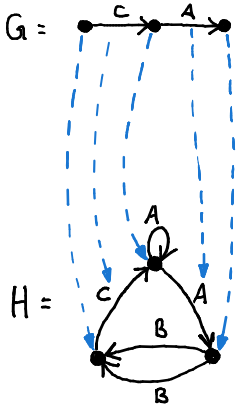
Counterexample-Guided Abstraction Refinement for Generalized Graph Transformation Systems

Lara Stoltenow

Graphs and graph morphisms



Morphism
$m : G \to H$
(injective)

Morphism
$m' : G \to H$
(not injective)

This talk: directed graphs with labeled multi edges
(works for other types of graphs too, e.g. hyperedges, node labels, ...)
& only injective morphisms

Graph conditions

$\forall f.A$, $\exists f.A$ for a graph morphism f and child condition A

$A \wedge B$, $A \vee B$, $\neg A$, true, false

(basically first-order logic)

"somewhere in the graph there is ...." $\exists \emptyset \to [\bullet \xrightarrow{A} \bullet]$. true

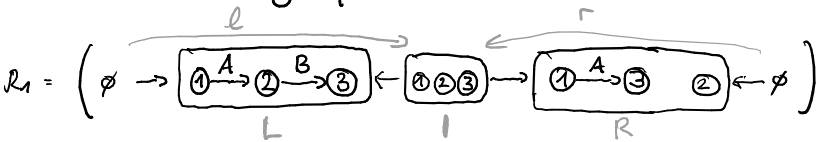
" whenever there is ... , then there also is ... around it" $\forall \emptyset \to [1 \xrightarrow{A} 2]$. $\left(\exists [1 \xrightarrow{A} 2] \to [1 \xrightarrow{A} 2 \xrightarrow{B} 3]. \text{true} \;\vee\; \exists [1 \xrightarrow{A} 2] \to [0 \xrightarrow{B} 1 \xrightarrow{A} 2]. \text{true}\right)$

"there are no antiparallel edges" $\forall \emptyset \to [1 \underset{A}{\overset{A}{\rightleftarrows}} 2]$. false

(negation: as in FOL: $\neg\exists \ldots \equiv \forall\neg \ldots$)

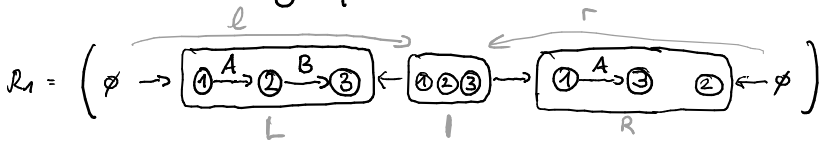
Graph Transformation Systems

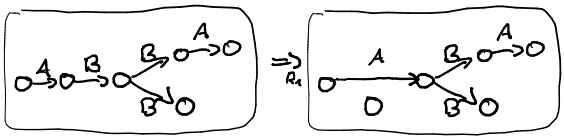
Rules allow replacing subgraph occurrences with another graph:



Graph Transformation Systems

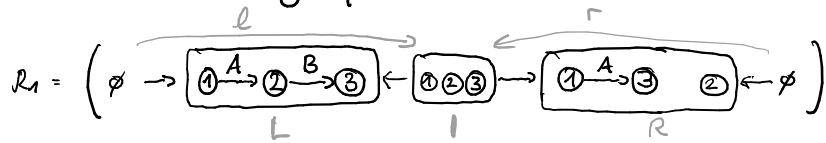
Rules allow replacing subgraph occurrences with another graph:

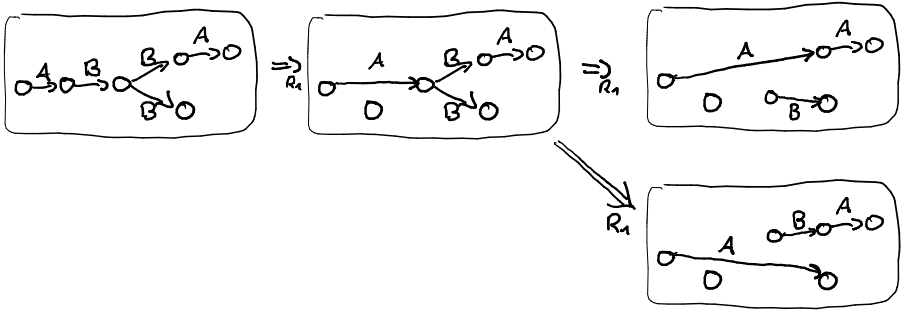




Graph Transformation Systems

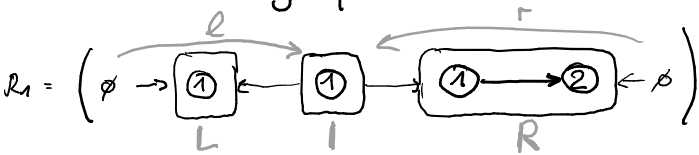
Rules allow replacing subgraph occurrences with another graph:





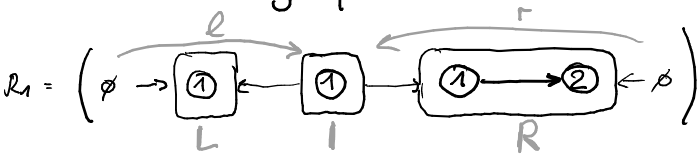
Graph Transformation Systems

Rules allow replacing subgraph occurrences with another graph:



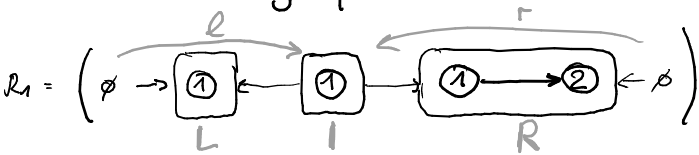
Graph Transformation Systems

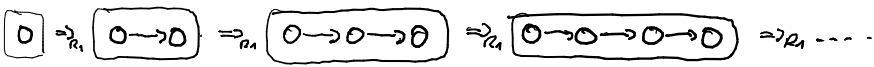
Rules allow replacing subgraph occurrences with another graph:



Graph Transformation Systems

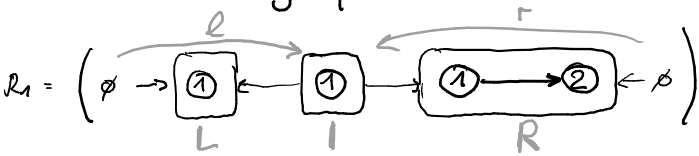
Rules allow replacing subgraph occurrences with another graph:

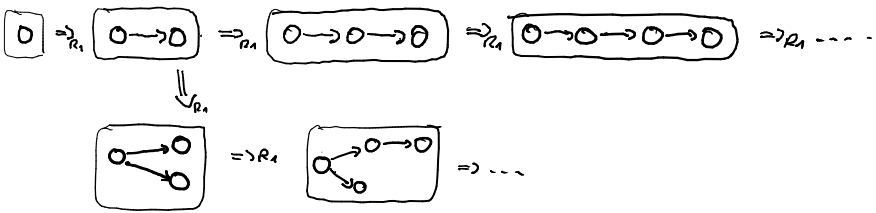




Graph Transformation Systems

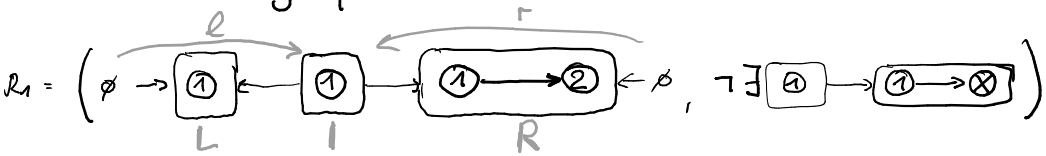
Rules allow replacing subgraph occurrences with another graph:

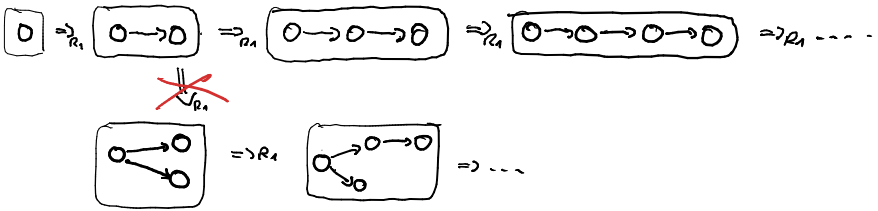




Graph Transformation Systems

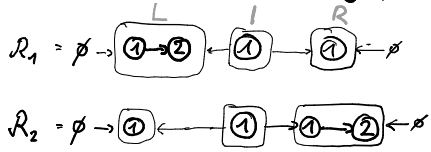
Rules allow replacing subgraph occurrences with another graph:





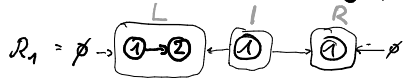
Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

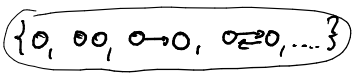


Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

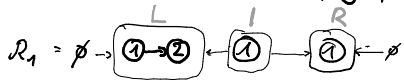


Concrete transition system:

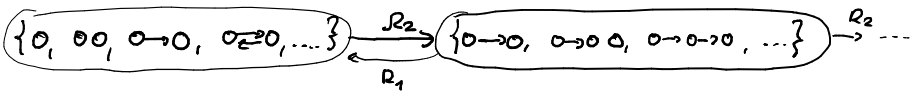


Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

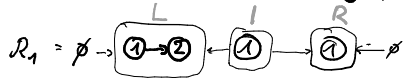


Concrete transition system:

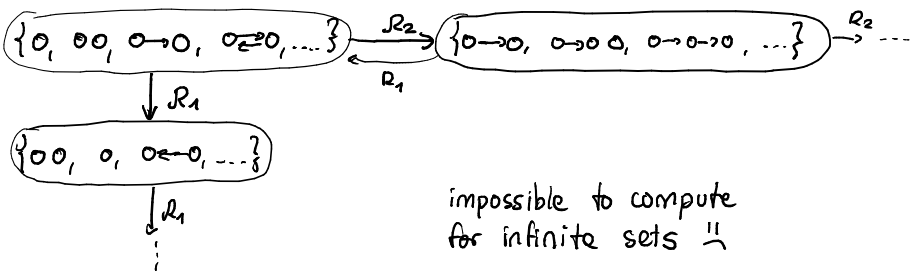


Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"



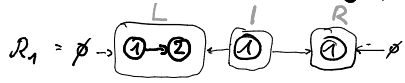
Concrete transition system:



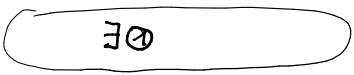
impossible to compute for infinite sets ☹

Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

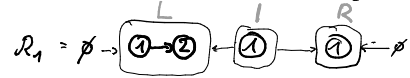


Concrete* transition system:

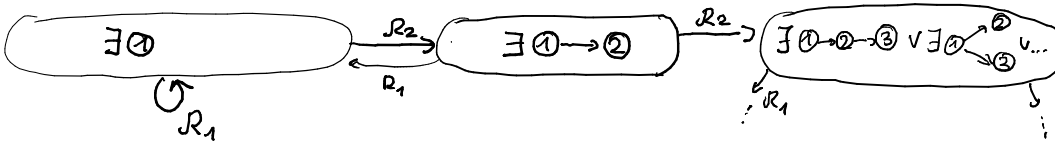


Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"



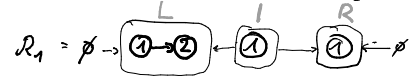
Concrete* transition system:



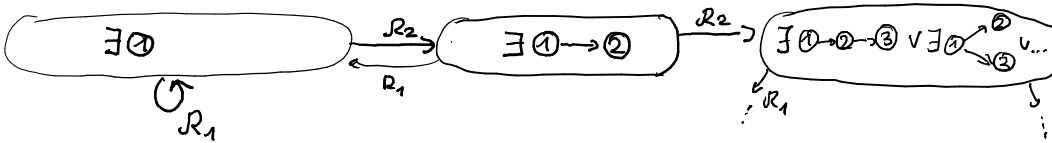
Successors are now computable: $sp(\mathcal{A}, (\ell, r, C)) = \exists r.\,(C \wedge \mathcal{A}_{\downarrow \ell})$

Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"



Concrete* transition system:



Successors are now computable: $sp(\mathcal{A}, (\ell, r, C)) = \exists r.\,(C \wedge \mathcal{A}_{\downarrow \ell})$

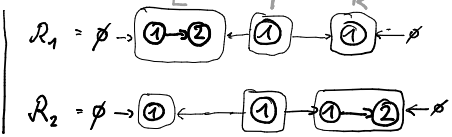
... but transition system is still infinite !!

Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

Abstract transition system with predicates

$p_1 = \exists\, ①$, $p_2 = \exists\, ① \to ②$, $p_3 = \nexists\, ①$



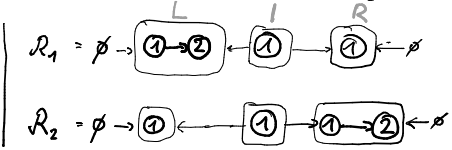
$p_1 \wedge \neg p_3$

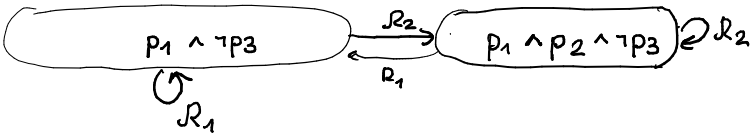
Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

Abstract transition system with predicates

$p_1 = \exists ①$, $p_2 = \exists ① \to ②$, $p_3 = \exists\!\!\!/\ ①$





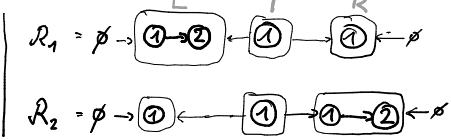
Compute strongest postcondition as before, then pick the best available combination of predicates

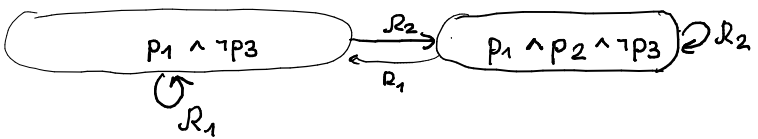
Verifying GTS

"If I start from a non-empty graph, can these rules reach an empty graph?"

Abstract transition system with predicates

$p_1 = \exists ①$, $p_2 = \exists ① \to ②$, $p_3 = \nexists ①$



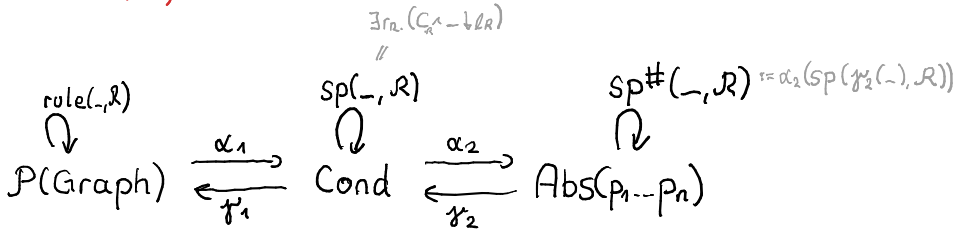


finite !!

Compute strongest postcondition as before, then pick the best available combination of predicates

All reachable states have $\neg p_3$, so empty graph is unreachable!

Abstraction(s)

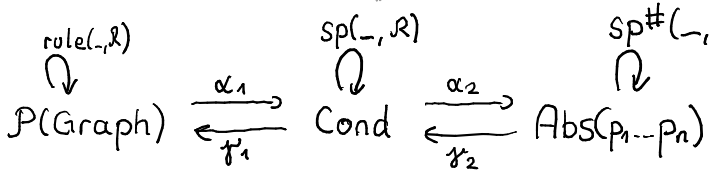


Abstraction(s)

$\exists r_R.(C_{\mathcal{A}} \sqsubseteq b_{LR})$

$sp(\_, \mathcal{R})$ ∥ $sp^{\#}(\_, \mathcal{R}) = \alpha_2(sp(\gamma_2(\_), \mathcal{R}))$

rule(_, $\mathcal{R}$)



$\alpha_2(\mathcal{A})$ = over-approximate $\mathcal{A}$ using available predicates — not computable !!

for each p_i check $\mathcal{A} \models p_i$ and $\mathcal{A} \models \neg p_i$:

- build conjunction of all $(\neg)p_i$ for which $\mathcal{A} \models (\neg)p_i$
- if $\mathcal{A} \not\models (\neg)p_i$ or solver times out, don't add $(\neg)p_i$

↖ satisfiability checkers from previous talks used here ☺

Over-approximation and obtaining suitable predicates

$\mathcal{R} = (\emptyset \to \textcircled{1}\,\textcircled{2} \leftarrow \emptyset \to \emptyset \leftarrow \emptyset,\ \text{true})$ (deletes two nodes at once)

$\text{Init} = \exists \emptyset \to \textcircled{1}\,\textcircled{2}\,\textcircled{3} \leftarrow \emptyset.\ \forall \emptyset \to \textcircled{4} \leftarrow \emptyset.\ \text{false}$ (there are exactly 3 nodes)

$\text{Bad} = \forall \emptyset \to \textcircled{1} \leftarrow \textcircled{1}.\ \text{false}$ (graph is empty)

With predicates $\{\text{Init}, \text{Bad}\}$:

$\hookrightarrow (\text{Init} \wedge \neg \text{Bad})$

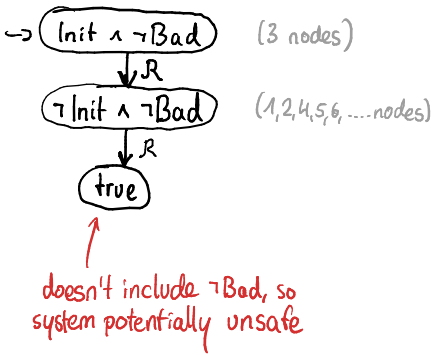
Over-approximation and obtaining suitable predicates

$\mathcal{R} = \left(\emptyset \rightarrow \boxed{① ②} \leftarrow \emptyset \rightarrow \emptyset \leftarrow \emptyset, \text{true}\right)$ (deletes two nodes at once)

$\text{Init} = \exists \emptyset \rightarrow \boxed{① ② ③} \leftarrow \emptyset . \forall \emptyset \rightarrow \boxed{④} \leftarrow \emptyset . \text{false}$ (there are exactly 3 nodes)

$\text{Bad} = \forall \emptyset \rightarrow \boxed{①} \leftarrow \boxed{①} . \text{false}$ (graph is empty)

With predicates $\{\text{Init}, \text{Bad}\}$:



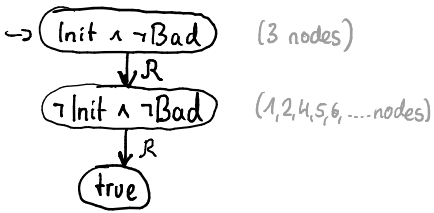
Over-approximation and obtaining suitable predicates

$\mathcal{R} = (\emptyset \to \textcircled{1}\,\textcircled{2} \leftarrow \emptyset \to \emptyset \leftarrow \emptyset, \text{true})$ (deletes two nodes at once)

$\text{Init} = \exists \emptyset \to \textcircled{1}\,\textcircled{2}\,\textcircled{3} \leftarrow \emptyset . \forall \emptyset \to \textcircled{4} \leftarrow \emptyset . \text{false}$ (there are exactly 3 nodes)

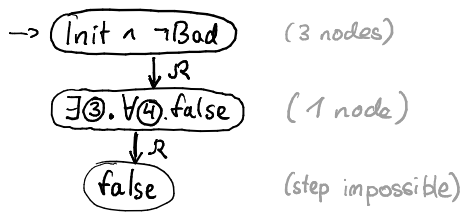
$\text{Bad} = \forall \emptyset \to \textcircled{1} \leftarrow \textcircled{1} . \text{false}$ (graph is empty)

With predicates {Init, Bad}:



doesn't include ¬Bad, so system potentially unsafe

Concrete run (w/o abstraction):



Abstract run on the left is spurious!
(= introduced by the abstraction)

CEGAR loop

Initial predicates : $\{Init, Bad\}$

Construct abstract TS starting from $Init \wedge \neg Bad$ until...

- No more new states & all states have $\neg Bad$
  → verification successful!

- state without $\neg Bad$ is reached → possible counterexample
  - Re-play steps $R_1, \ldots, R_n$ to this state without abstracting:
    $(Init \wedge \neg Bad) \xrightarrow{sp(\_, R_1)} q_1 \xrightarrow{sp(\_, R_2)} q_2 \rightarrow \ldots \xrightarrow{sp(\_, R_n)} q_n$
  - $q_n \models \neg Bad$? → spurious run
    Add $q_1, \ldots, q_n$ to predicate set to eliminate spurious counterexample
    Restart analysis
  - $q_n \not\models \neg Bad$? → verification failed, system has error

To do...

- Define α, γ properly (... α not computable...)
- Proof: $sp^{\#}(\bar{p}, \mathcal{R}) = \alpha(sp(\gamma(\bar{p}), \mathcal{R}))$ is a safe approximation of $sp(\mathcal{A}, \mathcal{R})$

(- Proof: $sp(\_, \mathcal{R})$ is a safe app. of $rule(\_, \mathcal{R})$)

- Write everything down nicely
- ???