)), T_{CLIQUE})$$

CLIQUE-to-*INDEP*

initial: *simple&unlabeled&looped* + $gr(\mathbb{N})$

rules:

complement: 



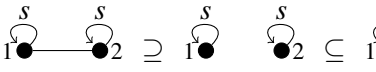
remove: 

relabel: 

control: *complement*! ; *remove*! ; *relabel*!

terminal: *simple&unlabeled&looped* + $gr(\mathbb{N})$

Figure 3: A graph transformation unit for the reduction from *clique* to *indep*

where $test(INDEP)$ is the rule  .

The rule *complement* of the unit *CLIQUE*-to-*INDEP* inserts a d-edge between all pairs of distinct nodes provided that they are not connected via an undirected edge in the initial graph. The rule *remove* deletes all original undirected edges and, finally, the rule *relabel* turns each d-edge into an unlabeled edge.

Since every derivation that cannot be prolonged reaches a terminal graph and since the rule applications of each rule are locally confluent, we get by Propositions 1 and 2 that the unit *CLIQUE-to-INDEP* is functional. The following observation concerns the successful derivations of *CLIQUE*-to-*INDEP* and can be shown by induction. It states that *CLIQUE-to-INDEP* generates for each initial graph $G + gr(k)$ the graph $H + gr(k)$ where H is the complement graph of G.

Observation 3 Let $G \in \mathscr{G}$. Then $G \overset{*}{\Longrightarrow} H$ is a successful derivation of *CLIQUE*-to-*INDEP* if and only if H is obtained from G by inserting an unlabeled undirected edge between each pair of nodes that is not connected via an unlabeled undirected edge and by deleting all original unlabeled undirected edges.

In every successful derivation, each of the three rules is applied at most n^2 times where n is the number of nodes of the initial graph. Hence, taking into account the considerations of Section 2 we get that *CLIQUE*-to-*INDEP* is polynomial.

4.4 Example 2: From Vertex Covers to Hamiltonian Cycles

In the following, a more sophisticated example of a reduction is presented.

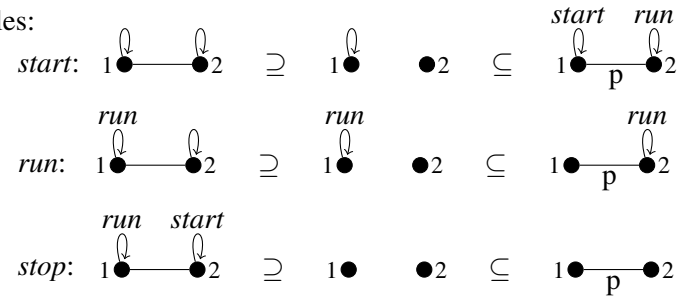
Let *VC* be the graph transformation unit

$$(I_{CLIQUE}, \{select, test(VC)\}, select!; try(test(VC)), T_{CLIQUE})$$

HC

initial: *simple&unlabeled&looped*

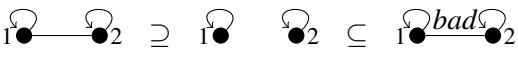
rules:



control: $start; run!; stop$

terminal: *forbidden*( ,  *start*)

Figure 4: A graph transformation unit for Hamiltonian cycles

where $test(VC)$ is the rule  . This unit *VC* is the graph transformational version of the decision problem *vc* with a graph G and a natural number k as inputs. It returns *true* if and only if G has a vertex cover of size k, i.e., a set of k nodes so that every edge is incident to at least one of these nodes.

The graph transformation unit *HC* in Figure 4 models the decision problem *hc* the input of which is a graph G. It returns *true* if and only if G has a Hamiltonian cycle, i.e., a cycle that visits every node exactly once.

The following unit *VC*-to-*HC* models a reduction from *VC* to *HC*. It is based on the construction presented in [GJ79].

VC-to-*HC*

initial: *simple&unlabeled&looped* + $gr(\mathbb{N})$

rules: $\{r_1, \ldots, r_{11}\}$

control: $r_1! \,;\, r_2 \,;\, r_3!; \cdots ; r_8!; r_9(n)!; r_9(b)!; r_{10}!$

terminal: *simple&unlabeled&looped*

The rules of the unit *VC*-to-*HC* are depicted in Figure 5, 6, and 7.

According to the control condition the first rule is applied as long as possible before trying to apply the second rule once. This converts the graph $gr(k)$ into k n-nodes. The third rule generates a $\{u,v\}$-*edge-ladder*, two 6-edges and two c-edges for each subset $\{u,v\}$ of distinct nodes that are adjacent in the initial graph. After applying it as long as possible each initial node v is connected by a c-edge to l different ladders where l is the number of undirected edges incident to v. The target of a c-edge originating from v is called a v-*entry* and the target of the 6-edge starting from a v-entry is a v-*exit*. (The 6-edges are for remembering in further rules which exits belong to which entries.) A ladder with a v-entry is also called a v-*ladder*. The fourth rule, applied as long as possible, chooses one c-edge for each (non-isolated) initial node v replacing it by an a-edge; hence, this rule selects one $\{u,v\}$-ladder for each initial node v. The rule r_5 connects the v-entry of each selected ladder to each n-node. The rule r_6 selects for each initial

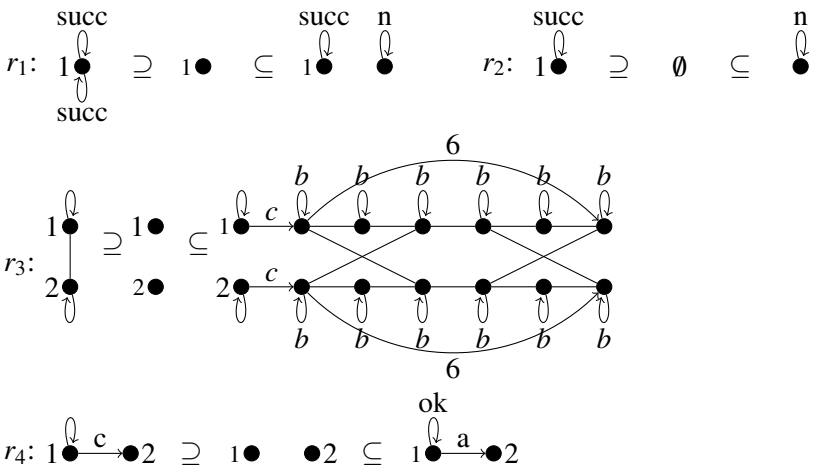


Figure 5: The rules $r_1, \ldots, r_4$ of the unit *VC*-to-*HC*

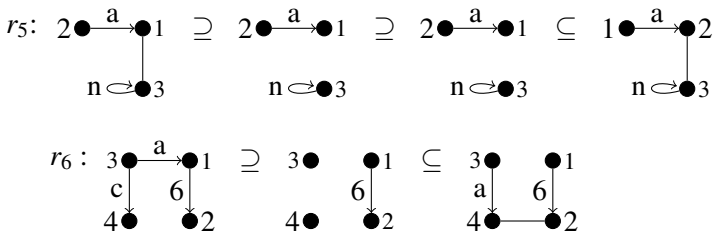


Figure 6: The rules r_5 and r_6 of the unit *VC*-to-*HC*

node v a not yet selected v-ladder and connects the v-exit of the previously selected ladder to the v-entry of this ladder. This is repeated as long as possible so that finally all v-ladders are selected.

Rule r_7 connects for each initial node v the v-exit of the last selected ladder to each n-node. The rule r_8 removes all initial nodes together with the attached loops and the incident a-edges; with the rule r_9 every n-loop and every b-loop is turned into an unlabeled loop; r_{10} deletes all 6-edges and r_{11} all isolated initial nodes.

The next observation concerns the successful derivations of *VC*-to-*HC*.

Observation 4 Let $(G + gr(k)) \in SEM(I_{VC})$ and let $H \in \mathscr{G}$. Then $H \in VC\text{-to-}HC(G + gr(k))$ if and only if H consists of the following components:

- A $\{u,v\}$-ladder for each set $\{u,v\}$ of distinct adjacent nodes in G,
- k nodes, say $1, \ldots, k$ (not being part of a ladder),
- for each $v \in V_G$ there is some ordering $l^v_1, \ldots, l^v_{m(v)}$ of the set of v-ladders such that for $j = 1, \ldots, m(v) - 1$ the v-exit of l^v_j is adjacent to the v-entry of l^v_{j+1} and every node in $[k]$ is adjacent to the v-entry of l^v_1 as well as to the v-exit of $l^v_{m(v)}$. [2]

[2] $[k]$ denotes the set $\{1, \ldots, k\}$.

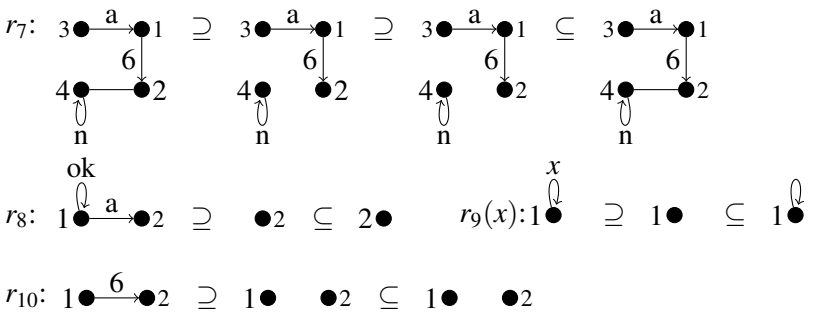


Figure 7: The rules $r_7,\ldots,r_{11}$ of the unit *VC*-to-*HC*

The nodes $1,\ldots,k$ will also be called *clip nodes*.

Since every permitted derivation that cannot be prolonged ends in a terminal graph we get by Proposition 1 that *VC*-to-*HC* is deadlock-free. By Observation 4, it is not functional. Moreover, since each successful derivation consists of a polynomial number of steps, we get together with the considerations in Section 2 that *VC*-to-*HC* is polynomial.

5 Correctness

The following proposition is useful for proving forward correctness of reductions and provides a first step towards a proof scheme for correctness of reductions. Roughly spoken, it states that the existence of a set *Aux* of deadlock-free graph transformation units leads to the forward correctness of a reduction from gtu_1 to gtu_2 if the units in *Aux* satisfy certain compatibility conditions that may be seen as stepwise correctness.

Proposition 3 For $i = 1,2$, let $gtu_i = (I_i, P_i, C_i, T_i)$ be polynomial graph transformation units. Let $red = (I_1, P, C, I_2)$ be a deadlock-free polynomial graph transformation unit. Then *red* is a forward correct unit from gtu_1 to gtu_2, if there is a set *Aux* of deadlock-free polynomial graph transformation units such that for each successful derivation $G_0 \Longrightarrow \cdots \Longrightarrow G_n$ in gtu_1 and each $H_0 \in red(G_0)$ there are units $aux_1,\ldots,aux_n \in Aux$ and graphs $H_1,\ldots,H_n \in \mathscr{G}$ such that the following hold:

1. For $i = 1,\ldots,n$, the graph H_i is in $aux_i(G_i)$, and there is a derivation $der_i = (H_{i-1} \overset{*}{\Longrightarrow} H_i)$ such that the sequential composition $der_1 \cdots der_n$ is permitted in gtu_2.

2. $H_n \in SEM(T_2)$.

Remarks.

1. This proposition allows to prove forward correctness by induction on the length of the permitted derivations that can be prolonged to successful derivations. To this aim, stepwise control is essential.

2. Let $\widehat{red}$ be a deadlock-free polynomial unit from gtu_2 to gtu_1 such that for each $G \in SEM(I_1)$ and each $H \in red(G)$ with a successful derivation in gtu_2, $G \in \widehat{red}(H)$. Then forward correctness of $\widehat{red}$ implies backward correctness of red.

5.1 Correctness of *CLIQUE*-to-*INDEP*

Let $der = (G_0 \overset{n}{\Longrightarrow} G_n)$ be a successful derivation of *CLIQUE*. Then in every derivation step the rule *select* is applied. Let $H_0 \in$ *CLIQUE-to-INDEP*(G_0). Then due to the functionality of *CLIQUE-to-INDEP*, H_0 is the unique graph in *CLIQUE-to-INDEP*(G_0). Let *aux* be the unique auxiliary unit defined as

$$aux = (all, P_{\text{CLIQUE-to-INDEP}}, C_{\text{CLIQUE-to-INDEP}}, all),$$

where $SEM(all) = \mathscr{G}$. Please note that *aux* is used for transforming the graphs $G_1, \ldots, G_n$ which are not in $SEM(I_{CLIQUE})$. Hence, the initial graph class expression of *aux* is more general than that of *CLIQUE*. For similar reasons the terminal expression of *aux* is more general than the initial expression of *INDEP*. By Propositions 1 and 2, *aux*′ is functional. Moreover, for $i = 1, \ldots, n$, $red(G_i)$ consists of the complement of the underlying simple graph of G_i plus the loops of G_i. We assume without loss of generality that the node set of the graph in $aux(G_i)$ is equal to V_{G_i}.

If $n = 0$ then *CLIQUE-to-INDEP*$(G_n) = \{H_0\}$ and $H_0 \overset{*}{\Longrightarrow} H_0$ is permitted in *INDEP*. Now assume that for some $n \in \mathbb{N}$, $H_0 \overset{*}{\underset{select}{\Longrightarrow}} H_n$ is a derivation in *INDEP* where $H_n \in aux(G_n)$ if $n > 0$ and $H_n \in$ *CLIQUE-to-INDEP*(G_n) if $n = 0$. Let $G_n \underset{select}{\Longrightarrow} G_{n+1}$. Then G_n contains a node v, an unlabeled loop of which is replaced by an s-loop and a *succ*-loop which is deleted. Then v has also an unlabeled loop in H_n and the *succ*-loop is also present in H_n. Hence there is a derivation step $H_n \underset{select}{\Longrightarrow} H_{n+1}$ in which the rule *select* is applied to v and the *succ*-loop. Since *select* only affects loops and *aux* does not affect loops, $H_{n+1} \in aux(G_{n+1})$. Since $H_0 \overset{*}{\underset{select}{\Longrightarrow}} H_{n+1}$ is permitted, Point 1 of Proposition 3 is satisfied.

If $G_n \in SEM(T_{CLIQUE})$, then $test(INDEP))$ is not applicable to H_n because otherwise, there would be an edge between s-nodes in H_n which is not present in G_n. This means that $test(CLIQUE)$ would be applicable to G_n, i.e., $G_n \notin SEM(T_{CLIQUE})$ which is a contradiction. Hence, $H_n \in SEM(T_{INDEP})$, i.e., the second condition of the proposition is also satisfied.

Hence, *CLIQUE-to-INDEP* is forward correct. The situation is illustrated in Figure 8.

If *CLIQUE-to-INDEP* is applied twice we obtain the original graph again, i.e., the complement of the complement of a graph G equals G. Hence, due to the second remark of Observation 3 *CLIQUE-to-INDEP* is also backward correct. This leads to the following observation.

Observation 5 The unit *CLIQUE-to-INDEP* is correct.

5.2 Forward Correctness of *VC*-to-*HC*

Let $G_0 = (G_0' + gr(k)) \in SEM(I_{VC})$ and let $G_0 \overset{n}{\underset{select}{\Longrightarrow}} G_n$. Let $H_0 \in$ *VC*-to-*HC*(G_0) and for $v \in V_{G_0'}$ let $l_1^v, \ldots, l_{m(v)}^v$ be the ordering of the v-ladders in H_0 and let $c_1, \ldots, c_n$ be clip nodes (cf. Obser-

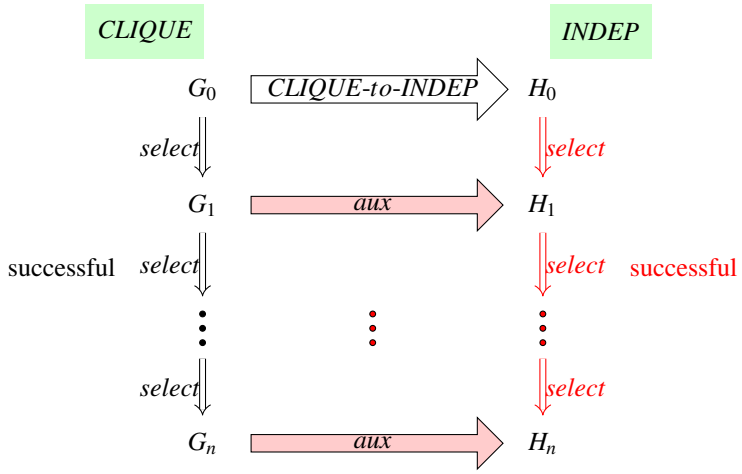


Figure 8: Forward correctness of *CLIQUE-to-INDEP*

vation 4).[3] Then for each ordering $v_1,\ldots,v_n$ of the s-nodes in G_n the sequence

$$(c_1, l_1^{v_1},\ldots,l_{m(v_1)}^{v_1},\ldots,c_n,l_1^{v_n},\ldots,l_{m(v_n)}^{v_n})$$

induces a set $Paths_n$ consisting of all paths in H_0 that visit nodes $c_1,\ldots,c_n$ in this order so that after each c_i the ladders $l_1^{v_i},\ldots,l_{m(v_i)}^{v_i}$ are visited in this order. In more detail, for $j=1,\ldots,m(v_i)$ the ladder $l_j^{v_i}$ is passed straight from its v_i-entry to its v_i-exit if the other entry of the ladder (i.e., the entry not equal to the v_i-entry) is an s-node; otherwise it is passed straight or zigzag from its v_i-entry to its v_i-exit. The ladders depicted in Figure 9 illustrate the courses of the straight and zigzag paths. The top nodes of the ladders represent their entries and the bottom nodes their exits.

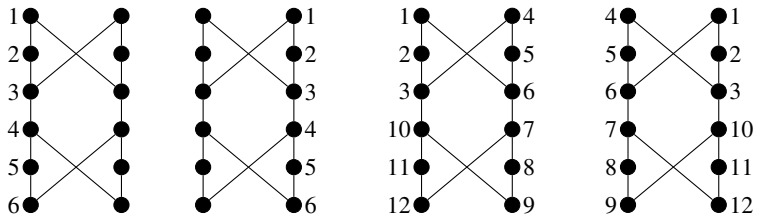


Figure 9: Straight and zigzag paths through ladders

We can construct a polynomial deadlock-free graph transformation unit *red* such that for each graph G derivable from some $G_0 \in SEM(I_{VC})$ by successive applications of the rule *select*, $red(G)$ consists of all graphs H which can be obtained from some $H_0 \in VC\text{-to-}HC(G_0)$ by highlighting one of the described paths in such a way that all edges on the path are labeled with p the first clip node has a *start*-loop and all other nodes on the path have a *run*-loop. For reasons

[3] Since $n \leq k$ these clip nodes exist.

of space limitations *red* is not presented here; but it is worth noting that it is quite similar to the unit *VC*-to-*HC* although needing more complicated control conditions. By induction on n it can be shown that for every $H_0 \in VC\text{-to-}HC(G_0)$ there is a derivation $H_0 \stackrel{*}{\Longrightarrow} H_n$ with application sequence $init\, start\, run^{n-1}$ and $H_n \in red(G_n)$ if $n > 0$. If $n = 0$ the application sequence is equal to λ.

Moreover, if G_n is a terminal graph, then $test(VC)$ is not applicable to G_n i.e., the path in H_n contains all ladders and all clip nodes of H_0 (remember that k cannot be larger than the number of nodes in G'_0). Hence the application of *stop* to the last and the first node of the path yields a terminal graph of *HC*.

Altogether we get that *VC*-to-*HC* satisfies the conditions for the forward correctness and hence the following observation is holds.

Observation 6 The unit *VC*-to-*HC* is forward correct.

6 Conclusion

In this paper, we have shown how reductions in NP can be modeled by graph transformation units in a visual and formal way. In particular, we have presented a first step towards a proof scheme for the correctness of reduction units. It turned out that the presented approach is suitable to specify and prove the (forward) correctness of two well-known reductions where the latter is rather complex.

In the future, we want to undertake further steps in the following directions. (1) The presented proof scheme for forward correctness is based on the correctness of a set of auxiliary units which in turn can be shown by induction. Hence, an interesting question is how the induction proof of the forward correctness can be interwoven with the induction proofs of the auxiliary units in a systematic way. (2) The first of our two reduction examples is backward correct because the reduction can be done the other way round. However, the proof of backward correctness should be integrated in the presented proof scheme in a systematic way. (cf. also [EEHP09]). (3) Since reductions are a special kind of model transformations we would like to investigate how the presented ideas can be used to prove correctness of model transformations, in general. To this aim, the presented ideas should be related to other approaches to correctness proofs of model transformations based of graph transformations. In particular, we will compare our results with those obtained in the field of model transformations using triple grammars. (4) In addition to the considered class of stepwise control conditions based on *try* and *as-long-as-possible*, we want to find out whether more general control conditions are suitable for our purposes (see, e.g., [FNTZ00, Kus00, Plu09]).

Acknowledgment. We are grateful to Hans-Jörg Kreowski for his helpful comments.

Bibliography

[CEH+97] 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. Pp. 163–245 in [Roz97].

[EEHP09] H. Ehrig, C. Ermel, F. Hermann, U. Prange. On-the-Fly Construction, Correctness and Completeness of Model Transformations Based on Triple Graph Grammars. In Schürr and Selic (eds.), *12th International Conference on Model Driven Engineering Languages and Systems(MoDELS 2009)*. Lecture Notes in Computer Science 5795, pp. 241–255. 2009.

[FNTZ00] T. Fischer, J. Niere, L. Torunski, A. Zündorf. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In Ehrig et al. (eds.), *Proc. 6th International Workshop on Theory and Application of Graph Transformations (TAGT'98), Selected Papers*. Lecture Notes in Computer Science 1764, pp. 296–309. Springer, 2000.

[GJ79] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae* 26(3,4):287–313, 1996.

[HMU07] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to automata theory, languages, and computation*. Pearson/Addison Wesley, 2007.

[KK11] H.-J. Kreowski, S. Kuske. Graph Multiset Transformation – A New Framework for Massively Parallel Computation Inspired by DNA Computing. *Natural Computing* 10(2):961–986, 2011.

[KK12] H.-J. Kreowski, S. Kuske. Polynomial Graph Transformability. *Theoretical Computer Science* 429:193–201, 2012.

[KKR08] H.-J. Kreowski, S. Kuske, G. Rozenberg. Graph Transformation Units – An Overview. In Degano et al. (eds.), *Concurrency, Graphs and Models*. Lecture Notes in Computer Science 5065, pp. 57–75. Springer, 2008.

[Kus00] S. Kuske. More about control conditions for transformation units. In Ehrig et al. (eds.), *Proc. 6th International Workshop on Theory and Application of Graph Transformations (TAGT'98), Selected Papers*. Lecture Notes in Computer Science 1764, pp. 323–337. 2000.

[Plu09] D. Plump. The Graph Programming Language GP. In *Proc. Algebraic Informatics, Third International Conference (CAI 2009)*. Lecture Notes in Computer Science 5725, pp. 99–122. Springer-Verlag, 2009.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, Singapore, 1997.