Graph Minors and the Analysis of Graph Transformation Systems

Barbara König

Universität Duisburg-Essen, Germany

Joint work with Salil Joshi, IIT Delhi

Overview

Graph Minor Theory

- Graph minor theory by Robertson and Seymour
- Long series of papers (Graph minors I–XXIII)
- Deep graph-theoretical results with applications in computer science (mainly efficient algorithms, complexity theory)
- What about applications in verification?

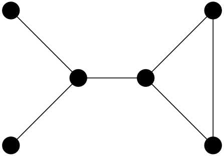
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



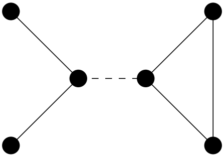
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



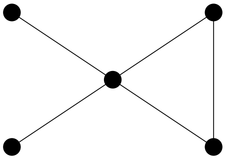
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



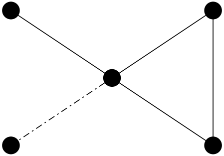
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



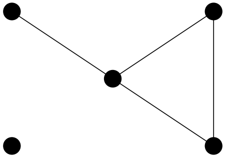
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



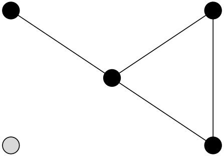
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



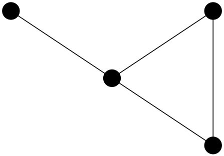
Graph Minor Theory

Minor of a graph

The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



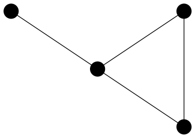
Graph Minor Theory

Minor of a graph

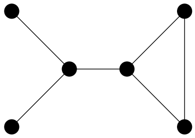
The minors of a graph G can be obtained by (iteratively)

- Deleting edges.
- Deleting isolated nodes.
- Contracting edges.

We write $M \leq G$ if M is a minor of G.



is a minor of



Graph Minor Theory

Graph minor theorem

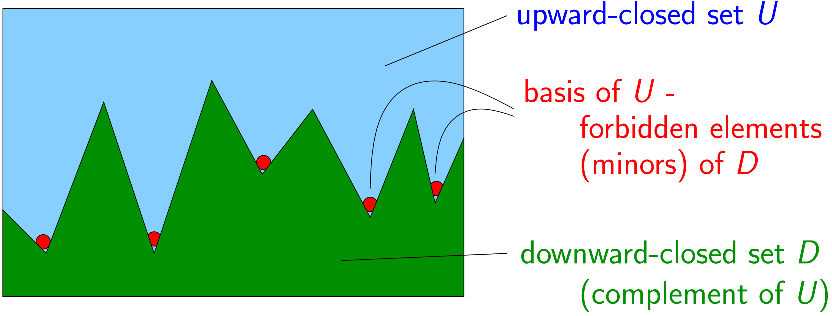
In every infinite sequence $G_0, G_1, G_2, G_3, \ldots$ there exist indices $i < j$ such that G_i is a minor of G_j.

In other words: the minor ordering $\leq$ is a well-quasi-order (wqo).

Graph Minor Theory

Consequences:

- every upward-closed set of graphs has a finite basis (i.e., a finite set of minimal elements).
- every downward-closed set of graphs can be characterized by finitely many forbidden minors.



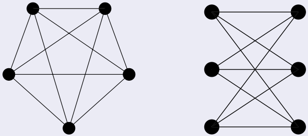
Graph Minor Theory

Downward-closed sets of graphs:

- Graphs that are disjoint unions of paths
- Forests
- Planar graphs
- Graphs that can be embedded in a torus
- ...

Kuratowski's theorem

A graph is planar if and only if it does not contain the K_5 and the $K_{3,3}$ as a minor.



Graph Minor Theory

What about labelled graphs, directed graphs, hypergraphs?

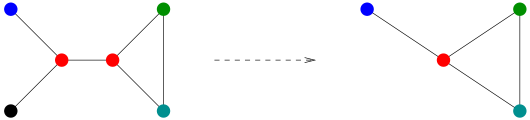
$\leadsto$ The graph minor theorem holds even for labelled hypergraphs!

Minor morphisms (Joshi/König)

$H \leq G$ iff there exists a minor morphism $G \mapsto H$, that is

- there is a partial graph morphism $G \rightharpoonup H$,
- which is surjective, injective on edges *and*
- whenever two nodes v, w of G are mapped to z in H, there exists an (undirected) path between v and w which is contracted.

A minor morphism:



Well-Structured Transition Systems

Well-quasi-orders are also an important ingredient of well-structured transition systems (WSTS) (Finkel/Schnoebelen)

WSTS (Well-structured transition system)

Let S be a set of states, $\Rightarrow$ a transition relation and $\leq$ a partial order on states. The transition system is well-structured if

- $\leq$ is a well-quasi-order
- Whenever $s_1 \leq t_1$ and $s_1 \Rightarrow s_2$, there exists a state t_2 such that $t_1 \Rightarrow^* t_2$ and $s_2 \leq t_2$ (compatibility condition).

$$\begin{array}{ccc} t_1 & \Longrightarrow^* & t_2 \\ \vee\!\mid & & \vee\!\mid \\ s_1 & \Longrightarrow & s_2 \end{array}$$

Well-Structured Transition Systems

The prototypical example for a WSTS are Petri nets:

- States: markings
- Transition relation: firing of transitions as specified by the net
- Well-quasi-order: $m_1 \leq m_2$ if m_2 covers m_1 (m_2 contains at least as many tokens in every place)

Other examples:

- Context-free string rewrite systems
- Basic process algebra
- "Lossy" systems
- Systems with home-states

Well-Structured Transition Systems

Backward Reachability

Take a set $I \subseteq S$ of states and compute $Pred^*(I)$ (the set of all predecessors) as the limit of the sequence

$$I_0 = I \qquad\qquad I_{i+1} = I_i \cup Pred(I_i),$$

where *Pred* returns the direct predecessors of a set of states.

Backward Reachability and WSTS

In the case of WSTS it holds that

- If I is upward-closed (and hence representable by a finite basis), then $Pred^*(I)$ is upward-closed.
- The sequence $I_0, I_1, I_2, \dots$ eventually becomes stationary, i.e., $I_n = I_{n+1}$ and $Pred^*(I) = I_n$.

Well-Structured Transition Systems

Covering problem

Covering problem: Given an initial state s_0 and another state s_f. Can we reach a state s from s_0, i.e., $s_0 \Rightarrow^* s$ such that $s \geq s_f$?

The covering problem for WSTS is decidable if we can effectively compute a finite basis for (the upward-closure of) $Pred(I)$ whenever we have a finite basis for I.

Well-Structured Transition Systems

Question: can we view (some) graph transformation systems as well-structured transition systems?

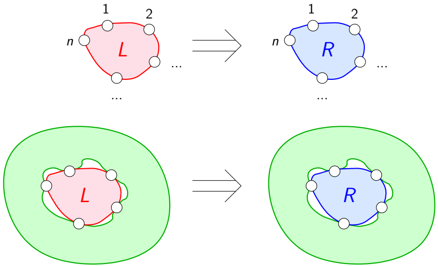
But first a short introduction to graph transformation . . .

Graph transformation systems

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

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

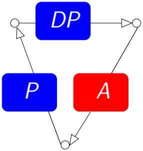
A graph transformation system (GTS) consists of an initial (hyper-)graph and a set of productions/rules:



Running example: Termination detection

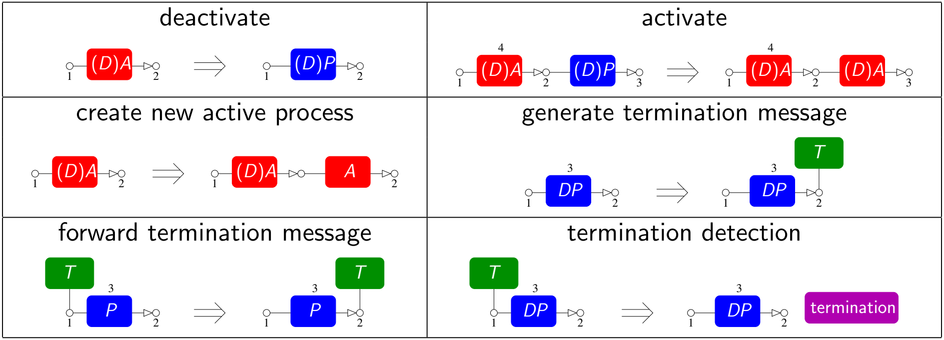
- A ring consisting of active and passive processes.

Start graph:



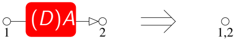
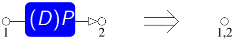
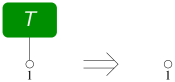
- Active processes may become passive at any time.
- Active processes may activate passive processes and create new active processes.
- There is a special process (the detector *DA*, *DP*) that may generate a message for termination detection.
- This message is forwarded by passive processes and received by the (passive) detector which then declares termination.

Running example: Termination detection



Running example: Termination detection

Additionally: The system is unreliable. Processes may leave the ring at any time and messages may get lost.

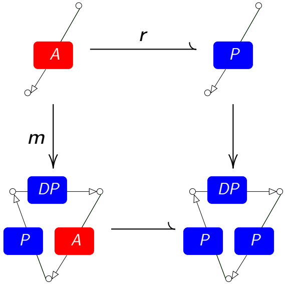
active process leaves	passive process leaves
	
message is lost	**termination flag is lost**
	

SPO (single pushout) rewriting rules, given by partial graph morphisms from the left-hand side to the right-hand side.

Single-pushout approach

Take the pushout of the partial rule morphism ($r\colon L \rightharpoonup R$) and the total match ($m\colon L \to G$) in the category of partial graph morphisms in order to obtain the resulting graph H.

$$\begin{tikzcd} L \arrow[r, harpoon, "r"] \arrow[d, "m"'] & R \arrow[d] \\ G \arrow[r, harpoon] & H \end{tikzcd}$$



Construct H by

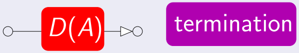
- deleting elements of G which are undefined under r
- creating elements which are new in R

It can be shown that minor morphisms are preserved by pushouts along total morphisms (important for our theory!)

Running example: Termination detection

Correctness

- Are the rules correct?
- That is, can we reach a graph where termination has been declared, but there are still active processes?
- Can we reach a graph which contains the following graph as a subgraph?



$\rightsquigarrow$ View graph transformation as a WSTS and solve the covering problem for the graph above via backward analysis!

GTS as Well-Structured Transition Systems

Graph transformation systems are in general Turing-complete $\rightsquigarrow$ not all GTS can be well-structured

But some subclasses are WSTS with respect to the minor ordering:

- Context-free graph grammars
- GTS where the left-hand sides consist of disconnected edges
- GTS which contain edge contraction rules for every edge label ("lossy" system)

GTS as Well-Structured Transition Systems

Obtaining a WSTS by adding edge contraction rules

$$\begin{array}{ccc} H_1 & & \\ \vee\!| & & \\ G_1 & \overset{r}{\Longrightarrow} & G_2 \end{array}$$

If G_1 is a minor of H_1 and G_1 is rewritten to G_2 ...

GTS as Well-Structured Transition Systems

Obtaining a WSTS by adding edge contraction rules

$$
\begin{array}{lcl}
H_1 & \overset{*}{\Longrightarrow} H' & \\
\vee\!| & & \\
G_1 & \xrightarrow{\quad r \quad} & G_2
\end{array}
$$

…then H_1 contains a possibly disconnected left-hand side which can be contracted via the edge contraction rules, resulting in H' and …

GTS as Well-Structured Transition Systems

Obtaining a WSTS by adding edge contraction rules

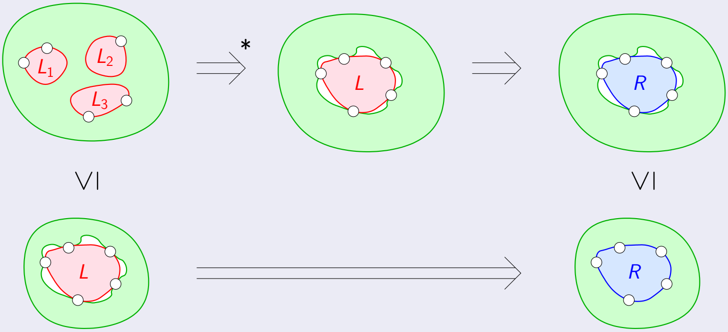
$$
\begin{array}{ccccc}
H_1 & \Longrightarrow^* & H' & \xRightarrow{r} & H_2 \\
\rotatebox{90}{$\leq$} & & & & \rotatebox{90}{$\leq$} \\
G_1 & & \xRightarrow{\quad\quad r \quad\quad} & & G_2
\end{array}
$$

... H' can be rewritten to H_2 (of which G_2 is a minor) by using the same rule as for G_1.

GTS as Well-Structured Transition Systems

$$\begin{array}{ccccc} H_1 & \overset{*}{\Longrightarrow} & H' & \overset{r}{\Longrightarrow} & H_2 \\ \vee\!| & & & & \vee\!| \\ G_1 & & \overset{r}{\Longrightarrow} & & G_2 \end{array}$$

...



Backward Analysis

What remains to be done in order to perform the backward analysis?

> Given a finite basis F for an upward-closed set of graphs $\mathcal{U}$ we have to compute a finite basis for (the upward-closure of) $Pred(\mathcal{U})$.

Ideas:

- Given a graph $H \in F$, apply all rules backward.
- But: H need not contain the full right-hand side, parts of it might have been lost by the minor operations (edge/node deletions, edge contractions)

  $\leadsto$ Instead of taking all rules $r\colon L \rightharpoonup R$ take as rules $L \overset{r}{\rightharpoonup} R \overset{\mu}{\mapsto} M$, where μ is an arbitrary minor morphism, i.e., take minors of the right-hand side.

Backward Analysis

Why does this work?

Let $H \in \mathcal{U}$.

$$\begin{tikzcd} L \arrow[r, hook, "r"] & R \arrow[r, mapsto, "\mu"] & M \arrow[d, "m'"] \\ & & H \end{tikzcd}$$

Find a match of a minor M of the right-hand side in H.

Backward Analysis

Why does this work?

Let $H \in \mathcal{U}$.

$$\begin{tikzcd} L \arrow[r, "r"] \arrow[d, "m"] & R \arrow[r, mapsto, "\mu"] & M \arrow[d, "m'"] \\ G \arrow[rr] & & H \end{tikzcd}$$

Make a backward step by applying the rule backward (find a pushout complement).

Backward Analysis

Why does this work?

Let $H \in \mathcal{U}$.

$$\begin{tikzcd} L \arrow[r, hook, "r"] \arrow[d, "m"] & R \arrow[r, mapsto, "\mu"] \arrow[d] & M \arrow[d, "m'"] \\ G \arrow[r, hook] & \hat{H} \arrow[r, mapsto] & H \end{tikzcd}$$

This pushout splits into two pushouts (standard pushout splitting).
$\leadsto$ G can be rewritten to $\hat{H}$ and H is a minor of $\hat{H}$ (since minor morphisms are preserved by pushouts).

Backward Analysis

Why does this work?

Let $H \in \mathcal{U}$.

$$\begin{tikzcd} L \arrow[r, hook, "r"] \arrow[d, "m"] & R \arrow[r, mapsto, "\mu"] \arrow[d] & M \arrow[d, "m'"] \\ G \arrow[r, hook] & \hat{H} \arrow[r, mapsto] & H \end{tikzcd}$$

$\Rightarrow$ $\hat{H} \in \mathcal{U}$ and $G \in \mathit{Pred}(\mathcal{U})$, i.e., the procedure is correct.

Backward Analysis

Why does this work?

Let $H \in \mathcal{U}$.

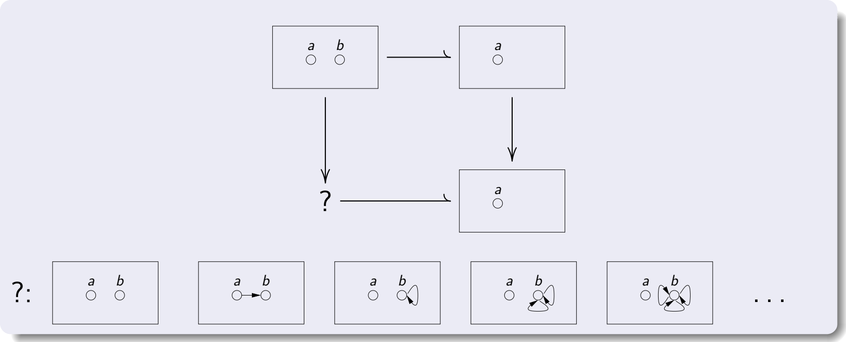
$$\begin{tikzcd} L \arrow[r, hook, "r"] \arrow[d, "m"] & R \arrow[r, mapsto, "\mu"] \arrow[d] & M \arrow[d, "m'"] \\ G \arrow[r, hook] & \hat{H} \arrow[r, mapsto] & H \end{tikzcd}$$

$\Rightarrow$ $\hat{H} \in \mathcal{U}$ and $G \in \mathit{Pred}(\mathcal{U})$, i.e., the procedure is correct.

Completeness, i.e., the fact that we generate the entire basis, also holds, but is more difficult to prove.

Backward Analysis

Another problem: in the category of partial morphisms, there are usually infinitely many pushout complements.



$\rightsquigarrow$ It is sufficient to compute only the minimal pushout complements with respect to the minor ordering. We have an algorithm which does this.

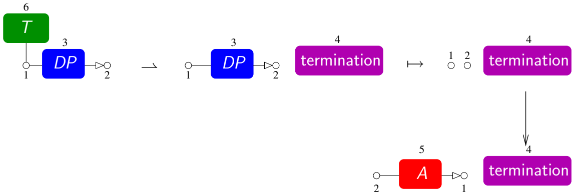
Backward Analysis

Backward analysis for the running example:

termination

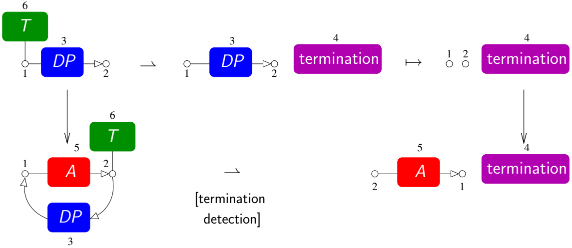
Backward Analysis

Backward analysis for the running example:



Backward Analysis

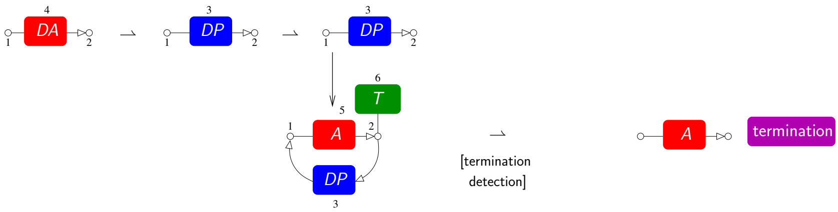
Backward analysis for the running example:



Apply rule [termination detection] backward.

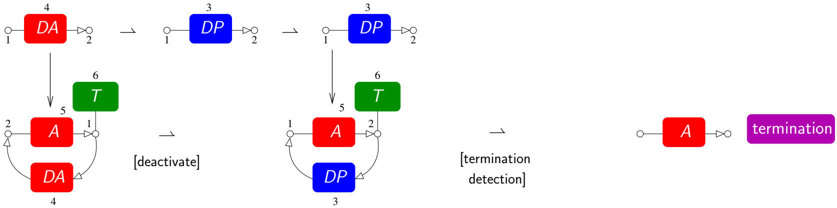
Backward Analysis

Backward analysis for the running example:



Backward Analysis

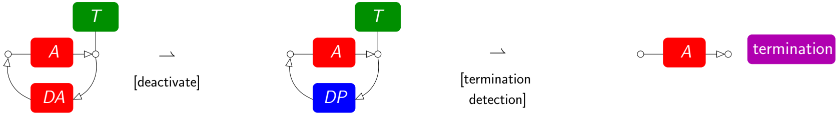
Backward analysis for the running example:



Apply rule [deactivate] backward.

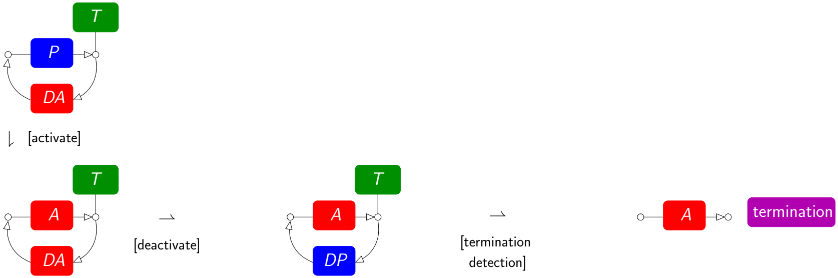
Backward Analysis

Backward analysis for the running example:



Backward Analysis

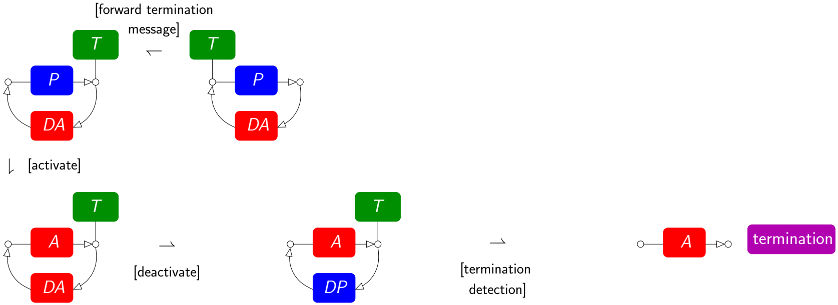
Backward analysis for the running example:



Apply rule [activate] backward.

Backward Analysis

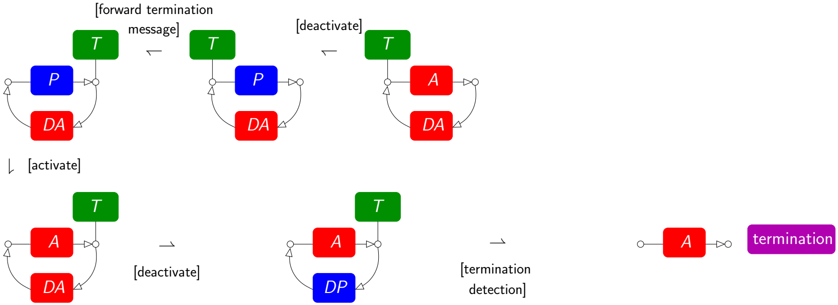
Backward analysis for the running example:



Apply rule [forward termination message] backward.

Backward Analysis

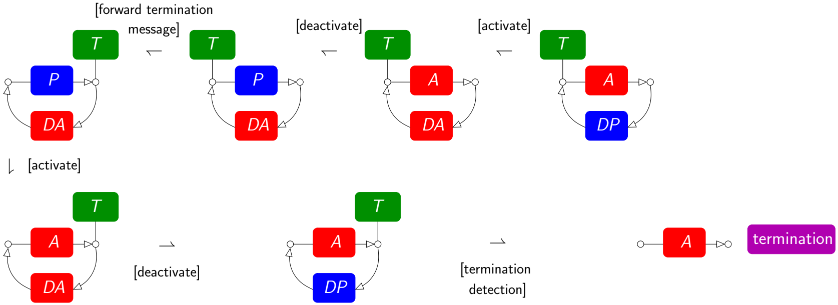
Backward analysis for the running example:



Apply rule [deactivate] backward.

Backward Analysis

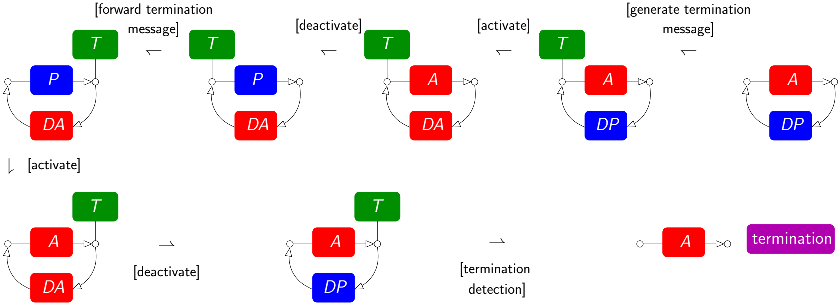
Backward analysis for the running example:



Apply rule [activate] backward.

Backward Analysis

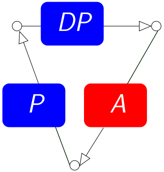
Backward analysis for the running example:



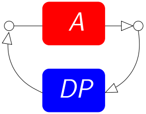
Apply rule [generate termination message] backward.

Backward Analysis

The last graph in this chain is a minor of the start graph!



$\mapsto$



This means that the error graph is indeed coverable and the termination detection rules are wrong.

Reason: after a passive detector sends a termination message he has to record whether he became again active (and then passive) before receiving this message

$\leadsto$ Rules have to be changed accordingly. Then the property can be verified (since this a decision procedure).

Conclusion

Efficiency and Implementation

- We still need an implementation
- The implementation will enable us to answer the following questions more precisely:
  - How efficient is the technique?
  - After how many steps does the backward analysis terminate?
  - Usually, how large is the basis of forbidden minors representing the error graphs?
- Apart from match finding and computation of pushout complements we have to implement a procedure which checks whether a graph G is a minor of H.

  This is in PTIME (in the size of H), due to Robertson and Seymour, but the algorithm has unreasonably large constants.

  $\rightsquigarrow$ we will probably use a heuristics which does not guarantee polynomial runtime

Conclusion

Backward reachability vs. forward reachability:

- Most known verification techniques for GTS use forward reachability (sometimes with an abstraction mechanism)
- One notable exception:
  Saksena, Wibling, Jonsson: "Graph Grammar Modeling and Verification of Ad Hoc Routing Protocols", TACAS '08
- Backward reachability seems to be more convenient for handling negative application conditions
- However: the backward analysis generates many graphs that are not reachable from the start graph $\rightsquigarrow$ efficiency problem!

  Idea: combine with forward techniques in order to reduce the state space

  Forward reachability algorithms for WSTS are known, but are considerately more complex than backward algorithms

Conclusion

Relation to graph transformation frameworks

- Single-pushout vs. double-pushout: we use single-pushout since we have to handle partial morphisms anyway (minor morphisms are partial)
- What about using an abstract categorical framework (adhesive categories)?
  $\rightsquigarrow$ doubtful, how should we handle minors?
- Matches: here we use conflict-free matches (for technical reasons), what about arbitrary or injective matches?