# Algebraic Graph Transformation: A Crash Course

Graham Campbell

October 27, 2018

## 1 Graphs and Morphisms[1]

### 1.1 Unlabelled Graphs

**Definition 1.1.** We can formally define a **concrete graph** as:

$$G = (V, E, s : E \to V, t : E \to V)$$

where $V$ is a finite set of vertices, $E$ is a finite set of edges. We call $s : E \to V$ the source function, and $t : E \to V$ the target function.

**Definition 1.2.** Given two concrete graphs $G$ and $H$, a **graph morphism** $g : G \to H$ is a pair of mappings:

$$g = (g_V : V_G \to V_H, g_E : E_G \to E_H)$$

such that sources and targets are preserved. That is, these squares commute:

$$\begin{array}{ccc} E_G & \xrightarrow{s_G} & V_G \\ \downarrow{\scriptstyle g_E} & & \downarrow{\scriptstyle g_V} \\ E_H & \xrightarrow{s_H} & V_H \end{array} \qquad \begin{array}{ccc} E_G & \xrightarrow{t_G} & V_G \\ \downarrow{\scriptstyle g_E} & & \downarrow{\scriptstyle g_V} \\ E_H & \xrightarrow{t_H} & V_H \end{array}$$

**Definition 1.3.** A graph morphism $g : G \to H$ is **injective/surjective** iff both $g_V$ and $g_E$ are injective/surjective as functions. We say $g$ is **bijective** iff it is both injective and surjective.

**Definition 1.4.** We say that graphs $G, H$ are **isomorphic** iff there exists a bijective graph morphism $g : G \to H$, and we write $G \cong H$. This naturally gives rise to **equivalence classes** $[G]$, called **abstract graphs**.

[1]The definitions and results are derived from [1].

## 1.2 Labelled Graphs

**Definition 1.5.** A label alphabet $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ consists of a set $\mathcal{L}_V$ of **node labels**, and $\mathcal{L}_E$ a set of **edge labels**.

**Definition 1.6.** A **concrete labelled graph** over a label alphabet $\mathcal{L}$ is a concrete graph equipped with two label maps $l : V \to \mathcal{L}_V$, $m : E \to \mathcal{L}_E$:

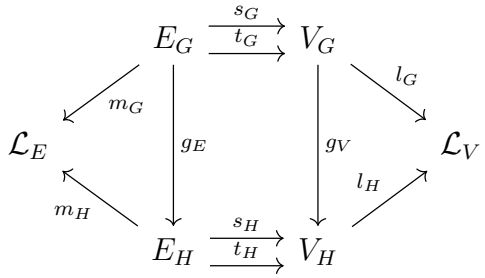
$$G(\mathcal{L}) = (V, E, s, t, l, m)$$

Diagrammatically, we have:

$$\mathcal{L}_E \xleftarrow[m]{} E \underset{t}{\overset{s}{\rightrightarrows}} V \xrightarrow{l} \mathcal{L}_V$$

**Remark 1.1.** By this definition, we **do not** work with the free monoid on the alphabet, as in string rewriting systems. Nodes and edges are labelled exactly with the elements from the respective alphabets themselves. We do not require alphabets to be finite, since images of label maps must be finite.

**Definition 1.7.** Given a common alphabet $\mathcal{L}$, a **labelled graph morphism** $g : G(\mathcal{L}) \to H(\mathcal{L})$ is a graph morphism on the underlying concrete graphs, with the extra constraint that labels must be preserved. The following diagram must commute (for $s_G, s_H$ and $t_G, t_H$ separately, as in Definition 1.2):




**Definition 1.8.** Given a common alphabet $\mathcal{L}$, a labelled graph morphism $g : G(\mathcal{L}) \to H(\mathcal{L})$ is **injective/surjective** iff the underlying graph morphism is injective/surjective. We define **isomorphism classes** in the same manner.

**Definition 1.9.** Given a common alphabet $\mathcal{L}$, we say $H(\mathcal{L})$ is a **subgraph** of $G(\mathcal{L})$ iff there exists an **inclusion morphism** $H(\mathcal{L}) \hookrightarrow G(\mathcal{L})$. This happens iff $V_H \subseteq V_G$, $E_H \subseteq E_G$, $s_H = s_G\big|_{E_H}$, $t_H = t_G\big|_{E_H}$, $l_H = l_G\big|_{V_H}$, $m_H = m_G\big|_{E_H}$.

**Remark 1.2.** The empty graph $\emptyset$ is trivially a subgraph of every graph.

# 2 Graph Rewriting

## 2.1 Rules and Derivations

We shall assume that all graphs are concrete and labelled from now on, writing simply $G$ for a labelled concrete graph over some label alphabet.

**Definition 2.1.** A **rule** $r = \langle L \leftarrow K \rightarrow R \rangle$ over $\mathcal{L}$ consists of graphs $L$, $K$ and $R$ over $\mathcal{L}$, and inclusions $K \hookrightarrow L$ and $K \hookrightarrow R$.

**Remark 2.1.** We define the **inverse rule** as $r^{-1} = \langle R \leftarrow K \rightarrow L \rangle$.

**Definition 2.2.** To **apply** a rule $r = \langle L \leftarrow K \rightarrow R \rangle$ to some graph $G$, we:

1. Find an **injective** graph morphism $g : L \hookrightarrow G$;
2. Check that no edge in $G \setminus (Lg)$ is incident to a node in $(L \setminus K)g$;
3. Delete $(L \setminus K)g$, giving an intermediate graph $D$;
4. Add $R \setminus K$ to D, giving the **result graph** $H$.

If the **dangling condition** fails, then the rule is not applicable using the **match** $g$. We can exhaustively check all matches to determine applicability.

**Definition 2.3.** We write $G \Rightarrow_{r,g} H$ for a successful application of $r$ to $G$ using match $g$, obtaining result graph $H$. We call this a **direct derivation**.

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ \downarrow{\scriptstyle g} & & \downarrow{\scriptstyle d} & & \downarrow{\scriptstyle h} \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

We call injective morphism $h$ the **comatch**. Every derivation $G \Rightarrow_{r,g} H$ may be reversed, using the comatch, giving $H \Rightarrow_{r^{-1},h} G$.

**Remark 2.2.** It turns out that **gluings** and **deletions** are **pushouts** in the category of labelled graphs. Moreover, direct derivations are **double pushouts**, and $H$ is unique up to isomorphism.

**Definition 2.4.** For a **finite** set of rules $\mathcal{R}$, we write $G \Rightarrow_{\mathcal{R}} H$ when $H$ is **directly derived** from $G$ using any of the rules from $\mathcal{R}$.

**Definition 2.5.** We write $G \Rightarrow_{\mathcal{R}}^{+} H$ when $H$ is **derived** from $G$ in one or more direct derivations, and $G \Rightarrow_{\mathcal{R}}^{*} H$ iff $G \cong H$ or $G \Rightarrow_{\mathcal{R}}^{+} H$.

## 2.2 Transformation Systems

**Definition 2.6.** A **graph transformation system** $(\mathcal{L}, \mathcal{R})$, consists of a label alphabet $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$, and a finite set $\mathcal{R}$ of rules over $\mathcal{L}$.

**Definition 2.7.** Given a label alphabet $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$, $\mathcal{P} = (\mathcal{P}_V, \mathcal{P}_E)$ is a **subalphabet** of $\mathcal{L}$ iff $\mathcal{P}_V \subseteq \mathcal{L}_V$ and $\mathcal{P}_E \subseteq \mathcal{L}_E$.

**Definition 2.8.** Given a graph transformation system $(\mathcal{L}, \mathcal{R})$, a subalphabet of **non-terminals** $\mathcal{N}$, and a **start graph** $S$ over $\mathcal{L}$, then a **graph grammar** is the system $\mathcal{G} = (\mathcal{L}, \mathcal{N}, \mathcal{R}, S)$.

**Definition 2.9.** Given a graph grammar $\mathcal{G}$ as defined above, we say that a graph $G$ is **terminally labelled** iff $Vl \cap \mathcal{N}_V = \emptyset$ and $Em \cap \mathcal{N}_E = \emptyset$. Thus, we can define the **graph language** generated by $\mathcal{G}$:

$$L(\mathcal{G}) = \{G \mid S \Rightarrow_{\mathcal{R}}^{*} G, \text{G terminally labelled}\}$$

**Remark 2.3.** Graph languages need not be finite. In fact, graph grammars are as powerful as unrestricted string grammars. As such, many questions like if the language is empty are undecidable in general.

## 2.3 Confluence and Termination

Let $\mathcal{G} = (\mathcal{L}, \mathcal{R})$ be a graph transformation system.

**Definition 2.10.** $\mathcal{G}$ is **strongly confluent** iff for all graphs $G, H_1, H_2$, $H_1 \Leftarrow_{\mathcal{R}} G \Rightarrow_{\mathcal{R}} H_2$ implies that either $H_1 \cong H_2$, or there is a graph $M$ such that $H_1 \Rightarrow_{\mathcal{R}} M \Leftarrow_{\mathcal{R}} H_2$.

**Definition 2.11.** $\mathcal{G}$ is **confluent** iff for all $G, H_1, H_2$, $H_1 \Leftarrow_{\mathcal{R}}^{*} G \Rightarrow_{\mathcal{R}}^{*} H_2$ implies that there is a graph $M$ such that $H_1 \Rightarrow_{\mathcal{R}}^{*} M \Leftarrow_{\mathcal{R}}^{*} H_2$.

**Definition 2.12.** $\mathcal{G}$ is **locally confluent** iff for all $G, H_1, H_2$, $H_1 \Leftarrow_{\mathcal{R}} G \Rightarrow_{\mathcal{R}} H_2$ implies that there is a graph $M$ such that $H_1 \Rightarrow_{\mathcal{R}}^{*} M \Leftarrow_{\mathcal{R}}^{*} H_2$.

**Definition 2.13.** $\mathcal{G}$ is **terminating** iff there is no infinite derivation sequence:

$$G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} G_2 \Rightarrow_{\mathcal{R}} G_3 \Rightarrow_{\mathcal{R}} \cdots$$

**Remark 2.4.** Determining confluence and termination of graph rewriting systems is undecidable in general. The same is true of term rewriting systems.

# References

[1] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Berlin, Heidelberg: Springer-Verlag, 2006, ISBN: 3540311874.