# Decidability Results for Graph Transformation Systems (with a Tutorial on Graph Transformation)

Barbara König

Universität Duisburg-Essen, Germany

Joint work with Jan Stückrath, Giorgio Delzanno, Arnaud Sangnier, Nathalie Bertrand

# Overview

# Overview

**Verification of dynamic systems using graph transformation**

1. Model the system by a graph transformation system
2. Use techniques for verifying graph transformation systems in order to verify the system

# Overview

## Verification of dynamic systems using graph transformation

1. Model the system by a graph transformation system
2. Use techniques for verifying graph transformation systems in order to verify the system

## Features of graph transformation

- Infinite state space
- Dynamic creation and deletion of objects
- Mobility
- Variable topology
- ...

These features are good for modelling, but problematic when it comes to verification!

# Graph Transformation Systems

Graph Transformation Systems (GTSs) as a computational model for dynamic systems.

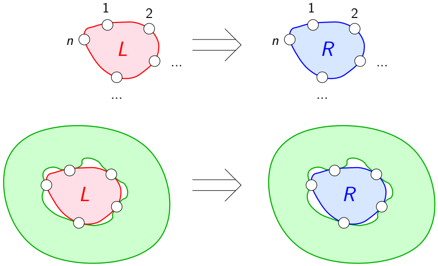
- the graph represents the state
- production applications represent state changes

# Graph Transformation Systems

Graph Transformation Systems (GTSs) as a computational model for dynamic systems.

- the graph represents the state
- production applications represent state changes

A graph transformation system (GTS) consists of an initial (hyper-)graph and a set of rules:



# Example: Leader Election

Example: leader election protocol on a ring (Chang/Roberts)

# Example: Leader Election

Example: leader election protocol on a ring (Chang/Roberts)

## Leader election protocol

- The leader should be the process with the smallest Id.
- Every process generates a message with its own Id and sends it to its successor.
- Upon reception of a message with content *MId* a process with Id *PId* acts as follows:
  - if $MId < PId$ forward the message to the next successor
  - if $MId = PId$ the process declares itself the leader
  - If $MId > PId$ do not pass on the message

# Example: Leader Election

Example: leader election protocol on a ring (Chang/Roberts)

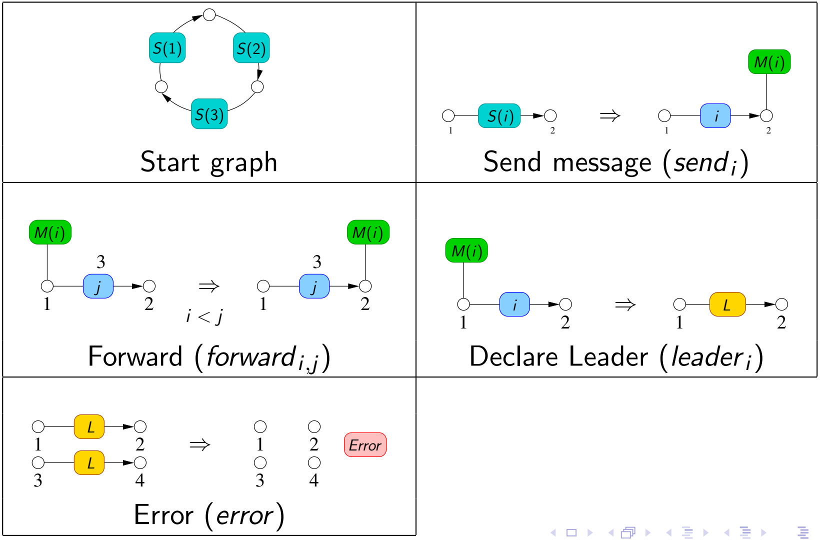
**Leader election protocol**

- The leader should be the process with the smallest Id.
- Every process generates a message with its own Id and sends it to its successor.
- Upon reception of a message with content $MId$ a process with Id $PId$ acts as follows:
  - if $MId < PId$ forward the message to the next successor
  - if $MId = PId$ the process declares itself the leader
  - If $MId > PId$ do not pass on the message

Properties to verify: there will never be two leaders, a leader will be elected eventually, . . .

# Example: Leader Election

Rules for the finite-state case (with three processes):



# Hypergraphs and Graph Morphisms

I will in the following work with hypergraphs, since I find them more suitable for system modelling. People have different preferences ... The results on verification are mostly independent of the choice of graph model.

## Hypergraph

Let $\Lambda$ be a set of labels. A hypergraph or graph $G$ is a tuple $G = (V, E, c, l)$, where

- $V$ is a set of nodes,
- $E$ is a set of (hyper-)edges,
- $c\colon E \to V^*$ is the connection function *and*
- $l\colon E \to \Lambda$ is the labelling function.

# Hypergraphs and Graph Morphisms

Graph morphisms are structure-preserving maps between graphs.

## Graph morphismus

Let $G_1 = (V_1, E_1, c_1, l_1)$, $G_2 = (V_2, E_2, c_2, l_2)$ be two graphs. A graph morphism $\varphi\colon G_1 \to G_2$ is a pair of mappings $\varphi_V\colon V_1 \to V_2$, $\varphi_E\colon E_1 \to E_2$ such that for all $e_1 \in E_1$ it holds that

- $c_2(\varphi_E(e_1)) = \varphi_V(c_1(e_1))$ *and*
- $l_2(\varphi_E(e_1)) = l_1(e_1)$.

# Hypergraphs and Graph Morphisms

Graph morphisms are structure-preserving maps between graphs.

## Graph morphismus

Let $G_1 = (V_1, E_1, c_1, l_1)$, $G_2 = (V_2, E_2, c_2, l_2)$ be two graphs. A graph morphism $\varphi\colon G_1 \to G_2$ is a pair of mappings $\varphi_V\colon V_1 \to V_2$, $\varphi_E\colon E_1 \to E_2$ such that for all $e_1 \in E_1$ it holds that

- $c_2(\varphi_E(e_1)) = \varphi_V(c_1(e_1))$ *and*
- $l_2(\varphi_E(e_1)) = l_1(e_1)$.

Analogously, a partial graph morphism consists of two partial mappings $\varphi_V, \varphi_E$. If $\varphi_E(e)$ is defined, $\varphi_V$ must be defined on all nodes attached to $e$.

Two graphs $G_1, G_2$ are isomorphic if there is a bijective total morphism between them (symbolically $G_1 \cong G_2$).

# Graph Transformation

## Graph transformation rule (definition)

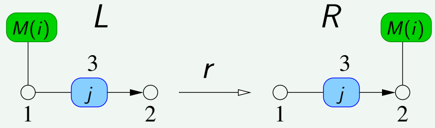
A (graph transformation) rule consists of two graphs $L, R$ and a partial graph morphisms $r\colon L \to R$.

# Graph Transformation

**Graph transformation rule (definition)**

A (graph transformation) rule consists of two graphs $L, R$ and a partial graph morphisms $r\colon L \to R$.

Example:




A rule $r$ specifies what is deleted (items of the left-hand side for which $r$ is undefined), what is preserved (items of the left-hand side for which $r$ is defined) and what is created (parts of the right-hand side not in the image of $r$).

# Graph Transformation

Graph transformation can be described by a gluing diagram, involving the match $m$ and the rule $r$.

## Graph transformation (single-pushout approach – SPO)

Let $r\colon L \to R$ be a rule. We say that a graph $G$ is transformed into a graph $H$ (symbolically: $G \overset{r}{\Rightarrow} H$) if there is a total graph morphisms $m\colon L \to G$ (the match) and additional morphisms into $H$ such that the following diagram is a pushout (= gluing diagram of partial graph morphisms).

$$\begin{array}{ccc} L & \xrightarrow{\;r\;} & R \\ {\scriptstyle m}\downarrow & & \downarrow \\ G & \longrightarrow & H \end{array}$$

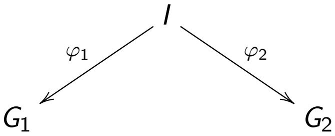
# Graph Transformation

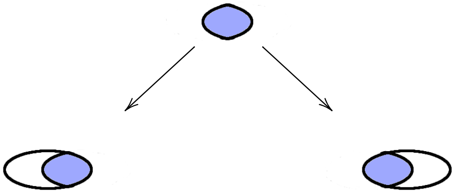
In order to understand (single-pushout) graph transformation we have to specify what it means to glue two graphs over a common subgraph.

# Graph Transformation

In order to understand (single-pushout) graph transformation we have to specify what it means to glue two graphs over a common subgraph.

Gluing of graphs for total morphisms (schematically)

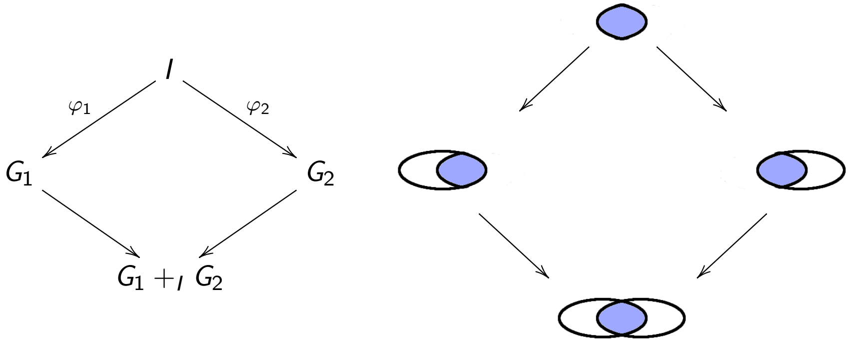


# Graph Transformation

In order to understand (single-pushout) graph transformation we have to specify what it means to glue two graphs over a common subgraph.

Gluing of graphs for total morphisms (schematically)



# Graph Transformation

## Gluing of graphs (total morphisms)

Let $I, G_1, G_2$ be graphs with total graph morphisms $\varphi_1 \colon I \to G_1$, $\varphi_2 \colon I \to G_2$. We call $I$ the interface.

Let $\equiv$ be the smallest equivalence relation on $G_1 \uplus G_2$ which satisfies $\varphi_1(x) \equiv \varphi_2(x)$ for all $x \in I$.

The gluing of $G_1, G_2$ over $I$ is defined as follows:

$$G_1 +_I G_2 = (G_1 \uplus G_2)/\equiv$$

(Take the disjoint union of $G_1, G_2$ and quotient through the equivalence $\equiv$.)

# Graph Transformation

For partial morphisms, things become slightly more complicated . . .

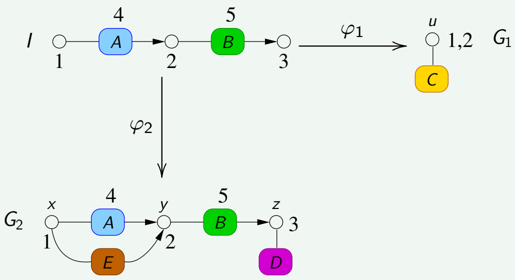
## Gluing of graphs (partial morphisms)

Let $I, G_1, G_2$ be graphs with *partial* graph morphisms $\varphi_1 \colon I \to G_1$, $\varphi_2 \colon I \to G_2$.

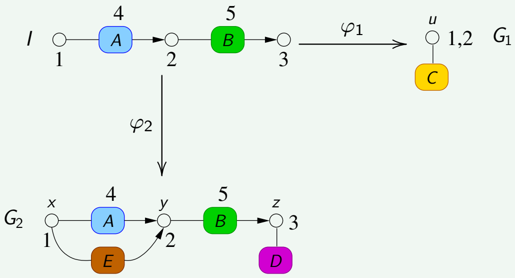
Compute equivalence classes as before, but remove those equivalence classes which contain the image $\varphi_1(x)$ of an item (node or edge) $x$ of $I$, for which $\varphi_2(x)$ is undefined (or vice versa).

In addition remove all equivalence classes which contain an edge attached to a node whose equivalence class has been removed (dangling edge).

# Graph Transformation



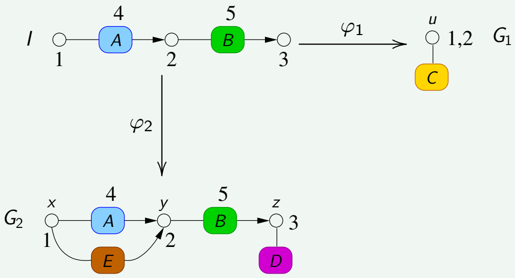
# Graph Transformation




Equivalence classes:

$\{x, y, u\}$
$\{z\}$
$\{A\}$
$\{B\}$
$\{C\}$
$\{D\}$
$\{E\}$

# Graph Transformation

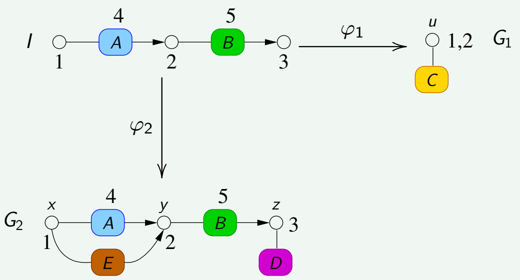



## Equivalence classes:

$\{x, y, u\}$
$\{z\}$ ← remove
$\{A\}$ ← remove
$\{B\}$ ← remove

$\{C\}$
$\{D\}$
$\{E\}$

# Graph Transformation

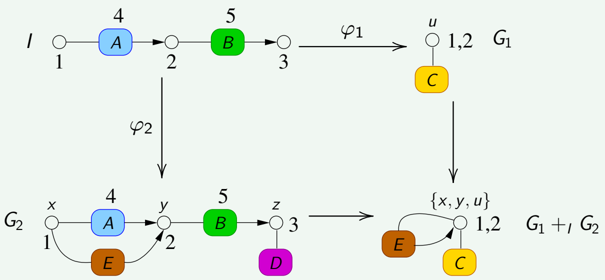



## Equivalence classes:

$\{x, y, u\}$
$\{z\}$ ← remove
$\{A\}$ ← remove
$\{B\}$ ← remove

$\{C\}$
$\{D\}$ ← remove (dangling edge)
$\{E\}$

# Graph Transformation




Equivalence classes:

$\{x, y, u\}$

$\{z\}$ ← remove

$\{A\}$ ← remove

$\{B\}$ ← remove

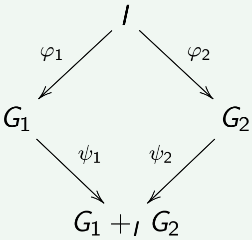
$\{C\}$

$\{D\}$ ← remove (dangling edge)

$\{E\}$

# Graph Transformation

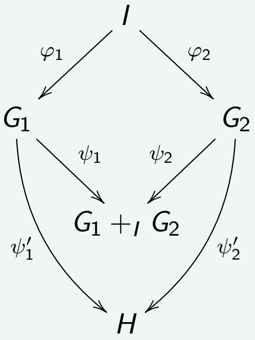
Such a "gluing diagram" has the following universal property:




The diagram commutes, i.e., $\psi_1 \circ \varphi_1 = \psi_2 \circ \varphi_2$.

# Graph Transformation

Such a "gluing diagram" has the following universal property:

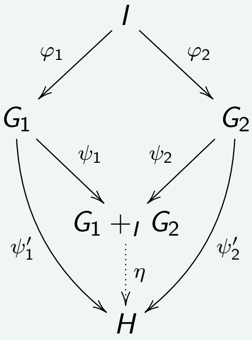



The diagram commutes, i.e., $\psi_1 \circ \varphi_1 = \psi_2 \circ \varphi_2$.

For any two morphisms $\psi_1' \colon G_1 \to H$, $\psi_2' \colon G_2 \to H$ satisfying $\psi_1' \circ \varphi_1 = \psi_2' \circ \varphi_2$

# Graph Transformation

Such a "gluing diagram" has the following universal property:

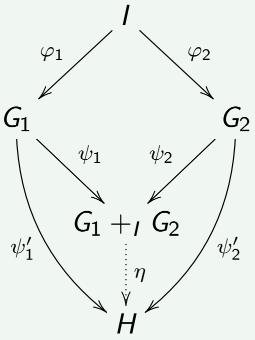



The diagram commutes, i.e., $\psi_1 \circ \varphi_1 = \psi_2 \circ \varphi_2$.

For any two morphisms $\psi_1' \colon G_1 \to H$, $\psi_2' \colon G_2 \to H$ satisfying $\psi_1' \circ \varphi_1 = \psi_2' \circ \varphi_2$ there exists a unique morphism $\eta \colon G_1 +_I G_2 \to H$ such that $\eta \circ \psi_1 = \psi_1'$ and $\eta \circ \psi_2 = \psi_2'$.

# Graph Transformation

Such a "gluing diagram" has the following universal property:



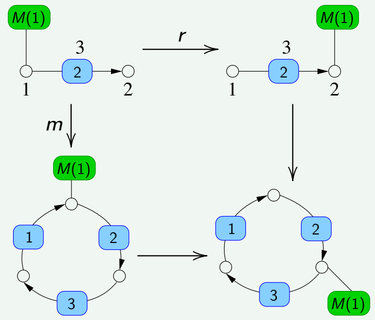

The diagram commutes, i.e., $\psi_1 \circ \varphi_1 = \psi_2 \circ \varphi_2$.

For any two morphisms $\psi_1' \colon G_1 \to H$, $\psi_2' \colon G_2 \to H$ satisfying $\psi_1' \circ \varphi_1 = \psi_2' \circ \varphi_2$ there exists a unique morphism $\eta \colon G_1 +_I G_2 \to H$ such that $\eta \circ \psi_1 = \psi_1'$ and $\eta \circ \psi_2 = \psi_2'$.

Diagrams with this property are called pushouts in category theory. The graph $G_1 +_I G_2$ is unique up to isomorphism.

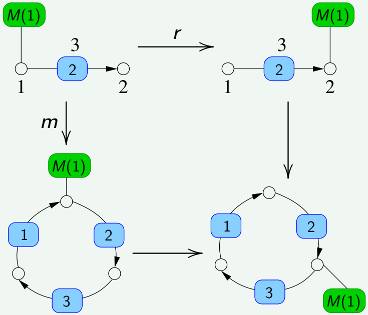
# Graph Transformation

Example diagram describing a graph transformation:



# Graph Transformation

Example diagram describing a graph transformation:




In the following: we consider only injective matches

# Graph Transformation

There are other graph transformation approaches, for instance

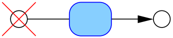
the double-pushout approach (DPO)

# Graph Transformation

There are other graph transformation approaches, for instance

the double-pushout approach (DPO)

The main practical difference is the treatment of so-called dangling edges, i.e., edges which are *not* deleted by a rule, but where at least one of the attached nodes is deleted.



- In the single-pushout approach the edge is deleted as well.
- In the double-pushout approach the corresponding rule can not be applied.

# Decidability

We are interested in the following decidability questions ...

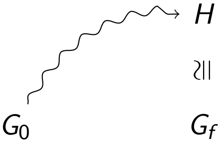
- For what kind of GTS are reachability and coverability decidable?
- How can GTS be restricted, such that these problems become decidable?
- What are the key features that cause undecidability?

Survey on existing results and our results published at RTA '12

# Reachability and Coverability

A graph transformation system (GTS) consists of a set of rules $\mathcal{R}$ and an initial graph $G_0$.

Reachability: Is there a graph $H$ such that $G_0 \Rightarrow^* H$ and $H$, $G_f$ are isomorphic?



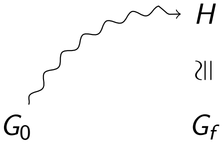

$G_f$ is reachable

# Reachability and Coverability

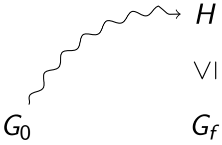
A graph transformation system (GTS) consists of a set of rules $\mathcal{R}$ and an initial graph $G_0$.

Reachability: Is there a graph $H$ such that $G_0 \Rightarrow^* H$ and $H$, $G_f$ are isomorphic?

Coverability: Is there a graph $H$ such that $G_0 \Rightarrow^* H$ and $G_f$ is isomorphic to a subgraph of $H$?




$G_f$ is reachable




$G_f$ is coverable

# General GTS

In the general case GTSs are Turing-complete:

Reachability: Undecidable

$\rightsquigarrow$ Encode a Turing Machine into a GTS and ask if a final state is reachable. Additional rules delete the tape once a final state is reached.

Coverability: Undecidable

$\rightsquigarrow$ Encode a Turing Machine into a GTS and ask if a final state is coverable.

# General GTS

In the general case GTSs are Turing-complete:

Reachability: Undecidable

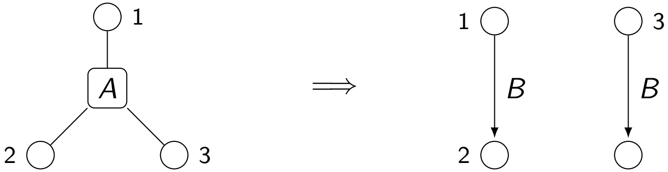
$\rightsquigarrow$ Encode a Turing Machine into a GTS and ask if a final state is reachable. Additional rules delete the tape once a final state is reached.

Coverability: Undecidable

$\rightsquigarrow$ Encode a Turing Machine into a GTS and ask if a final state is coverable.

Reachability and coverability are naturally decidable for GTSs that are finite-state (up to isomorphism).

# Context-free GTS




Left-hand sides of rules consist of a single hyperedge and no nodes are deleted.

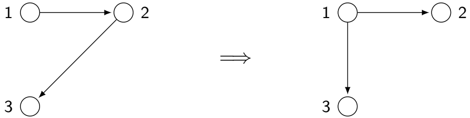
Reachability: NP-complete

$\rightsquigarrow$ The membership problem for such GTSs is NP-complete [Habel]

Coverability: In PSPACE, NP-hard

$\rightsquigarrow$ Exact complexity unknown

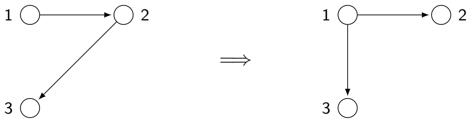
# Node Deletion and Creation




Assume the GTS neither creates nor deletes nodes.

Connections between SPO and Petri nets have been studied by Baldan, Corradini, Montanari.

# Node Deletion and Creation




Assume the GTS neither creates nor deletes nodes.

Connections between SPO and Petri nets have been studied by Baldan, Corradini, Montanari.

Reachability and Coverability are decidable for Petri nets [Mayr].

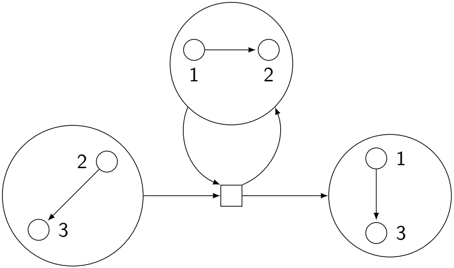
Reachability and Coverability: Decidable

$\rightsquigarrow$ These problems can be reduced to reachability and coverability on Petri nets.

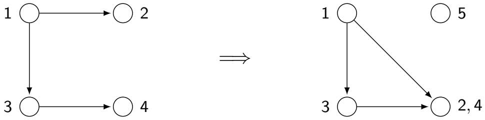
# Node Deletion and Creation

Procedure:

- Think of a "complete" graph with all nodes and an edge between each pair of nodes (loops included) for each label.
- The Petri net has one place for each edge of this graph.
- Add transitions to the net for every possible instantiation of a rule.
- The tokens count the occurrence of each edge.



# Node Deletion and Creation




The GTS may delete and merge nodes, but the number of nodes is constant.

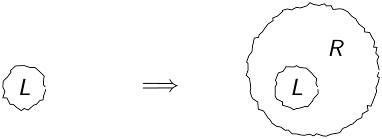
Reachability: Undecidable

$\rightsquigarrow$ Reachability of Petri nets with reset and transfer arcs can be reduced to this type of GTS.

Coverability: Decidable

$\rightsquigarrow$ This can be reduced to Petri nets with reset and transfer arcs.

# Non-deleting GTS




The GTS deletes no nodes or edges, i.e. the rule morphism is a total injection.

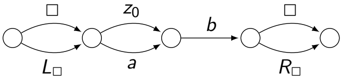
Reachability: Decidable

⇝ Due to the monotonicity of the rules

Coverability: Undecidable

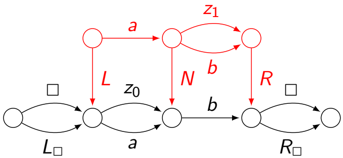
⇝ Reduce the halting problem for Turing machines to this problem

# Non-deleting GTS

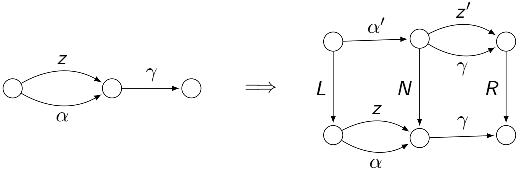



- Start computation with the initial graph.

# Non-deleting GTS

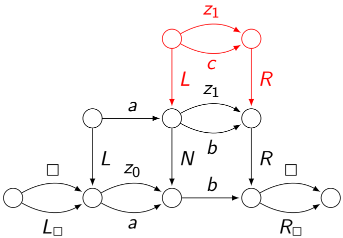



- Start computation with the initial graph.
- Each application of a $\delta$-rule increases the height of the grid.

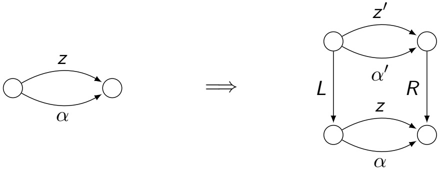



$$\delta(z, \alpha) = (z', \alpha', R)$$

# Non-deleting GTS

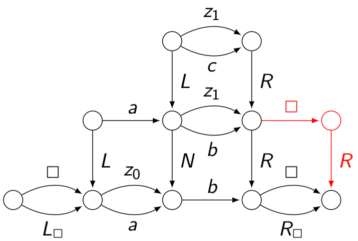



- Start computation with the initial graph.
- Each application of a $\delta$-rule increases the height of the grid.

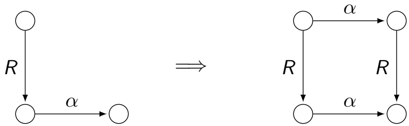



$$\delta(z, \alpha) = (z', \alpha', N)$$

# Non-deleting GTS

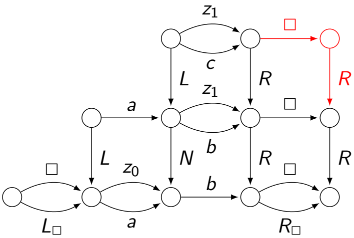



- Start computation with the initial graph.
- Each application of a $\delta$-rule increases the height of the grid.
- Auxiliary rules copy the tape from lower levels to higher levels.

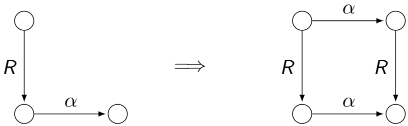



auxiliary rule

# Non-deleting GTS

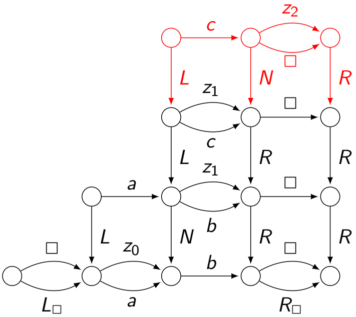



- Start computation with the initial graph.
- Each application of a $\delta$-rule increases the height of the grid.
- Auxiliary rules copy the tape from lower levels to higher levels.

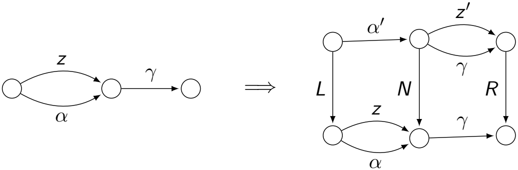



auxiliary rule

# Non-deleting GTS




- Start computation with the initial graph.
- Each application of a $\delta$-rule increases the height of the grid.
- Auxiliary rules copy the tape from lower levels to higher levels.




$$\delta(z, \alpha) = (z', \alpha, R)$$

# GTS and Well-structured Transition Systems

In the following we will present decidability results based on well-structured transition systems [Finkel, Schnoebelen] [Abdulla et al.]:

- Graphs are represented symbolically by using well-quasi orders (wqo's)
- Coverability may depend on the used order
- Rule formats may be restricted

# GTS and Well-structured Transition Systems

| order | wqo on graph class | well-structured for |
|---|---|---|
| minor ordering | all graphs | lossy systems (contraction rules) |
| subgraph ordering | bounded path length | GTS without NACs |
| induced subgr. ordering | bounded path length and edge multiplicity | GTS with restricted NACs |

NACs = negative application conditions

# GTS and Well-structured Transition Systems

| order | wqo on graph class | well-structured for |
|---|---|---|
| minor ordering | all graphs | lossy systems (contraction rules) |
| subgraph ordering | bounded path length | GTS without NACs |
| induced subgr. ordering | bounded path length and edge multiplicity | GTS with restricted NACs |

NACs = negative application conditions

---

In these cases the coverability problem is decidable if only graphs contained the graph class (e.g., graphs of bounded path length) are reachable from the start graph $G_0$.

# GTS and Well-structured Transition Systems

| order | wqo on graph class | well-structured for |
|---|---|---|
| minor ordering | all graphs | lossy systems (contraction rules) |
| subgraph ordering | bounded path length | GTS without NACs |
| induced subgr. ordering | bounded path length and edge multiplicity | GTS with restricted NACs |

NACs = negative application conditions

---

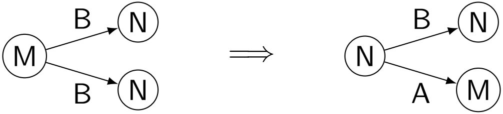
In these cases the coverability problem is decidable if only graphs contained the graph class (e.g., graphs of bounded path length) are reachable from the start graph $G_0$.

See our paper at CONCUR '14

Implementation: UNCOVER

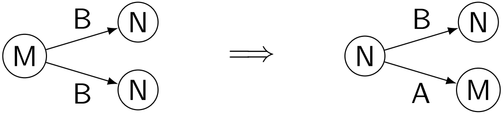
# Node and Edge Relabelling

Nodes and edges are labelled, the rules delete no nodes and edges but may modify node and edge labels.



# Node and Edge Relabelling

Nodes and edges are labelled, the rules delete no nodes and edges but may modify node and edge labels.




Reachability and coverability are clearly decidable (the set of states is finite), but there is another interesting problem:

**Definition (Existential Coverability Problem)**

Is there an initial graph $G_0$, labeled only with initial labels, such that we can reach a graph $G'$ that covers a given graph $G$?

# Node or Edge Relabelling

If only nodes have labels:

Existential Coverability: Decidable

$\rightsquigarrow$ We can use a simple fixed point computation.

If only edges have labels.

Existential Coverability: Decidable

$\rightsquigarrow$ Same as node relabelling case.

# Node and Edge Relabelling

Assume nodes and edges have labels.

Existential Coverability: Undecidable

$\leadsto$ We can encode a Turing machine (TM) into this setting.

# Node and Edge Relabelling

Assume nodes and edges have labels.

Existential Coverability: Undecidable

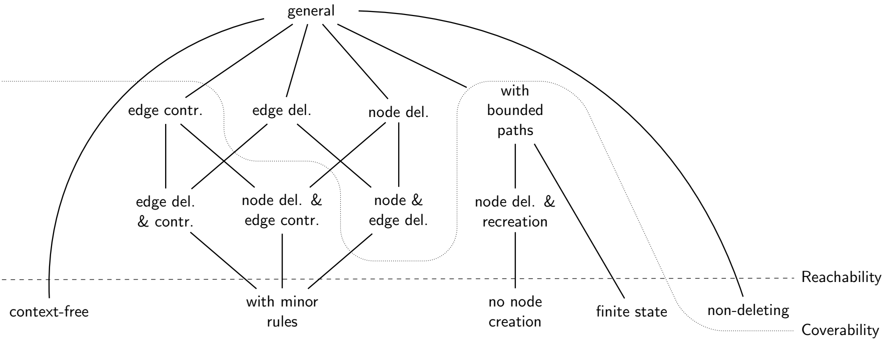
$\rightsquigarrow$ We can encode a Turing machine (TM) into this setting.

Procedure:

- Extract tapes out of the initial graph.
  - $\rightsquigarrow$ Node and edge labels are needed!
- Rules of the TM can be translated directly.
- If the TM halts, then the initial graph was of sufficient size to simulate the TM.

# Conclusion

Reachability is not strictly more difficult than coverability:



# Future Work

## Future Work

- Are there other interesting restricted GTSs?
- Why is reachability sometimes decidable if coverability is not and vice versa?
- Which properties of GTSs cause undecidability?
- Decidability results for the double pushout approach?
- Can other types of GTSs be modelled as WSTS?
- What is the exact complexity of coverability for context-free GTS?