Efficient Graph Rewriting

York Semigroup

Graham Campbell

May 2019

Unlabelled Graphs I

Definition 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.

Unlabelled Graphs I

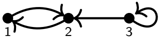
Definition 1

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.

Example: $G = (\{1, 2, 3\}, \{a, b, c, d\}, s, t)$ where $s = \{(a, 1), (b, 2), (c, 3), (d, 3)\}$, $t = \{(a, 2), (b, 1), (c, 1), (d, 3)\}$.



Unlabelled Graphs I

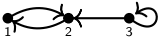
Definition 1

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.

Example: $G = (\{1, 2, 3\}, \{a, b, c, d\}, s, t)$ where $s = \{(a, 1), (b, 2), (c, 3), (d, 3)\}$, $t = \{(a, 2), (b, 1), (c, 1), (d, 3)\}$.



Definition 2

A **graph morphism** $g : G \to H$ is a pair $(g_V : V_G \to V_H, g_E : E_G \to E_H)$ such that sources and targets are preserved. That is, $\forall e \in E_G$, $g_V(s_G(e)) = s_H(g_E(e))$ and $g_V(t_G(e)) = t_H(g_E(e))$.

Unlabelled Graphs II

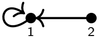
Definition 3

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 an **isomorphism** iff it is both injective and surjective.

Unlabelled Graphs II

Definition 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 an **isomorphism** iff it is both injective and surjective.

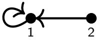
$G =$  $H =$ 

There are four morphisms $G \to H$, three of which are injective, none of which are surjective. There are actually also four morphisms $H \to G$, three of which are surjective.

Unlabelled Graphs II

Definition 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 an **isomorphism** iff it is both injective and surjective.

$G =$ 

$H =$ 

There are four morphisms $G \to H$, three of which are injective, none of which are surjective. There are actually also four morphisms $H \to G$, three of which are surjective.

Definition 4

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

Labelled Graphs I

Definition 5

A label alphabet $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ consists of **finite** sets of **node labels** $\mathcal{L}_V$ and **edge labels** $\mathcal{L}_E$.

Labelled Graphs I

Definition 5

A label alphabet $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ consists of **finite** sets of **node labels** $\mathcal{L}_V$ and **edge labels** $\mathcal{L}_E$.

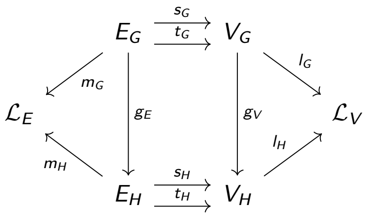
Definition 6

A **concrete labelled graph** over a label alphabet $\mathcal{L}$ is a tuple $G = (V, E, s, t, l, m, p)$ where:

1. V is a **finite** set of **vertices**;
2. E is a **finite** set of **edges**;
3. $s : E \to V$ is a source function;
4. $t : E \to V$ is a target function;
5. $l : V \to \mathcal{L}_V$ is the node labelling function;
6. $m : E \to \mathcal{L}_E$ is the edge labelling function;

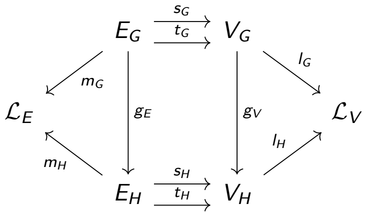
Labelled Graphs II

For morphisms between labelled graphs, we require that labels are preserved: $\forall v \in V_G,\ l_G(v) = l_H(g_V(v))$ and $\forall e \in E_G,\ m_G(e) = m_H(g_E(e))$.



Labelled Graphs II

For morphisms between labelled graphs, we require that labels are preserved: $\forall v \in V_G,\ l_G(v) = l_H(g_V(v))$ and $\forall e \in E_G,\ m_G(e) = m_H(g_E(e))$.



Definition 7

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

Rules

Let $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ be the ambient label alphabet, and graphs be concrete.

Definition 8

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

Rules

Let $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ be the ambient label alphabet, and graphs be concrete.

Definition 8

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

Definition 9

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

Rules

Let $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ be the ambient label alphabet, and graphs be concrete.

Definition 8

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

Definition 9

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

Definition 10

If $r = \langle L \leftarrow K \rightarrow R \rangle$ is a **rule**, then $|r| = max\{|L|, |R|\}$, where the size of a graph G is $|G| = |V_G| + |E_G|$.

Rule Application

Definition 11

Given a **rule** $r = \langle L \leftarrow K \rightarrow R \rangle$ and a **labelled graph** G, we say that an **injective** morphism $g : L \hookrightarrow G$ satisfies the **dangling condition** iff no edge in $G \setminus g(L)$ is incident to a node in $g(L \setminus K)$.

Rule Application

Definition 11

Given a **rule** $r = \langle L \leftarrow K \rightarrow R \rangle$ and a **labelled graph** G, we say that an **injective** morphism $g : L \hookrightarrow G$ satisfies the **dangling condition** iff no edge in $G \setminus g(L)$ is incident to a node in $g(L \setminus K)$.

Definition 12

To **apply** a rule r to some **labelled graph** G, find an **injective** graph morphism $g : L \hookrightarrow G$ satisfying the **dangling condition**, then:

1. Delete $g(L \setminus K)$, giving the **intermediate graph** D;
2. Add disjointly $R \setminus K$ to D, giving the **result graph** H.

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

Direct Derivations

Definition 13

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

Direct Derivations

Definition 13

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

Theorem 14 (Derivation Uniqueness)

It turns out that **deletions** *are* **natural pushout complements** *and* **gluings** *are* **natural pushouts** *in the category of labelled graphs. Moreover, direct derivations are* **natural double pushouts**, *D and H are* **unique up to isomorphism**, *and derivations $G \Rightarrow_{r,g} H$ are* **invertible**.

Direct Derivations

Definition 13

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

Theorem 14 (Derivation Uniqueness)

It turns out that **deletions** *are* **natural pushout complements** *and* **gluings** *are* **natural pushouts** *in the category of labelled graphs. Moreover, direct derivations are* **natural double pushouts**, *D and H are* **unique up to isomorphism**, *and derivations $G \Rightarrow_{r,g} H$ are* **invertible**.

Definition 15

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

Graph Transformation

Definition 16

A **graph transformation system** (**GT system**) is a pair $T = (\mathcal{L}, \mathcal{R})$ where $\mathcal{L}$ is a **label alphabet** and $\mathcal{R}$ is a **finite** set of **rules**.

Graph Transformation

Definition 16

A **graph transformation system** (**GT system**) is a pair $T = (\mathcal{L}, \mathcal{R})$ where $\mathcal{L}$ is a **label alphabet** and $\mathcal{R}$ is a **finite** set of **rules**.

Definition 17

Let $\mathcal{L}$ be some fixed label alphabet. Then we let $\mathcal{G}(\mathcal{L})$ be the **countable set** of all **labelled abstract graphs**.

Graph Transformation

Definition 16

A **graph transformation system** (**GT system**) is a pair $T = (\mathcal{L}, \mathcal{R})$ where $\mathcal{L}$ is a **label alphabet** and $\mathcal{R}$ is a **finite** set of **rules**.

Definition 17

Let $\mathcal{L}$ be some fixed label alphabet. Then we let $\mathcal{G}(\mathcal{L})$ be the **countable set** of all **labelled abstract graphs**.

Definition 18

Let $T = (\mathcal{L}, \mathcal{R})$ be a **GT system**. Then $(\mathcal{G}(\mathcal{L}), \to_{\mathcal{R}})$ is the induced **ARS** defined by $\forall [G], [H] \in \mathcal{G}(\mathcal{L}), [G] \to_{\mathcal{R}} [H]$ iff $G \Rightarrow_{\mathcal{R}} H$.

Graph Transformation

Definition 16

A **graph transformation system** (**GT system**) is a pair $T = (\mathcal{L}, \mathcal{R})$ where $\mathcal{L}$ is a **label alphabet** and $\mathcal{R}$ is a **finite** set of **rules**.

Definition 17

Let $\mathcal{L}$ be some fixed label alphabet. Then we let $\mathcal{G}(\mathcal{L})$ be the **countable set** of all **labelled abstract graphs**.

Definition 18

Let $T = (\mathcal{L}, \mathcal{R})$ be a **GT system**. Then $(\mathcal{G}(\mathcal{L}), \to_{\mathcal{R}})$ is the induced **ARS** defined by $\forall [G], [H] \in \mathcal{G}(\mathcal{L}), [G] \to_{\mathcal{R}} [H]$ iff $G \Rightarrow_{\mathcal{R}} H$.

Lemma 19

Consider the **ARS** $(\mathcal{G}(\mathcal{L}), \to)$ *induced by a* **GTS**. *Then* $\to$ *is a* **binary relation** *on* $\mathcal{G}(\mathcal{L})$ *(that is, it is both well-defined and closed). Moreover, it is* **finitely branching** *and* **decidable**.

Graph Grammars

Definition 20

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

Graph Grammars

Definition 20

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

Definition 21

Given a **graph grammar** $\boldsymbol{G}$ as defined above, we say that a graph G is **terminally labelled** iff $l(V) \cap \mathcal{N}_V = \emptyset$ and $m(E) \cap \mathcal{N}_E = \emptyset$. Thus, we can define the **graph language** generated by $\boldsymbol{G}$:

$$\boldsymbol{L}(\boldsymbol{G}) = \{[G] \mid [S] \rightarrow^*_{\mathcal{R}} [G], G \text{ terminally labelled}\}$$

Graph Grammars

Definition 20

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

Definition 21

Given a **graph grammar** $\boldsymbol{G}$ as defined above, we say that a graph G is **terminally labelled** iff $l(V) \cap \mathcal{N}_V = \emptyset$ and $m(E) \cap \mathcal{N}_E = \emptyset$. Thus, we can define the **graph language** generated by $\boldsymbol{G}$:

$$\boldsymbol{L}(\boldsymbol{G}) = \{[G] \mid [S] \rightarrow^*_{\mathcal{R}} [G], G \text{ terminally labelled}\}$$

Theorem 22 (Membership Test)

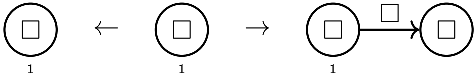
Given a **grammar** $\boldsymbol{G} = (\mathcal{L}, \mathcal{N}, \mathcal{R}, S)$, $[G] \in \boldsymbol{L}(\boldsymbol{G})$ *iff* $[G] \rightarrow^*_{\mathcal{R}^{-1}} [S]$ *and* G *is terminally labelled.*

TREE Language

Let ***TREE*** $= (\mathcal{L}, \mathcal{N}, S, \mathcal{R})$ where:

1. $\mathcal{L} = (\{\square\}, \{\square\})$ where $\square$ denotes the empty label;
2. $\mathcal{N} = (\emptyset, \emptyset)$;
3. S be the graph with a single node labelled with $\square$;
4. $\mathcal{R} = \{r\}$.

r:

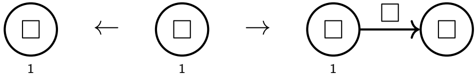


TREE Language

Let $\textbf{\textit{TREE}} = (\mathcal{L}, \mathcal{N}, S, \mathcal{R})$ where:

1. $\mathcal{L} = (\{\Box\}, \{\Box\})$ where $\Box$ denotes the empty label;
2. $\mathcal{N} = (\emptyset, \emptyset)$;
3. S be the graph with a single node labelled with $\Box$;
4. $\mathcal{R} = \{r\}$.

r:



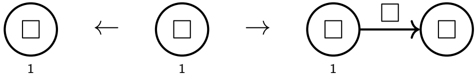
To see that this grammar generates the set of all trees, we must show that every graph in the language is a tree, and then that every tree is in the language. This is easy to see by induction.

TREE Language

Let $\textit{TREE} = (\mathcal{L}, \mathcal{N}, S, \mathcal{R})$ where:

1. $\mathcal{L} = (\{\square\}, \{\square\})$ where $\square$ denotes the empty label;
2. $\mathcal{N} = (\emptyset, \emptyset)$;
3. S be the graph with a single node labelled with $\square$;
4. $\mathcal{R} = \{r\}$.

r:



To see that this grammar generates the set of all trees, we must show that every graph in the language is a tree, and then that every tree is in the language. This is easy to see by induction.

One can see (via critical pair analysis) that $\textit{TREE}^{-1} = (\mathcal{L}, \{r^{-1}\})$ is confluent too... QUEUE WAFFLE

Time Complexity

Given a GT system, how long does it take to compute a normal form of an input graph?

Time Complexity

Given a GT system, how long does it take to compute a normal form of an input graph?

Definition 23 (Graph Matching Problem (GMP))

Given a graph G and a rule $r = \langle L \leftarrow K \rightarrow R \rangle$, find the set of injective graph morphisms $L \rightarrow G$.

Time Complexity

Given a GT system, how long does it take to compute a normal form of an input graph?

Definition 23 (Graph Matching Problem (GMP))

Given a graph G and a rule $r = \langle L \leftarrow K \rightarrow R \rangle$, find the set of injective graph morphisms $L \rightarrow G$.

Definition 24 (Rule Application Problem (RAP))

Given a graph G, a rule $r = \langle L \leftarrow K \rightarrow R \rangle$, and an injective match $g : L \rightarrow G$, find the result graph H. That is, does it satisfy the "dangling condition", and if it does, construct H.

Time Complexity

Given a GT system, how long does it take to compute a normal form of an input graph?

Definition 23 (Graph Matching Problem (GMP))

Given a graph G and a rule $r = \langle L \leftarrow K \rightarrow R \rangle$, find the set of injective graph morphisms $L \rightarrow G$.

Definition 24 (Rule Application Problem (RAP))

Given a graph G, a rule $r = \langle L \leftarrow K \rightarrow R \rangle$, and an injective match $g : L \rightarrow G$, find the result graph H. That is, does it satisfy the "dangling condition", and if it does, construct H.

Lemma 25

The GMP requires $O(|G|^{|L|})$ time. The RAP requires $O(|r|)$ time.

Root Nodes I

What if we were to limit our search area of the host graph?

Root Nodes I

What if we were to limit our search area of the host graph?

Introduce “root” nodes into the rules, and match them in the host graph. This idea was first proposed by Dörr (1995) and was implemented by Bak and Plump in 2015 by “pointing” a graph with a set of root nodes.

Root Nodes I

What if we were to limit our search area of the host graph?

Introduce “root” nodes into the rules, and match them in the host graph. This idea was first proposed by Dörr (1995) and was implemented by Bak and Plump in 2015 by “pointing” a graph with a set of root nodes.

That is, if G is a (partially) labelled graph, then (G, P_G) is a rooted (partially) labelled graph, where $P_G \subseteq V_G$.

Root Nodes I

What if we were to limit our search area of the host graph?

Introduce "root" nodes into the rules, and match them in the host graph. This idea was first proposed by Dörr (1995) and was implemented by Bak and Plump in 2015 by "pointing" a graph with a set of root nodes.

That is, if G is a (partially) labelled graph, then (G, P_G) is a rooted (partially) labelled graph, where $P_G \subseteq V_G$.

Morphisms between these pointed structures are then required to be "rootedness preserving". That is, if $g : G \to H$, then $P_G \subseteq g_V^{-1}(P_H)$.

Root Nodes I

What if we were to limit our search area of the host graph?

Introduce "root" nodes into the rules, and match them in the host graph. This idea was first proposed by Dörr (1995) and was implemented by Bak and Plump in 2015 by "pointing" a graph with a set of root nodes.

That is, if G is a (partially) labelled graph, then (G, P_G) is a rooted (partially) labelled graph, where $P_G \subseteq V_G$.

Morphisms between these pointed structures are then required to be "rootedness preserving". That is, if $g : G \to H$, then $P_G \subseteq g_V^{-1}(P_H)$.

BUT... what goes wrong...

Root Nodes II

We can no longer use natural DPOs as our definition of rule application!
Worse still, derivations are no longer reversible!

Root Nodes II

We can no longer use natural DPOs as our definition of rule application!
Worse still, derivations are no longer reversible!

A way out?

Root Nodes II

We can no longer use natural DPOs as our definition of rule application! Worse still, derivations are no longer reversible!

A way out?

What if we have a rootedness function, which can decide if a node is "unrooted", "rooted", or has "undefined rootedness".

Root Nodes II

We can no longer use natural DPOs as our definition of rule application! Worse still, derivations are no longer reversible!

A way out?

What if we have a rootedness function, which can decide if a node is "unrooted", "rooted", or has "undefined rootedness".

Our graph morphisms would then need to more strongly preserve rootedness of nodes. That is, and unrooted node can no longer be mapped to a rooted node. Only a node of undefined rootedness can be changed... Rather similar to the trick with partial labelling.

Rooted Graphs

Definition 26

A **graph** over $\mathcal{L}$ is a tuple $G = (V, E, s, t, l, m, p)$ where:

1. V is a **finite** set of **vertices**;
2. E is a **finite** set of **edges**;
3. $s : E \to V$ is a **total** source function;
4. $t : E \to V$ is a **total** target function;
5. $l : V \to \mathcal{L}_V$ is a **partial** function, labelling the vertices;
6. $m : E \to \mathcal{L}_E$ is a **total** function, labelling the edges;
7. $p : V \to \mathbb{Z}_2$ is a **partial** function, determining vertex rootedness.

Rooted Graphs

Definition 26

A **graph** over $\mathcal{L}$ is a tuple $G = (V, E, s, t, l, m, p)$ where:

1. V is a **finite** set of **vertices**;
2. E is a **finite** set of **edges**;
3. $s : E \to V$ is a **total** source function;
4. $t : E \to V$ is a **total** target function;
5. $l : V \to \mathcal{L}_V$ is a **partial** function, labelling the vertices;
6. $m : E \to \mathcal{L}_E$ is a **total** function, labelling the edges;
7. $p : V \to \mathbb{Z}_2$ is a **partial** function, determining vertex rootedness.

Definition 27

A graph G is **totally labelled** iff l_G is total, and **totally rooted** if p_G is total. If G is both, then we call it a **TLRG**.

Rooted Morphisms

Definition 28

A **graph morphism** between graphs G and H is a pair of functions $g = (g_V : V_G \to V_H, g_E : E_G \to E_H)$ such that sources, targets, labels, and rootedness are preserved. That is:

1. $\forall e \in E_G,\ g_V(s_G(e)) = s_H(g_E(e))$;
2. $\forall e \in E_G,\ g_V(t_G(e)) = t_H(g_E(e))$;
3. $\forall e \in E_G\ m_G(e) = m_H(g_E(e))$;
4. $\forall v \in l_G^{-1}(\mathcal{L}_V),\ l_G(v) = l_H(g_V(v))$;
5. $\forall v \in p_G^{-1}(\mathbb{Z}_2),\ p_G(v) = p_H(g_V(v))$.

Rooted Morphisms

Definition 28

A **graph morphism** between graphs G and H is a pair of functions $g = (g_V : V_G \to V_H, g_E : E_G \to E_H)$ such that sources, targets, labels, and rootedness are preserved. That is:

1. $\forall e \in E_G,\ g_V(s_G(e)) = s_H(g_E(e))$;
2. $\forall e \in E_G,\ g_V(t_G(e)) = t_H(g_E(e))$;
3. $\forall e \in E_G\ m_G(e) = m_H(g_E(e))$;
4. $\forall v \in l_G^{-1}(\mathcal{L}_V),\ l_G(v) = l_H(g_V(v))$;
5. $\forall v \in p_G^{-1}(\mathbb{Z}_2),\ p_G(v) = p_H(g_V(v))$.

All of the other theory we've seen for the standard case also holds... transformation occurs on the TLRGs with K partially labelled and partially rooted.

Complexity Theorems I

Definition 29

We call a rule $r = \langle L \leftarrow K \rightarrow R \rangle$ **fast** iff every **connected component** of L contains a root node.

Complexity Theorems I

Definition 29

We call a rule $r = \langle L \leftarrow K \rightarrow R \rangle$ **fast** iff every **connected component** of L contains a root node.

Theorem 30 (Fast Derivations)

Given a **TLRG** *G of* **bounded degree** *containing a* **bounded** *number of root nodes, and a GT system $T = (\mathcal{L}, \mathcal{R})$ where each rule is* **fast**, *then one can decide in* **constant time** *the* **direct successors** *of $[G]$.*

Complexity Theorems I

Definition 29

We call a rule $r = \langle L \leftarrow K \rightarrow R \rangle$ **fast** iff every **connected component** of L contains a root node.

Theorem 30 (Fast Derivations)

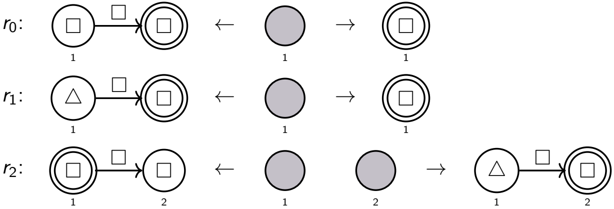
Given a **TLRG** G *of* **bounded degree** *containing a* **bounded** *number of root nodes, and a GT system* $T = (\mathcal{L}, \mathcal{R})$ *where each rule is* **fast**, *then one can decide in* **constant time** *the* **direct successors** *of* $[G]$.

Corollary 31

If each rule is additionally **root non-increasing** *and* **degree non-increasing**, *and* T **terminating** *with maximum derivation length* $N \in \mathbb{N}$, *then one can find a* **normal form** *of* $[G]$ *in* $O(N)$ *time.*

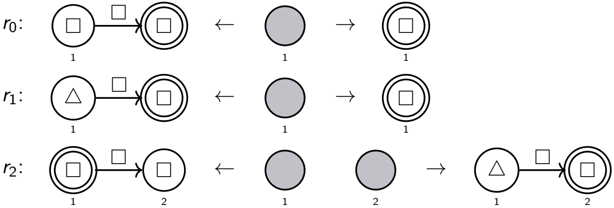
Recognising Trees I

Let $\mathcal{L} = (\{\square, \triangle\}, \{\square\})$, and $\mathcal{R} = \{r_0, r_1, r_2\}$.



Recognising Trees I

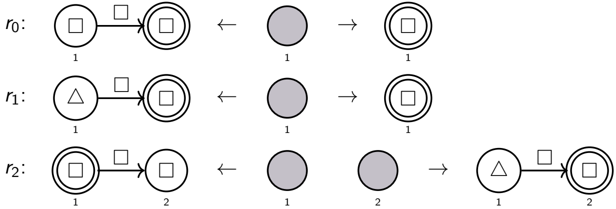
Let $\mathcal{L} = (\{\square, \triangle\}, \{\square\})$, and $\mathcal{R} = \{r_0, r_1, r_2\}$.



Intuitively, this works by pushing the “root” to the bottom of a branch, and then pruning. If we start with a tree and run this until we cannot do it anymore, we must be left with a single node.

Recognising Trees I

Let $\mathcal{L} = (\{\square, \triangle\}, \{\square\})$, and $\mathcal{R} = \{r_0, r_1, r_2\}$.



Intuitively, this works by pushing the “root” to the bottom of a branch, and then pruning. If we start with a tree and run this until we cannot do it anymore, we must be left with a single node.

The triangle labels are necessary so that, in the case that the input graph is not a tree, we could “get stuck” in a directed cycle.

Recognising Trees II

Definition 32

Given a graph G, we define $G^{\ominus}$ to be exactly G, but with every node unrooted, and everything labelled by $\square$. That is, $G^{\ominus} = (V_G, E_G, s_G, t_G, V_G \times \{\square\}, E_G \times \{\square\}, V_G \times \{0\})$.

Recognising Trees II

Definition 32

Given a graph G, we define $G^{\ominus}$ to be exactly G, but with every node unrooted, and everything labelled by $\square$. That is, $G^{\ominus} = (V_G, E_G, s_G, t_G, V_G \times \{\square\}, E_G \times \{\square\}, V_G \times \{0\})$.

Definition 33

By "input graph", we mean any TLRG containing exactly one "root" node, with edges and vertices all labelled by $\square$.

Recognising Trees II

Definition 32

Given a graph G, we define $G^{\ominus}$ to be exactly G, but with every node unrooted, and everything labelled by $\square$. That is, $G^{\ominus} = (V_G, E_G, s_G, t_G, V_G \times \{\square\}, E_G \times \{\square\}, V_G \times \{0\})$.

Definition 33

By "input graph", we mean any TLRG containing exactly one "root" node, with edges and vertices all labelled by $\square$.

Theorem 34 (Tree Recognition)

Given an input graph G, one may use the system $(\mathcal{L}, \mathcal{R})$ from G to find a normal form for G, say H. H is the single root-node graph labelled by $\square$ iff $[G^{\ominus}] \in \boldsymbol{L}(TREE)$. Moreover, for input graphs of bounded degree, we terminate in linear time (with respect to $|V_G|$).

Motivation

Notice in the rooted tree example that every pair $H_1 \Leftarrow G \Rightarrow H_2$ where G was a tree can be joined, but this is not necessarily true of any G in general, so we don't have local confluence.

Motivation

Notice in the rooted tree example that every pair $H_1 \Leftarrow G \Rightarrow H_2$ where G was a tree can be joined, but this is not necessarily true of any G in general, so we don't have local confluence.

We only need to have joinability of pairs with non-garbage start graphs...

Motivation

Notice in the rooted tree example that every pair $H_1 \Leftarrow G \Rightarrow H_2$ where G was a tree can be joined, but this is not necessarily true of any G in general, so we don't have local confluence.

We only need to have joinability of pairs with non-garbage start graphs...

Definition 35

Let $T = (\mathcal{L}, \mathcal{R})$ be a GT system, and $D \subseteq \mathcal{G}(\mathcal{L})$ be a set of abstract graphs. Then, a graph G is called **garbage** iff $[G] \notin D$.

Motivation

Notice in the rooted tree example that every pair $H_1 \Leftarrow G \Rightarrow H_2$ where G was a tree can be joined, but this is not necessarily true of any G in general, so we don't have local confluence.

We only need to have joinability of pairs with non-garbage start graphs...

Definition 35

Let $T = (\mathcal{L}, \mathcal{R})$ be a GT system, and $D \subseteq \mathcal{G}(\mathcal{L})$ be a set of abstract graphs. Then, a graph G is called **garbage** iff $[G] \notin D$.

Definition 36

Let $T = (\mathcal{L}, \mathcal{R})$, and $D \subseteq \mathcal{G}(\mathcal{L})$. T is **weakly garbage separating** w.r.t. D iff for all G, H such that $G \Rightarrow_{\mathcal{R}} H$, if $[G] \in D$ then $[H] \in D$. T is **garbage separating** iff we have $[G] \in D$ iff $[H] \in D$.

Confluence Modulo Garbage

Definition 37

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. If for all graphs G, H_1, H_2, such that $[G] \in D$, if $H_1 \Leftarrow_{\mathcal{R}}^* G \Rightarrow_{\mathcal{R}}^* H_2$ ($H_1 \Leftarrow_{\mathcal{R}} G \Rightarrow_{\mathcal{R}} H_2$) implies that H_1, H_2 are **joinable**, then T is (**locally**) **confluence modulo garbage** w.r.t. D.

Confluence Modulo Garbage

Definition 37

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. If for all graphs G, H_1, H_2, such that $[G] \in D$, if $H_1 \Leftarrow^*_{\mathcal{R}} G \Rightarrow^*_{\mathcal{R}} H_2$ ($H_1 \Leftarrow_{\mathcal{R}} G \Rightarrow_{\mathcal{R}} H_2$) implies that H_1, H_2 are **joinable**, then T is (**locally**) **confluence modulo garbage** w.r.t. D.

Definition 38

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. If there is no infinite derivation sequence $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} G_2 \Rightarrow_{\mathcal{R}} \cdots$ such that $[G_0] \in D$, then T is **terminating modulo garbage** w.r.t. D.

Confluence Modulo Garbage

Definition 37

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. If for all graphs G, H_1, H_2, such that $[G] \in D$, if $H_1 \Leftarrow_{\mathcal{R}}^* G \Rightarrow_{\mathcal{R}}^* H_2$ ($H_1 \Leftarrow_{\mathcal{R}} G \Rightarrow_{\mathcal{R}} H_2$) implies that H_1, H_2 are **joinable**, then T is (**locally**) **confluence modulo garbage** w.r.t. D.

Definition 38

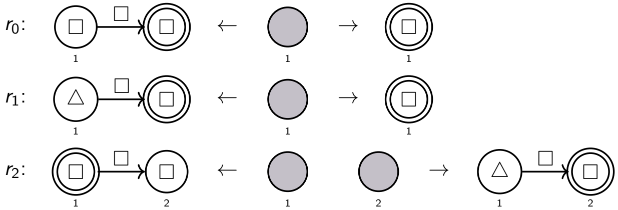
Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. If there is no infinite derivation sequence $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} G_2 \Rightarrow_{\mathcal{R}} \cdots$ such that $[G_0] \in D$, then T is **terminating modulo garbage** w.r.t. D.

Theorem 39 (Newman-Garbage Lemma)

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. If T is **terminating modulo garbage** *and* **weakly garbage separating**, *then it is* **confluent modulo garbage** *iff it is* **locally confluent modulo garbage**.

Tree Recognition Revisited

Let $\mathcal{L} = (\{\square, \triangle\}, \{\square\})$, $\mathcal{R} = \{r_0, r_1, r_2\}$.



Lemma 40

The GT system $T = (\mathcal{L}, \mathcal{R})$ *is* **garbage separating** *w.r.t. to* $D = \{[G] \in \mathcal{G}^{\oplus}(\mathcal{L}) \mid [G^{\ominus}] \in \mathbf{L}(\textbf{\textit{TREE}}), |p_G^{-1}(\{1\})| = 1\}$ *and* **confluent modulo garbage** *w.r.t.* $E = \{[G] \in D \mid l_G(V_G) = \{\square\}\}$.

Showing Confluence

It is well known that for totally labelled systems (that is, the interface graph K is totally labelled), that it is sufficient (but not necessary) to check “strong joinability” of “critical pairs”.

Showing Confluence

It is well known that for totally labelled systems (that is, the interface graph K is totally labelled), that it is sufficient (but not necessary) to check "strong joinability" of "critical pairs".

It turns out it's enough to check strong joinability of only the critical pairs which have start graph in the subgraph closure of D.

Showing Confluence

It is well known that for totally labelled systems (that is, the interface graph K is totally labelled), that it is sufficient (but not necessary) to check "strong joinability" of "critical pairs".

It turns out it's enough to check strong joinability of only the critical pairs which have start graph in the subgraph closure of D.

Theorem 41 (Non-Garbage Critical Pair Lemma)

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. *If all its* **non-garbage critical pairs** *are* **strongly joinable**, *then* T *is* **locally confluent mod garbage** *w.r.t.* D.

Showing Confluence

It is well known that for totally labelled systems (that is, the interface graph K is totally labelled), that it is sufficient (but not necessary) to check "strong joinability" of "critical pairs".

It turns out it's enough to check strong joinability of only the critical pairs which have start graph in the subgraph closure of D.

Theorem 41 (Non-Garbage Critical Pair Lemma)

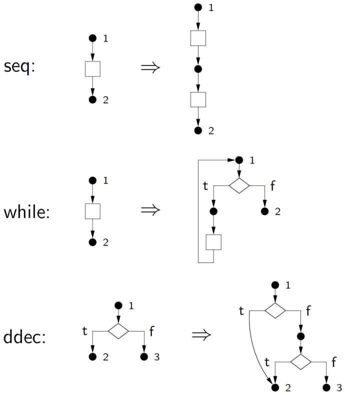
Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. *If all its* **non-garbage critical pairs** *are* **strongly joinable**, *then* T *is* **locally confluent mod garbage** *w.r.t.* D.

Corollary 42

Let $T = (\mathcal{L}, \mathcal{R})$, $D \subseteq \mathcal{G}(\mathcal{L})$. *If* T *is* **terminating modulo garbage**, **weakly garbage separating**, *and all its* **non-garbage critical pairs** *are* **strongly joinable** *then* T *is* **confluent modulo garbage**.

Extended Flow Diagrams I

The language of **extended flow diagrams** is generated by $\textbf{\textit{EFD}} = (\mathcal{L}, \mathcal{N}, \mathcal{R}, S)$ where $\mathcal{L}_V = \{\bullet, \square, \diamond\}$, $\mathcal{L}_E = \{t, f, \square\}$, $\mathcal{N}_V = \mathcal{N}_E = \emptyset$, $\mathcal{R} = \{seq, while, ddec, dec1, dec2\}$, and $S = \bullet\!\!\rightarrow\!\square\!\!\rightarrow\!\!\bullet$.



Extended Flow Diagrams II

Lemma 43

$EFD^{-1} = (\mathcal{L}, \mathcal{R}^{-1})$ is **terminating**. *Moreover, it is* **garbage separating** *w.r.t.* $\boldsymbol{L}(\boldsymbol{EFD})$.

Lemma 44

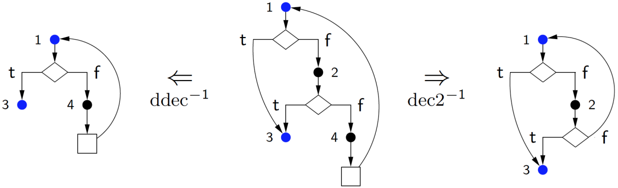
Every **directed cycle** *in a graph in the* **subgraph closure** *of* $\boldsymbol{L}(\boldsymbol{EFD})$ *contains a t-labelled edge.*

Theorem 45 (EFD Recognition)

$EFD^{-1} = (\mathcal{L}, \mathcal{R}^{-1})$ is **confluent modulo garbage** *w.r.t.* $\boldsymbol{L}(\boldsymbol{EFD})$, *but not* **confluent**.

Extended Flow Diagrams III

By Lemma 43 and the Newman-Garbage Lemma, it suffices to show local confluent modulo garbage. Consider the critical pairs of the system. It turns out there are ten critical pairs, all but one of which are strongly joinable.



Thus, we do not have confluence, however by Lemma 44, the non-joinable critical pair is garbage, so by the Non-Garbage Critical Pair Lemma, we have local confluence modulo garbage, as required.