Carl von Ossietzky Universität Oldenburg

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Resilience of Graph Transformation Systems: Modeling Adverse Conditions and Infinite-state Verification

Technischer Bericht

Okan Özkan
o.oezkan@informatik.uni-oldenburg.de

23. Januar 2023

Abstract

Graph transformation systems are a visualizable, yet mathematically precise formalism for modeling systems. In this perception, system states are captured by graphs and state changes by graph transformations. Other model formalisms can often be easily translated into the graph-transformational setting.

Correctness and *resilience* are broadly used concepts in computer science and software engineering. For systems in which correctness w.r.t. a safety condition is unachievable, *fast recovery* is demanded. In general systems engineering, fast recovery from a degraded state is often termed resilience. Correctness *under adverse conditions* addresses systems interacting with an adversary environment. Correctness in this context means that the interaction between system and environment satisfies desired behavioral properties.

The use of visual modeling techniques alone does not guarantee the resilience of a design. In the context of rising standards for trustworthy systems, there is a growing need for formal verification of graph transformation systems.

First, we present an approach for modeling adverse conditions by graph transformation systems. We show how to express resilience notions in temporal logics. Second, we investigate on the decidability of resilience problems. Given a graph transformation system, a graph set INIT, and graph constraints *Bad* and *Safe*, we ask: starting from any INITial graph, whenever we reach a graph satisfying *Bad*, can we reach a graph satisfying *Safe* in a bounded number of steps? Solving the corresponding problem for well-structured transition systems and applying it to graph transformation systems, we obtain decidability results for suitable subclasses of graph transformation systems of bounded path length and so-called proper graph constraints *Bad* and *Safe*. Moreover, we identify sufficient, rule-specific criteria for decidability.

Zusammenfassung

Graphtransformationssysteme sind ein visualisierbarer Formalismus für das Modellieren von Systemen. Systemzustände werden als Graphen aufgefasst und Zustandsänderungen als Graphtransformationen. Andere Modell-Formalismen können oft einfach in den Graphtransformations-Formalismus übersetzt werden.

Korrektheit und *Resilienz* sind Konzepte, die breite Anwendung in der Informatik und Softwaretechnik finden. Für Systeme, in denen Korrektheit nicht erreichbar ist, ist eine *schnelle Wiederherstellung* gefragt. In der allgemeinen Systemtechnik wird eine schnelle Wiederherstellung ausgehend von einem schlechten Zustand oft als Resilienz bezeichnet. Korrektheit *unter widrigen Umständen* spricht Systeme an, die mit einer Umgebung als Kontrahenten interagieren. Korrektheit in diesem Sinne bedeutet, dass die Interaktion zwischen System und Umgebung erwünschte Verhaltenseigenschaften erfüllt.

Der bloße Einsatz visueller Modellierungsmethoden garantiert noch keine Resilienz des Designs. Im Zuge steigender Anfordungen an zuverlässige Systeme gibt es einen zunehmenden Bedarf an formaler Verifikation von Graphtransformationssystemen.

Zunächst präsentieren wir einen Ansatz für die Modellierung von widrigen Umständen durch Graphtransformationssysteme. Wir zeigen, wie Resilienz-Begriffe in temporaler Logik ausgedrückt werden können. Anschließend untersuchen wir die Entscheidbarkeit von Resilienz-Problemen. Gegeben ein Graphtransformationssystem, eine Graphenmenge INIT und Graphbedingungen *Bad* und *Safe*, stellen wir folgende Frage: Beginnend von einem INITialen Graphen, können wir, wann immer wir einen Graphen, der *Bad* erfüllt, erreichen, in einer beschränkten Anzahl an Schritten einen Graphen, der *Safe* erfüllt, erreichen? Durch das Lösen des korrespondierenden Problems für wohl-strukturierte Transitionssyteme und das Anwenden auf Graphtransformationssysteme erhalten wir Entscheidbarkeitsresultate für geeignete Teilklassen von Graphtransformationssystemen von beschränkter Pfadlänge und sogenannte propere Graphbedingungen *Bad* und *Safe*. Darüberhinaus ermitteln wir hinreichende, regelspezifische Kriterien für die Entscheidbarkeit.

Contents

Chapter 1

Introduction

There is a great demand for visual notations in the software development process and related activities. *Graph transformation systems (GTSs)*, as considered, e.g., in Ehrig et al. [EHK+97, EEPT06], are a visualizable, yet mathematically precise formalism for modeling a system. In this perception, system states are captured by graphs and state changes by graph transformations. Topics in graph transformation are growing in popularity. Other model formalisms can often be easily translated into the graph-transformational setting, see, e.g., Baldan et al. [BCM05].

Trustworthiness is a fundamental paradigm for contemporary system design. As a matter of fact, *correctness* of computational systems has proven to be a key principle throughout almost all areas of computer science, e.g., Apt et al. [AOdB09]. In particular, system correctness under *adverse conditions* is a topic of recent research, see, e.g., Olderog et al. [OFTK21]. This concept addresses systems which interact with an *environment* and finds use also in the development of evolving technologies such as autonomous driving (Schwammberger [Sch18]) and neural networks (Worzyk et al. [WKK19]). For further topics, consult, e.g., [OFTK21]. Correctness in this context means that the interaction between system and environment satisfies desired behavioral properties. One assumes that the environment exhibits an only partially predictable behavior or may have an adverse effect.

Resilience is a broadly used concept in computer science and software engineering with varying notions, see, e.g., Jackson & Ferris [JF13]. For systems in which correctness w.r.t. a safety constraint is unachievable, *fast recovery* is demanded. In general systems engineering, fast recovery from a degraded state is often termed resilience, e.g., Trivedi et al. [TKG09]. Resilience can be classified as a special case of system correctness under adverse conditions.

The use of visualizable modeling techniques alone does not guarantee the resilience of a design. In view of rising standards for trustworthy systems, there is a growing need for formal verification of graph transformation systems.

This thesis addresses this need.

Our first objective tackles the following questions:

(1) How can adverse conditions be modeled by graph transformation systems?

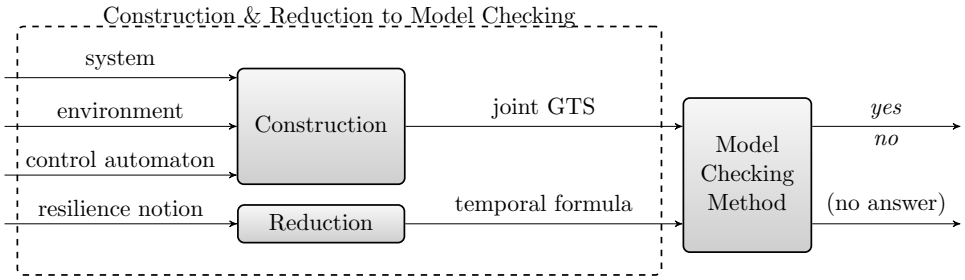
(2) What are meaningful notions of resilience in the setting of adverse conditions?

(3) How can resilience be checked?

We present an approach for modeling adverse conditions by graph transformation systems. That is, we show how to construct a *joint graph transformation system* from a system, an environment, and a control automaton. Both, system and environment, are graph transformation systems. The control automaton is modeling their interaction. We introduce meaningful resilience notions for joint graph transformation systems. These notions capture the recovery of a safety constraint Safe after an interference of the environment Env, i.e.,

$$\text{Interference of Env} \Rightarrow \text{Recovery of Safe.}$$

Resilience of the joint system generalizes classical correctness of safety constraints. A safety constraint may be violated but must be recovered within a specified time window. We express these resilience notions by means of temporal logics s.t. a model checker, e.g., the tool GROOVE by Kastenberg & Rensink [KR06] can be applied. The temporal logics we use are linear temporal logic (LTL) and computation tree logic (CTL), see, e.g., Baier & Katoen [BK08]. As atomic propositions we use graph constraints.



We perform the construction and reduction as preparation for model checking of meaningful resilience notions. Unfortunately, decidability is, in general, not guaranteed and the model checker might deliver no result. The reason is that graph transformation systems are, in general, working over infinite sets of graphs. For finite-state graph transformation systems, this approach works without major obstacles.

Our second objective is the verification of resilience for graph transformation systems. Abstracting from the original setting, we investigate on the following bounded resilience problem for marked graph transformation systems each consisting of a graph transformation system together with a graph class closed under rule application and an INITial subset of graphs: Given a marked graph transformation system and sets BAD and SAFE of graphs, we ask whether there exists a bound k s.t. starting from any INITial graph, whenever we reach a BAD graph, we can reach a SAFE graph in at most k steps. A BAD (SAFE) graph is a graph satisfying a graph constraint Bad (Safe). The bounded resilience problem can analogously be formulated for marked systems, replacing the graph constraints Bad and Safe by subsets BAD and SAFE of states.

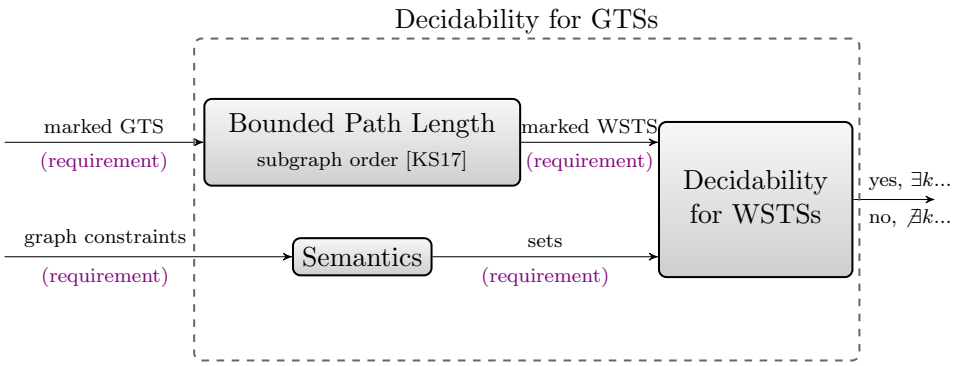
(1) Under which general conditions of a system class is the bounded resilience problem decidable?

(2) For which subclasses of marked graph transformation systems and graph constraints is the bounded resilience problem decidable?

(3) What are sufficient, rule-specific criteria for decidability?

In order to analyze graph transformation systems, we consider their induced *transition systems*. A transition system consists of a set of states of any kind (not necessarily graphs) and a transition relation on the state set. Usually, the state set – also in the context of graph transformation systems – is infinite. For handling infinite state sets (sets of graphs), we exploit the concept of well-structuredness, e.g., Abdulla et al. [AČJT96], Finkel & Schnoebelen [FS01], König & Stückrath [KS17]. A *well-structured transition system (WSTS)* is, informally, a transition system equipped with a *well-quasi-order* satisfying that "larger" states simulate "smaller" states and that certain predecessor sets can be effectively computed. In this well-structured setting, *ideal-based* sets (*upward-* or *downward-closed sets*) play an important role. They enjoy a number of properties simplifying verification such as a finite representation and closure properties. For well-structured transition systems, the *ideal reachability problem* is decidable, see, e.g., Abdulla et al. [AČJT96], Finkel & Schnoebelen [FS01], which is an integrant of our results.

Well-structuredness of graph transformation systems is investigated, e.g., in König & Stückrath [KS17] for several well-quasi-orders. The well-quasi-order we use is the *subgraph order* which permits strong compatibility but comes with the restriction of the boundedness of the path length on the graph class. We show decidability for suitable subclasses of well-structured transition systems. Applying them, we obtain decidability results for suitable subclasses graph transformation systems of bounded path length, as illustrated below.



Each subclass exhibits additional requirements, i.e., effectiveness or unreliability properties. Ultimately, we apply our results to joint graph transformation systems, i.e., to the setting of adverse conditions. This is a special case of the decidability results for graph transformation systems.

Thesis Structure

In Chapter 2, we present the main concepts used in this thesis, namely, graph transformation systems and well-structured transition systems. In Chapter 3, we present an approach for modeling adverse conditions by graph transformation systems and express resilience notions in temporal logics. In Chapter 4, we show the decidability of resilience problems for suitable subclasses of well-structured transition systems. In Chapter 5, we show the decidability of resilience problems for subclasses of graph transformation systems (including results for the setting of Chapter 3) by applying our results from Chapter 4. Moreover, we identify sufficient criteria of graph transformations for decidability. In Chapter 6, we recapitulate related concepts, give a concluding summary, and discuss further topics. In the Appendix, related formalisms, proofs, and computations can be found.
The preliminaries on graph transformation systems are orientated on Ehrig et al. [EHK+97, EEPT06]. The preliminaries on well-structured transition systems are orientated on Abdulla et al. [AČJT96], Finkel & Schnoebelen [FS01], and König & Stückrath [KS17].

The following publications form the basis of this work. Their content has been revised and extended.

[Özk20] Okan Özkan. Modeling Adverse Conditions in the Framework of Graph Transformation Systems. In *Proc. Graph Computational Models*, volume 330 of *EPTCS*, pages 35–54, 2020. ⇒ Chapter 3 is based on this paper.

[Özk21] Okan Özkan. Infinite-state Graph Transformation Systems under Adverse Conditions. *it Inf. Technol.*, 63(5–6): 311–320, 2021. ⇒ This is an overview paper which contains ideas used in Chapter 3, 4, and 5.

[ÖW21] Okan Özkan and Nick Würdemann. Resilience of Well-structured Graph Transformation Systems. In *Proc. Graph Computational Models*, volume 350 of *EPTCS*, pages 69–88, 2021. ⇒ Some parts of Chapter 4 and 5 are based on this paper.

[Özk22] Okan Özkan. Decidability of Resilience for Well-structured Graph Transformation Systems. In *Proc. Int. Conference on Graph Transformation*, volume 13349 of *LNCS*, pages 38–57, 2022. ⇒ Most parts of Chapter 4 and 5 are based on this paper.

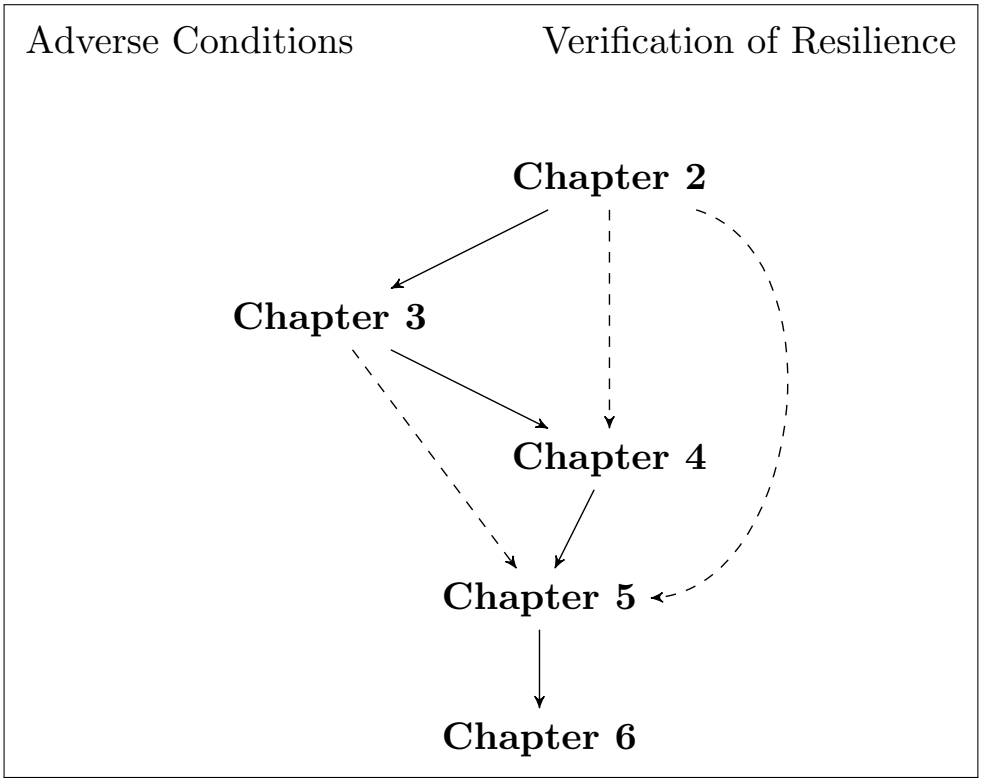


Figure 1.1: Reading guide.

In Figure 1.1, ways to read this thesis are depicted. The solid arrows represent the recommended reading directions, i.e., in the numerical order. The dashed edges are other possible reading directions.

This work comprises three main aspects: the modeling of adverse conditions, the verification of resilience for well-structured transition systems, and the verification of resilience for graph transformation systems. The

modeling of adverse conditions is based on graph transformation systems, graph constraints, and temporal logics. The verification of resilience for well-structured transition systems is based on the established theory of well-structuredness which uses the concept of ideals in well-quasi-orders. The verification of resilience for graph transformation systems is based on our results for well-structured transition systems and the theory of graph transformation systems. Our results for graph transformation systems can be used for the verification in the context of adverse conditions. An overview is depicted in Figure 1.2.

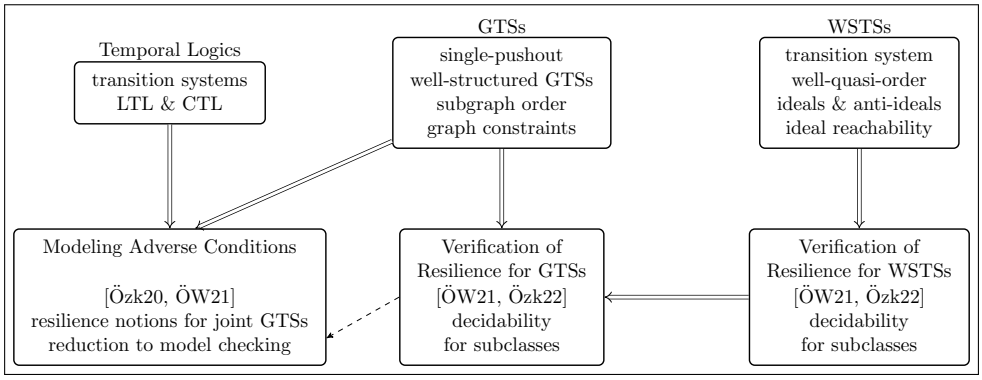


Figure 1.2: Overview of this thesis.

Chapter 2

Preliminaries

We present the main concepts used in this thesis, namely graph transformation systems and well-structured transition systems. In Section 2.1, we give a review on the basic notions of graphs and graph transformation systems. In Section 2.2, we recapitulate the basic notions of well-structured transition systems.

2.1 Graph Transformation Systems

In the following, we present the definitions of graphs, graph constraints, rules, and graph transformation systems, see, e.g., Ehrig et al. [EPS73, EHK+97, EEPT06].

A directed, labeled graph consists of a set of nodes and a set of edges where each edge is equipped with a source and a target node and where each node and edge is equipped with a label. This kind of graphs are a special case of the hypergraphs introduced in [Hab92].

Definition 2.1 (graphs). A *(directed) (labeled)*[1] *graph* over a finite label alphabet $\Lambda = \Lambda_V \cup \Lambda_E$ is a tuple $G = \langle V, E, \mathrm{src}, \mathrm{tgt}, \mathrm{lab}^V, \mathrm{lab}^E \rangle$, with finite sets V and E of *nodes* (or *vertices*) and *edges*, functions $\mathrm{src}_G, \mathrm{tgt}_G : E \to V$ assigning *source* and *target* to each edge, and *labeling functions* $\mathrm{lab}^V : V_G \to \Lambda_V$, $\mathrm{lab}^E : E_G \to \Lambda_E$. We may denote the components of a graph G with an index G. A node v is *incident* to an edge e, and vice versa, if $\mathrm{src}(e) = v$ or $\mathrm{tgt}(e) = v$.

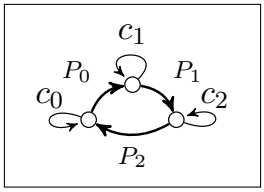
Convention. We draw graphs as follows. Nodes are drawn as circles and edges as arrows. Labels are indicated by a symbol or a color. If we consider only one label, we do not indicate the label. Opposite edges are drawn as one line without an arrow tip or as one line with two arrow tips.

[1] In definitions, words in brackets are optional, i.e., we allow to leave out words in brackets. E.g., if *(directed) (labeled) graphs* are defined, we allow to speak of "directed, labeled graphs", "labeled graphs", "directed graphs", as well as shortly of "graphs".

Example 2.1. Consider a graph G with the nodes v_0, v_1, v_2, the edges $e_0, e_1, e_2, e'_0, e'_1, e'_2$, and the following source, target, and labeling functions:

$$\begin{array}{llll} & & \mathrm{lab}^V(v_i) = \square & \text{for } i = 0,1,2 \\ \mathrm{src}(e_i) = v_i, & \mathrm{tgt}(e_i) = v_i, & \mathrm{lab}^E(e_i) = c_i & \text{for } i = 0,1,2 \\ \mathrm{src}(e'_i) = v_i, & \mathrm{tgt}(e'_i) = v_{i+1}, & \mathrm{lab}^E(e'_i) = P_i & \text{for } i = 0,1 \\ \mathrm{src}(e'_2) = v_2, & \mathrm{tgt}(e'_2) = v_0, & \mathrm{lab}^E(e'_2) = P_2 & \end{array}$$

A drawing representation of G is shown below.



It consists of three nodes with the same label ($\square$), three loops with three different labels c_0, c_1, c_2, and three edges, which are not loops, with three different labels P_0, P_1, P_2. Every node is incident to one loop and has, besides the loop, one incoming and one outgoing edge.

Graph morphisms consist of partial functions between nodes and edges, which preserve the graph structure.

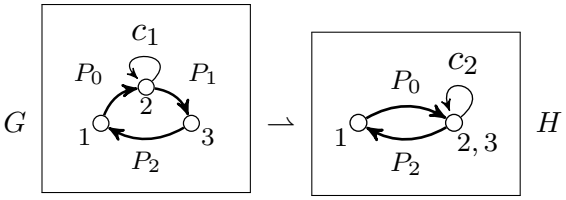
Definition 2.2 (graph morphisms). Given graphs G and H, a *(partial) (graph) morphism* $g : G \rightharpoonup H$ consists of partial functions $g_V : V_G \rightharpoonup V_H$ and $g_E : E_G \rightharpoonup E_H$ which preserve sources, targets, and labels, i.e., $g_V \circ \mathrm{src}_G(e) = \mathrm{src}_H \circ g_E(e)$, $g_V \circ \mathrm{tgt}_G(e) = \mathrm{tgt}_H \circ g_E(e)$, $\mathrm{lab}^V_G(v) = \mathrm{lab}^V_H \circ g_V(v)$, and $\mathrm{lab}^E_G(e) = \mathrm{lab}^E_H \circ g_E(e)$ on all nodes v and edges e, for which $g_V(v), g_E(e)$ is defined. Furthermore, if a morphism is defined on an edge, it must be defined on all incident nodes. The morphism g is *total* (*injective*, *surjective*) if both g_V and g_E are total (injective, surjective). If g is total (and injective), we write $g : G \to H$ ($g : G \hookrightarrow H$). The composition of morphisms is defined componentwise. An *isomorphism* is a total, bijective morphism. Two graphs G and H are *isomorphic* if there exists an isomorphism from G to H.

Convention. In drawings of (partial) morphisms, we equip the image of a node with the same index. Nodes on which the (partial) morphism is undefined have no index.

Example 2.2. Consider the graph G with nodes v_1, v_2, v_3, edges e_1, e_2, e_3 labeled with P_0, P_1, P_2, and a c_1-labeled loop e_4 and the graph H with nodes v'_1, v'_2, edges e'_1, e'_2 labeled with P_0, P_2, and a c_2-labeled loop e'_3, both as in the drawing representations below. The morphism $g : G \rightharpoonup H$ is formally given by the partial functions $g_V : \{v_1, v_2, v_3\} \rightharpoonup \{v'_1, v'_2\}$ and $g_E : \{e_1, e_2, e_3, e_4\} \rightharpoonup \{e'_1, e'_2, e'_3\}$:

$$g_V(v_1) = v_1', \quad g_V(v_2) = v_2', \quad g_V(v_3) = v_2',$$
$$g_E(e_1) = e_1', \quad g_E(e_3) = e_2'.$$

It is represented as follows:



The nodes v_1, v_2, v_3 with indices $1, 2, 3$ are mapped to the nodes with the indices $1, 2, 3$. In particular, the nodes with indices 2 and 3 are identified.The morphism g is undefined on the c_1-labeled loop and the P_1-labeled edge. The c_2-labeled loop is not in the image of the morphism.

Graph constraints are well-known (e.g., Rensink [Ren04], [HP09]) and equivalent to first-order graph formulas in the sense of Courcelle [Cou90, Cou97]. We focus on a special case of graph constraints, which are non-nested and based on "pure positive" / "pure negative" constraints, considered, e.g., in [HST18]. A definition of general graph constraints can be found in Appendix A.

Definition 2.3 (positive & negative constraints). A *pure positive constraint* is an expression of the form $\exists C$ where C is a graph. A *pure negative constraint* is an expression of the form $\neg\exists C$ where C is a graph. A *pure constraint* is a pure positive or pure negative constraint.
The class of *positive constraints* is the smallest class of expressions, which (i) contains all pure positive constraints and (ii) is closed under $\wedge, \vee$. The class of *negative constraints* is the smallest class of expressions, which (i) contains all pure negative constraints and (ii) is closed under $\wedge, \vee$. A *proper*[2] *constraint* is a positive or negative constraint.
The class of *non-nested constraints* is the smallest class of expressions, which (i) contains all pure positive constraints and (ii) is closed under $\neg, \wedge, \vee$.
A graph G *satisfies* $\exists C$ if there is a total, injective morphism $C \hookrightarrow G$. It satisfies the negation $\neg\exists C$ (the negation $\neg c$) if it does not satisfy $\exists C$ (the non-nested constraint c). A graph satisfies a disjunction of non-nested constraints if it satisfies one of them, and a conjunction of non-nested constraints if it satisfies both of them. We write $G \models c$ if the graph G satisfies the non-nested constraint c. For a non-nested constraint c, we denote by $[\![c]\!]$ the class of all graphs G with $G \models c$. Two non-nested constraints c, c' are *equivalent* if $[\![c]\!] = [\![c']\!]$.

Assumption. In the following, we focus on proper, i.e., positive and negative constraints.

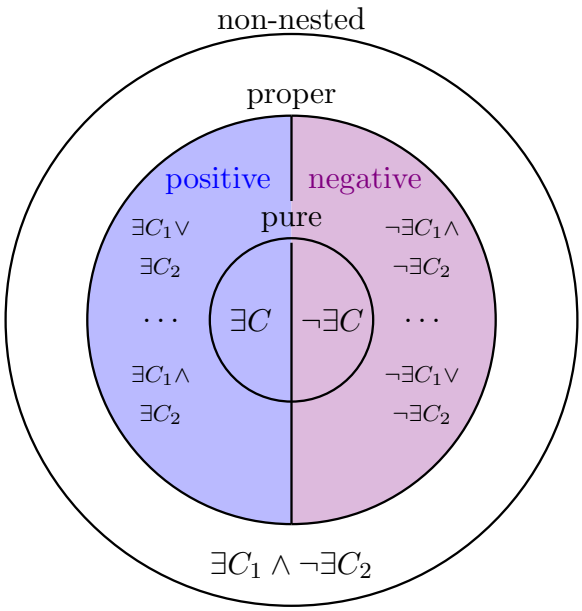


Figure 2.1: Non-nested constraints.

In Figure 2.1, the subclasses of non-nested constraints are illustrated: Proper constraints subsume positive (e.g., $\exists C_1 \vee \exists C_2$) and negative constraints (e.g., $\neg\exists C_1 \wedge \neg\exists C_2$). The non-nested constraint $\exists C_1 \wedge \neg\exists C_2$ is not proper. Pure constraints subsume pure positive ($\exists C$) and pure negative ($\neg\exists C$) constraints.

Example 2.3. Consider a single node label and three edge labels c_0, c_1, and c_2. The expressions

$$\text{Loop}(c_0) = \exists\left(\begin{array}{c} c_0 \\ \circlearrowright \end{array}\right) \quad \text{and} \quad \text{NoLoop}(c_0) = \neg\exists\left(\begin{array}{c} c_0 \\ \circlearrowright \end{array}\right)$$

are pure constraints. The pure positive constraint $\text{Loop}(c_0)$ expresses that there exists a c_0-labeled loop. The pure negative constraint $\text{NoLoop}(c_0)$ expresses that there does not exist a c_0-labeled loop. The expressions

$$\text{Loop} = \bigvee_{i=0,1,2} \exists\left(\begin{array}{c} c_i \\ \circlearrowright \end{array}\right) \quad \text{and} \quad \text{NoLoop} = \bigwedge_{i=0,1,2} \neg\exists\left(\begin{array}{c} c_i \\ \circlearrowright \end{array}\right)$$

are proper constraints. The positive constraint Loop expresses that there exists a loop. The negative constraint NoLoop expresses that there does not exist a loop.

[2] In Sandmann [San22], another type of proper constraints is defined.

The validity of positive contraints is inherited to "larger" graphs. Analogously, the validity of negative contraints is inherited to "smaller" graphs.

Fact 2.1 (upward & downward inheritance). Let $G \hookrightarrow H$.

(a) For a positive constraint c, $G \models c$ implies $H \models c$.

(b) For a negative constraint c, $H \models c$ implies $G \models c$.

Proof. We show (a) by structural induction over positive constraints. Let $h : G \hookrightarrow H$ be a total, injective morphism and c a positive constraint.
Hypothesis. $G \models c$ implies $H \models c$.
Base case. Let $c = \exists C$ and $G \models c$, i.e., there exists $g : C \hookrightarrow G$. Then, $h \circ g : C \hookrightarrow H$ is a total, injective morphism, i.e., $H \models \exists C$.
Induction Step. (i) Let $c = c_1 \vee c_2$ and $G \models c$, i.e., $G \models c_i$ where $i = 0$ or $i = 1$. We apply the induction hypothesis to c_i. We obtain $H \models c_i$ where $i = 1$ or $i = 2$. Thus, $H \models c$. (ii) Let $c = c_1 \wedge c_2$ and and $G \models c$, i.e., $G \models c_1$ and $G \models c_2$. We apply the induction hypothesis to both c_1 and c_2. We obtain $H \models c_1$ and $H \models c_2$. Thus, $H \models c$.
Statement (b) follows by the contraposition of statement (a). □

For every positive constraint, an equivalent constraint in "reduced $\vee$-normal form" is computable. A disjunction of pure positive constraints is *reduced* if for different graphs C, C' in the disjunction, there is no total, injective morphism from C to C'.

Lemma 2.1 ($\vee$-normal form). For every positive contraint, we can effectively construct an equivalent, reduced disjunction of pure positive constraints.

Proof sketch. The conjunction of two pure positive constraints can be expressed as an equivalent disjunction of pure positive constraints. This is done by considering all possible overlappings of two graphs. Thus, every positive constraint can be written in a disjunctive normal form. Formally, we perform a structural induction over positive constraints, see Appendix B. □

In graph transformation, the most popular approaches are the double-pushout approach and the single-pushout approach. We use the single-pushout approach with injective "matches" for modeling graph transformations, see, e.g., Löwe [Löw93], Ehrig et al. [EHK+97].

A rule is given by a graph morphism. It is applied to a graph by means of a set-theoretical or "pushout" construction. In a set-theoretical manner, the result of the application of the rule $r = \langle p : L \rightharpoonup R \rangle$ according to the "match" $m : L \hookrightarrow G$ is constructed from the graph G as follows:

(1) Delete all elements which are undefined under p.

(2) Add all elements which are new in R.

The construction is illustrated in Figure 2.2.

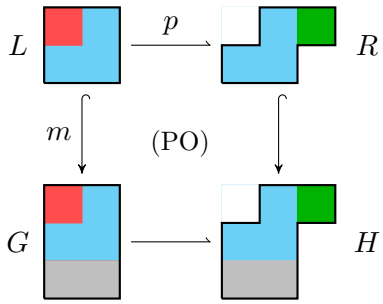


Figure 2.2: Illustration of a pushout.

The red area is undefined under p and therefore deleted. The green area is added. The light-blue area is preserved. The gray area illustrates the context of the left-hand side.

Definition 2.4 (rules & transformations). A *(graph transformation) rule* $r = \langle p : L \rightharpoonup R \rangle$ is a partial morphism from a graph L to a graph R.[3] A *(direct) transformation* $G \Rightarrow H$ from a graph G to a graph H via (application of) the rule r according to a total, injective *match morphism* $m : L \hookrightarrow G$ is given by a *pushout* of r and m as shown below. For a formal definition of pushouts, see Appendix A.

$$\begin{array}{ccc} L & \xrightarrow{p} & R \\ {\scriptstyle m}\downarrow & (\mathrm{PO}) & \downarrow \\ G & \longrightarrow & H \end{array}$$

We write $G \Rightarrow_r H$ to indicate the applied rule, and $G \Rightarrow_{\mathcal{R}} H$ if $G \Rightarrow_r H$ for a rule r in a rule set $\mathcal{R}$.
A *transformation sequence* is a (possibly infinite) sequence $G_0 \Rightarrow G_1 \Rightarrow \ldots$ of consecutive transformations. A transformation sequence (from G_0 to G_k) *of length* k is a finite transformation sequence $G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_k$. We denote a transformation sequence from G to H of length k by $G \Rightarrow^k H$, of length ≥ 0 by $G \Rightarrow^* H$, of length ≥ 1 by $G \Rightarrow^+ H$, and of length $\leq k$ by $G \Rightarrow^{\leq k} H$.

Example 2.4. In Figure 2.3, the rule on the top is applied to the graph in the bottom left corner. The construction of the pushout yields the graph in the bottom right corner. This transformation deletes (creates) one c_0-labeled loop at the source (target) node of the P_1-labeled edge.

[3] We use rules with names for referring to them.

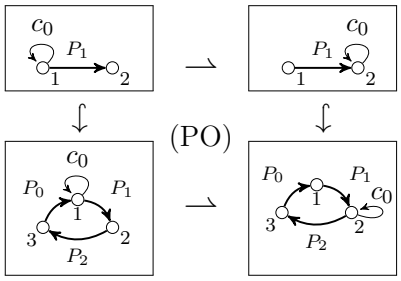


Figure 2.3: Example of a transformation.

Definition 2.5 (graph transformation system). A *graph transformation system (GTS)* is a finite set of rules.

Example 2.5. The `Initiate`-rule and the two `Forward`-rules form a GTS. The application of the `Initiate`-rule creates one c_0-labeled loop at the target node of a P_0-labeled edge. The application of the `Forward`-rules delete (create) one c_0-labeled loop at the source (target) node of a P_i-labeled edge where $i = 1, 2$.

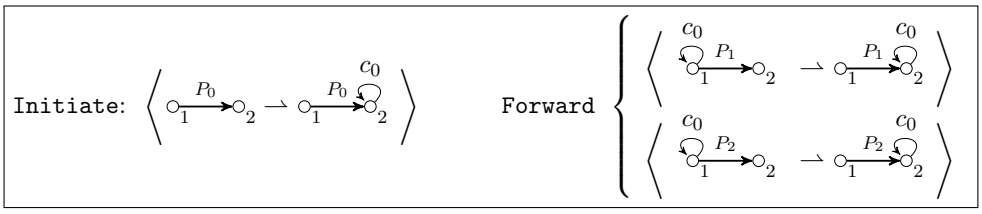


Figure 2.4: A GTS consisting of three rules.

Example 2.6 (reset Petri nets). Petri nets and reset Petri nets can be seen as graph transformation systems.[4] There are several possibilities for the encoding. Kreowski [Kre80] shows that Petri nets can be described by graph transformation in a very intuitive way. Baldan et al. [BCGM10] compare different variations of generalized Petri nets regarding their encodings.

2.2 Well-structured Transition Systems

Transition systems are an abstract model for describing transitions between the states of a system. The concept of well-structuredness (e.g., Abdulla et al. [AČJT96], Finkel & Schnoebelen [FS01], König & Stückrath [KS17]) is built upon the notion of transition systems.

Definition 2.6 (transition system). A *transition system* $\langle S, \rightarrow \rangle$ consists of a (possibly infinite) set S of *states* and a *transition relation* $\rightarrow \subseteq S \times S$.

[4] For a formal definition of reset Petri nets, see Appendix A.

Let $\to^0 = \mathrm{Id}_S$ (identity on S), $\to^1 = \to$, and $\to^k = \to^{k-1} \circ \to$ for every $k \geq 2$. Let $\to^{\leq k} = \bigcup_{0 \leq j \leq k} \to^j$ for every $k \geq 0$. The *transitive closure* is given by $\to^* = \bigcup_{k \geq 0} \to^k$.

A graph class S is *closed under (rule application of)* a GTS $\mathcal{R}$ if for every transformation $G \Rightarrow_{\mathcal{R}} H$ with $G \in S$, also $H \in S$.

Fact 2.2 (induced transition system). Every GTS $\mathcal{R}$ together with a graph class S closed under $\mathcal{R}$, induces the (graph) transition system $\langle S, \Rightarrow_{\mathcal{R}} \rangle$. Every reset Petri net together with the set of all markings and the firing as transition relation induces a (net) transition system.

The graph class is usually infinite as in the following example.

Example 2.7. Consider the graph class which consists of "cyclic" graphs with additional loops. The basic structure of these graphs is given by a cycle of three nodes connected by three edges labeled with P_0, P_1, and P_2 (the graph on the top left in Figure 2.5). Every loop is labeled with c_0. This graph class is closed under application of the `Initiate`-rule and `Forward`-rules shown in Figure 2.4. These rules together with the depicted graph class induce a transition system. An excerpt of the transition is shown in Figure 2.5.

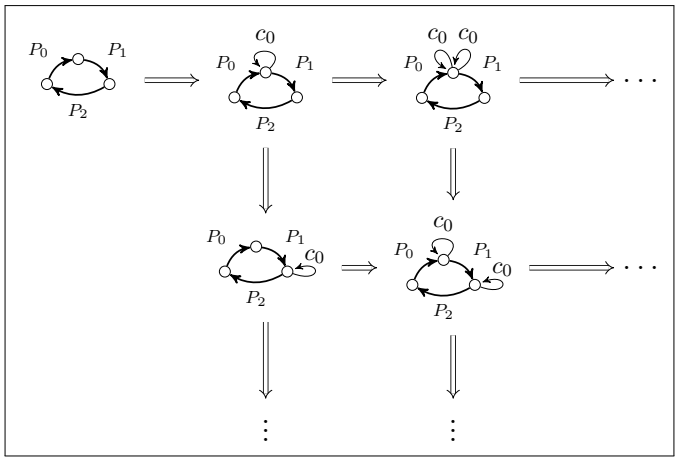


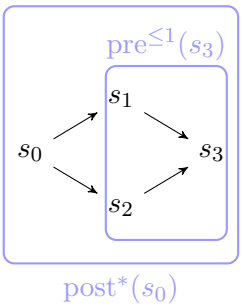
Figure 2.5: Transition system induced by a GTS and a graph class.

Often we are interested in the predecessors or successors of a set of states.

Definition 2.7 (pre- & postsets). Let $\langle S, \to \rangle$ be a transition system. For $S' \subseteq S$ and $k \geq 0$, we define $\mathrm{pre}^k(S') = \{s \in S \mid \exists s' \in S' : s \to^k s'\}$ and $\mathrm{post}^k(S') = \{s \in S \mid \exists s' \in S' : s' \to^k s\}$. Let $\mathrm{pre}^{\leq k}(S') = \bigcup_{j \leq k} \mathrm{pre}^j(S')$, $\mathrm{pre}^*(S') = \bigcup_{k \geq 0} \mathrm{pre}^k(S')$, $\mathrm{post}^{\leq k}(S') = \bigcup_{j \leq k} \mathrm{post}^j(S')$, and $\mathrm{post}^*(S') = \bigcup_{k \geq 0} \mathrm{post}^k(S')$. We abbreviate $\mathrm{post}^1(S')$ by $\mathrm{post}(S')$ and $\mathrm{pre}^1(S')$ by $\mathrm{pre}(S')$.

For a singleton $S' = \{s\}$, we omit the braces. A transition system $\langle S, \to \rangle$ is *finite-branching* if $\mathrm{post}(s)$ is finite and computable for every given state s.

Example 2.8. An example of a transition system consisting of four states s_0, s_1, s_2, s_3 and transitions $s_0 \to s_1, s_0 \to s_2, s_1 \to s_3, s_2 \to s_3$ is given below. The set $\mathrm{post}^*(s_0)$ of all states reachable from s_0 contains all four states. The set $\mathrm{pre}^{\leq 1}(s_3)$ of states from which s_3 can be reached in at most one step contains only s_1, s_2, s_3.



Pre- and postsets are "dual" in the sense that for a predecessor s of a state s', the state s' is a successor of s, and vice versa.

Fact 2.3 (pre- & postsets). Let $\langle S, \to \rangle$ be a transition system and $s, s' \in S$. For every $k \geq 0$,

$$s \in \mathrm{pre}^k(s') \iff s' \in \mathrm{post}^k(s).$$

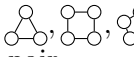
Several problems are undecidable for transition systems, in general, due to their infinite state space. However, interesting decidability results can be achieved if the system is well-structured, see, e.g., Abdulla et al. [AČJT96], Finkel & Schnoebelen, [FS01]. A prerequisite for this concept is a well-quasi-order on the set of states.

Definition 2.8 (well-quasi-order). A *quasi-order* is a reflexive, transitive relation. A *well-quasi-order (wqo)* over a set S is a quasi-order $\leq \subseteq S \times S$ s.t. every infinite sequence $\langle s_0, s_1, \ldots \rangle$ in S contains an increasing pair $s_i \leq s_j$ with $i < j$. A (well-)quasi-order is *decidable* if it can be decided whether $s \leq s'$ for every $s, s' \in S$.

In our setting, the subgraph order is of crucial importance.

Definition 2.9 (subgraph order). The subgraph order $\leq$ is given by $G \leq H$ iff there exists a total injective morphism $G \hookrightarrow H$ for graphs G, H.

The subgraph order is a quasi-order on all graphs. However, it is not a wqo on all graphs.

Example 2.9 (no increasing pair). The infinite sequence  , ... of cyclic graphs of increasing length contains no increasing pair.

To obtain a well-quasi-order, we restrict the graph class.

Definition 2.10 (bounded path length). A graph class S is *of bounded path length* if the length of any *path*[5] in any graph $G \in S$ is bounded.

König & Stückrath show that the subgraph order is a well-quasi-order on graphs of bounded path length, based on a result of Ding.

Fact 2.4 (subgraph order [KS17, Prop. 6], [Din92, Thm. 2.1]). For a graph class S of bounded path length, the restriction $\leq_S$ of the subgraph order $\leq$ to S is a wqo.

Upward- and downward-closed sets w.r.t. a given well-quasi-order are of special interest. Such sets are called ideals and anti-ideals. Ideals are, in general, infinite but can be represented by a finite basis (a minimal generating set), similar to algebraic structures.

Definition 2.11 (ideal & basis). Let S be a set and $\leq$ a quasi-order on S. For every subset A of S, we denote by $\uparrow A = \{s \in S \mid \exists a \in A : a \leq s\}$ the *upward-closure* and by $\downarrow A = \{s \in S \mid \exists a \in A : s \leq a\}$ the *downward-closure* of A. An *ideal* $I \subseteq S$ is an upward-closed set, i.e., $\uparrow I = I$. An *anti-ideal* $J \subseteq S$ is a downward-closed set, i.e., $\downarrow J = J$. An (anti-)ideal is *decidable* if membership is decidable. A *basis* of an ideal I is a subset $\mathcal{B} \subseteq I$ s.t. (i) $\uparrow\mathcal{B} = I$ and (ii) $b \neq b' \Rightarrow b \not\leq b'$ for all $b, b' \in \mathcal{B}$.

Since a downward-closed set does not have a "top basis" in general, we will demand that membership is decidable.

Convention. The symbols "$\uparrow$" and "$\downarrow$" are binding stronger than "$\cap$" and "$\cup$", e.g., $\uparrow A \cap B$ is the intersection of the upward-closure of the set A with the set B.

The upward- and downward-closure of a set A with three elements are visualized in Figure 2.6. The nearer to the bottom an element, the "smaller" the element. All elements "larger" ("smaller") than an element are located in the "cone" above (below) that element.

An ideal and an anti-ideal (its complement) are visualized in Figure 2.7. The basis of the ideal is given by the three elements represented by blue nodes.

[5] A *path* in a graph G of length ℓ is a sequence $v_1, e_1, v_2 \ldots, v_\ell, e_\ell, v_{\ell+1}$ of nodes and edges s.t. $\mathrm{src}_G(e_i) = v_i$ and $\mathrm{tgt}_G(e_i) = v_{i+1}$, or $\mathrm{tgt}_G(e_i) = v_i$ and $\mathrm{src}_G(e_i) = v_{i+1}$ for every $1 \leq i \leq \ell$, and all contained nodes and edges occur at most once.

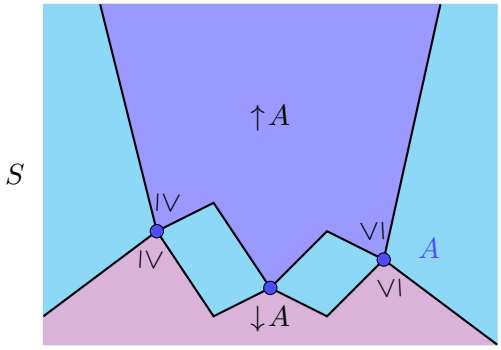


Figure 2.6: Visualization of upward- and downward-closure of a finite set A.

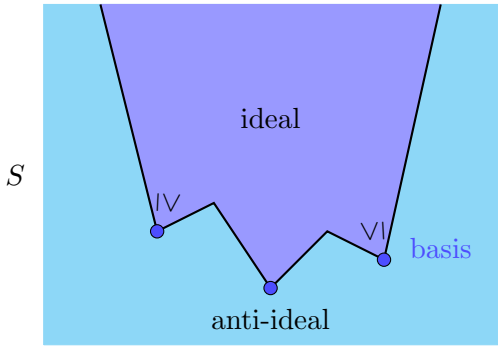


Figure 2.7: Visualization of an (anti-)ideal.

Regarding the subgraph order, positive and negative constraints constitute ideals and anti-ideals, respectively. For a graph class S, $[\![c]\!]_S$ denotes the class of all graphs in S satisfying c.

Fact 2.5 (ideals of graphs). Let S be a graph class. For every positive (negative) constraint c, the set $[\![c]\!]_S$ is an ideal (anti-ideal).

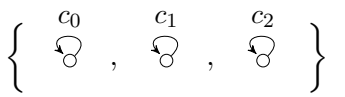
Proof. Let c be a positive constraint and $G \in S$ a graph with $G \models c$. By Fact 2.1, for every graph $H \in S$ with $G \leq H$, also $H \models c$. Thus, $[\![c]\!]_S$ is an ideal. The statement for negative constraints follows analogously. □

Example 2.10. Let S be the class of all graphs over the node label □ and edge labels c_1, c_2, c_3. For the proper constraints

$$\text{Command} = \bigvee_{i=0,1,2} \exists\left(\begin{array}{c} c_i \\ \circlearrowright \end{array} \right) \quad \text{and} \quad \text{NoCommand} = \bigwedge_{i=0,1,2} \neg\exists\left(\begin{array}{c} c_i \\ \circlearrowright \end{array} \right),$$

the set $[\![\text{Command}]\!]_S$ is an ideal consisting of all graphs in S with a c_0-, c_1-, or c_2-labeled loop and the set $[\![\text{NoCommand}]\!]_S$ is an anti-ideal consisting of

all graphs in S without c_0-, c_1-, and c_2-labeled loops. The set



is a basis of the ideal $[\![\text{Command}]\!]_S$.

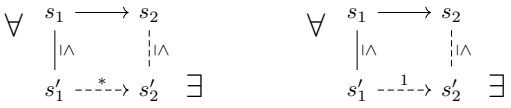
For a well-quasi-order, every ideal can be represented by a finite basis.

Fact 2.6 (finite basis [AČJT96, Lemma 3.3]). Every ideal has a basis and every basis is finite, provided that the superset is equipped with a wqo. Given a finite set A, we can compute a basis of $\uparrow A$, provided that the quasi-order is decidable.

For well-structuredness, we demand that the well-quasi-order yields a simulation of "smaller" states by "larger" states. This condition is called compatibility.

Definition 2.12 (well-structured transition systems). Let $\langle S, \to\rangle$ be a transition system and $\leq$ a decidable wqo on S, i.e., for each two given states $s, s' \in S$, it is decidable whether $s \leq s'$. The tuple $\langle S, \leq, \to\rangle$ is a *well-structured transition system (WSTS)*, if:

(i) The wqo is *compatible* with the transition relation, i.e., for all $s_1, s_1', s_2 \in S$ with $s_1 \leq s_1'$ and $s_1 \to s_2$, there exists $s_2' \in S$ with $s_2 \leq s_2'$ and $s_1' \to^* s_2'$. If $s_1' \to^1 s_2'$, we say that it is *strongly compatible*.



(a) Compatibility (b) Strong compatibility

(ii) Pre-*effectiveness*: For every $s \in S$, a basis of $\uparrow\text{pre}(\uparrow\{s\})$ is computable.

A *strongly WSTS (SWSTS)* is a WSTS with strong compatibility.

Fact 2.7. Every SWSTS is a WSTS.

Remark. For the single-pushout approach (SPO), the strong compatibility condition in Definition 2.12 is satisfied. If a rule can be applied to a graph G yielding a graph H, it can also be applied to every graph $G' \geq G$ yielding a graph $H' \geq H$.

$$\begin{array}{ccc} G & \Longrightarrow_r & H \\ \restriction\!\wedge & & \restriction\!\wedge \\ G' & \Longrightarrow_r & H' \end{array}$$

For the double-pushout (DPO) approach, the compatibility condition is, in general, not satisfied. The application of the same rule to a "larger" graph is not possible, in general. Consider, e.g., the rule $\langle \circ \rightharpoonup \varnothing \rangle$ which deletes a node. In DPO, this rule is applicable to the graph consisting of one node, but not to the "larger" graph consisting of two nodes connected by an edge. The *dangling condition* in DPO prohibits a rule application if incident edges of a node to be deleted are not specified to be deleted. In [Par92], Parisi-Presicce provides a comparison of DPO and SPO.

Assumption. In the following, let $\langle S, \leq, \rightarrow \rangle$ be a well-structured transition system.

The set of ideals of S is closed under preset, union, and intersection.

Fact 2.8 (stability of ideals [AČJT96, Lemma 3.2]). Let $I, I' \subseteq S$ be ideals and $J, J' \subseteq S$ anti-ideals. Then the sets $\mathrm{pre}^*(I)$, $I \cup I'$, and $I \cap I'$ are ideals and the sets $J \cup J'$, $J \cap J'$ are anti-ideals. For SWSTSs, also the sets $\mathrm{pre}(I)$, $\mathrm{pre}^{\leq k}(I)$ for every $k \geq 0$ are ideals.

Remark. The upward-closedness of $\mathrm{pre}^*(I)$ for WSTSs, where I is an ideal, is shown, e.g., in [FS01, Proof of Prop. 3.1].

A major point in our argumentation is the observation that every infinite, ascending sequence of ideals w.r.t. a well-quasi-order eventually becomes stationary.

Lemma 2.2 (termination [AČJT96, Lemma 3.4]). For every infinite, ascending sequence $I_0 \subseteq I_1 \subseteq \ldots$ of ideals, there exists a $k_0 \geq 0$ s.t. $I_k = I_{k_0}$ for all $k \geq k_0$.

The statement of the latter lemma is indeed equivalent to the definition of a well-quasi-order (Definition 2.8) as well as to the existence of bases (Fact 2.6).

Fact 2.9 ([Mil85, Thm. 1.2]). For a quasi-order, the following statements are equivalent:

(1) The quasi-order is a wqo.

(2) Every ideal has a basis.

(3) Every infinite, ascending sequence of ideals becomes stationary.

Abdulla et al. exploit Lemma 2.2 to show the decidability of the ideal reachability problem for SWSTSs.

Lemma 2.3 (ideal reachability [AČJT96, Thm. 4.1]). Given a basis of an ideal $I \subseteq S$, and a state s of a SWSTS, we can decide whether we can reach a state $s_I \in I$ from s. In particular, a basis of $\mathrm{pre}^*(I)$ is computable and $\mathrm{pre}^{\leq k}(I) = \mathrm{pre}^*(I) \iff \mathrm{pre}^{\leq k+1}(I) = \mathrm{pre}^{\leq k}(I)$.

The proof idea as well as the decidability of the ideal reachability problem will be used in Chapter 4 and 5 as an important integrant for the decidability proofs. For this reason, we present a sktech of the proof.

Proof sketch. We sketch the proof of Abdulla et al. For any ideal I, another ideal I^* is constructed, s.t. $\exists s' \in I : s \rightarrow^* s'$ iff $s \in I^*$, i.e., $I^* = \mathrm{pre}^*(I) = \bigcup_{j \geq 0} \mathrm{pre}^j(I)$. The idea is to iteratively construct the sequence of the ideals $I_k = \mathrm{pre}^{\leq k}(I)$ until it becomes stable. It is shown that $I_k = I_{k+1}$ is a decidable stop condition, i.e., a condition which guarantees that we can stop the algorithm, since $I_{k+1} = I \cup \mathrm{pre}(I_k)$. Since ideals are infinite, this construction cannot be carried out directly but using bases for representing ideals. By the definition of SWSTSs, we can iteratively compute bases of the predecessors of an ideal with a given basis. These considerations are similarly feasible for WSTSs. Instead of the preset $\mathrm{pre}(I')$ of an ideal I', one can consider the ideal $\uparrow \mathrm{pre}(I')$, see, e.g., Finkel & Schnoebelen, [FS01, Proof of Thm. 3.6]. □

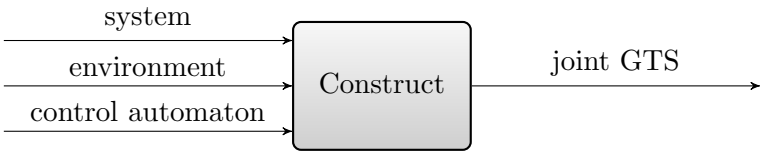
Chapter 3

Modeling Adverse Conditions

The concepts of resilience (e.g., Jackson & Ferris [JF13]) and adverse conditions (e.g., Olderog et al. [OFTK21]) are topics of recent research. Correctness under adverse conditions captures a variety of areas. To the best of our knowledge, there is little research on adverse conditions modeled by graph transformation, e.g., [Fli16, Peu18].

The concept of adverse conditions addresses systems interacting with an adversary environment. System correctness under adverse conditions means that the system withstands the adverse effect of the environment. It generalizes common notions of correctness. Models of adverse conditions include, e.g., graph transformational units (Hölscher et al. [HKK09]), adverse reconfiguration and 2-player programs (Flick [Fli16]), and 2-player hyperedge replacement games (Peuser [Peu18]).

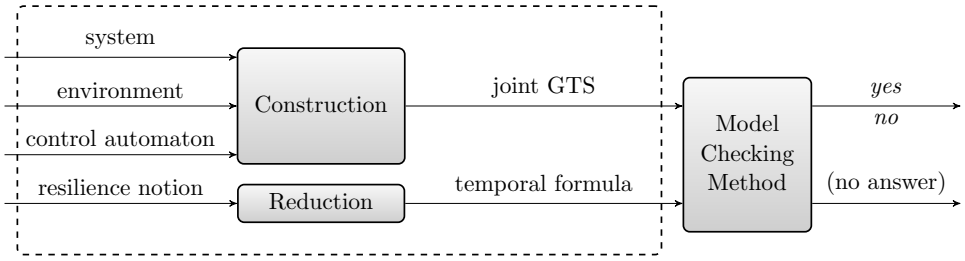
In this chapter, we introduce the model of so-called joint graph transformation systems each of which arises from the interplay of a system and an environment. Their interaction is modeled by a so-called control automaton.



The constructed joint system is again a graph transformation system s.t. graph-based methods and tools can be directly applied. We introduce meaningful notions of resilience for this kind of systems. These notions capture the recovery of a safety constraint Safe after an interference of the environment Env, i.e.,

$$\text{Interference of Env} \Rightarrow \text{Recovery of Safe.}$$

In this approach, we reduce verification of resilience to temporal model checking.



For the constructed joint graph transformation system, we try to check whether the temporal formula expressing a given resilience notion holds. To this aim, one may use a graph-based model checker.

In Section 3.1, we introduce the model of joint graph transformation systems. In, Section 3.2, we define meaningful resilience notions for this kind of systems. In Section 3.3, we show how these resilience notions can be expressed in temporal logics as a preparation for model checking. An introduction to temporal logics can be found in Appendix A.

3.1 Joint Graph Transformation Systems

We define so-called joint graph transformation systems, each of which involves the system, the environment, and a so-called control automaton modeling the interaction between them. Both, system and environment, are graph transformation systems.

Example 3.1 (traffic network system). We model a traffic network system where the adverse conditions are blocked connections due to jams, accidents, or damages on the tracks. Nodes correspond to traffic junctions while an edge of the form ○═○ represents a connection between them. An edge labeled with the symbol 🚗 describes a vehicle in the traffic line between the traffic junctions. The rule `Ascend` (`Descend`) formalizes the ascent (descent) of a vehicle onto (off) the traffic network. We realize a traffic flow by the rule `Move`, which enables vehicles to change their position to an adjacent track. Blocked tracks (○═⚡○) occur only if a connection is highly frequented, i.e., if n vehicles are on the same track. This is formalized by the environment rule `Block`. For the sake of simplicity, let $n = 2$. Blocked tracks can be repaired by the rule `Repair`. Formally, we consider the system $\mathcal{S}$ and environment $\mathcal{E}$ represented in Figure 3.1.

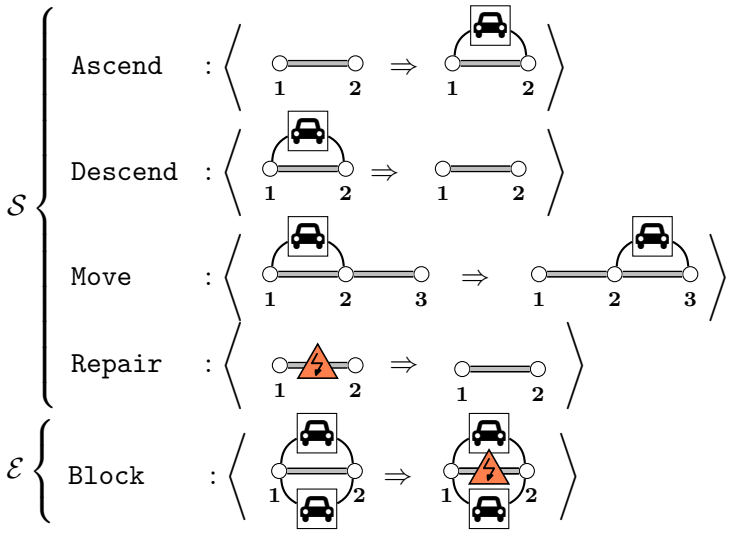


Figure 3.1: System and environment rules of the traffic nework system.

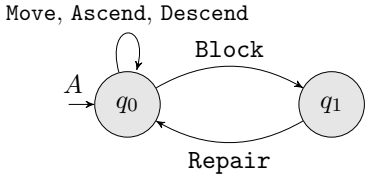
Assumption. In the following, let $\mathcal{S}$ and $\mathcal{E}$ be GTSs over a joint label alphabet Λ, called *system* and *environment*, respectively.

We specify the class of automata which are used to model the interaction between system and environment. These "control automata" are similar to ω-automata, see, e.g., Thomas [Tho90]. Each of the former consists of a finite set of states, an initial state, and a labeled transition relation. Labeling the transitions with *system* or *environment* would allow us to determine an "order" according to which system and environment interact. For more precise modeling, the transitions are labeled with a subset of rules which are admissible for the respective transition in the automaton. Equivalently, a transition can be labeled with a single rule. However, the representation of the automaton is better comprehensible with subsets of rules.

Definition 3.1 (control automaton). A *control automaton* of $\mathcal{S}$ and $\mathcal{E}$ is a tuple $A = \langle Q, \delta, q_0 \rangle$ consisting of a finite set Q disjoint from Λ, called the *state set*, an *initial state* $q_0 \in Q$, and a *(rule-labeled) transition relation* $\delta \subseteq Q \times \mathfrak{P}(\mathcal{S} \cup \mathcal{E}) \times Q$ where $\mathfrak{P}(\mathcal{S} \cup \mathcal{E})$ is the power set of $\mathcal{S} \cup \mathcal{E}$. For $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{E}$, we write $q \to_{\mathcal{R}} q'$ for $\langle q, \mathcal{R}, q' \rangle \in \delta$ and $q \to_r q'$ for a rule $r \in \mathcal{R}$ with $q \to_{\mathcal{R}} q'$.

Convention. In a graphical representation, a q-labeled node stands for a state q and a $\mathcal{R}$-labeled edge from q to q' stands for a transition $q \to_{\mathcal{R}} q'$. For a set $\mathcal{R} = \{r_1, \ldots, r_n\}$, we allow to omit the brackets.

Example 3.2. A control automaton modeling the interaction between system and environment in Example 3.1 is given below.



We assume that for given a rule $r \in \mathcal{S} \cup \mathcal{E}$, we can infer the information whether r is meant as a system or an environment rule. Formally, this can be done by considering the disjoint union of $\mathcal{S}$ and $\mathcal{E}$ in the set-theoretic sense.

For the synchronization on the level of transition systems, one may use tuples $\langle G, q\rangle$ of graphs and states and transitions $\langle G, q\rangle \to \langle H, q'\rangle$ if $G \Rightarrow_r H$ for some rule $r \in \mathcal{S} \cup \mathcal{E}$ and $q \to_r q'$ for some $q, q' \in Q$. For the synchronization on the level of graph transformation systems, we use graphs G^q, i.e., G equipped with a q-labeled node, (instead of tuples $\langle G, q\rangle$) and rules $\langle L^q \rightharpoonup R^{q'}\rangle$ for some rule $r = \langle L \rightharpoonup R\rangle \in \mathcal{S} \cup \mathcal{E}$ and $q \to_r q'$ for some $q, q' \in Q$. This yields transformations $G^q \Rightarrow H^{q'}$ if $G \Rightarrow_r H$ for some rule $r \in \mathcal{S} \cup \mathcal{E}$ and $q \to_r q'$ for some $q, q' \in Q$.

A so-called joint graph transformation system is obtained by synchronizing the system, repectively, the environment, with the control automaton, and then joining both sets of "enriched" rules.

Construction 3.1 (joint graph transformation system). Let be $A = \langle Q, \delta, q_0\rangle$ a control automaton of $\mathcal{S}$ and $\mathcal{E}$. The *joint graph transformation system* of $\mathcal{S}$ and $\mathcal{E}$ w.r.t. A is $\mathcal{S}_A \cup \mathcal{E}_A$, shortly $\mathcal{SE}$, where the *enriched rule set* $\mathcal{S}_A$ is constructed as

$$\mathcal{S}_A = \{\langle L^q \rightharpoonup R^{q'}\rangle \mid r = \langle L \rightharpoonup R\rangle \in \mathcal{S} \text{ and } q \to_r q'\},$$

and L^q ($R^{q'}$) denotes the disjoint union of L (R) and a node labeled with q (q'). In the partial morphism $L^q \rightharpoonup R^{q'}$, the node labeled with q is not in the domain, i.e., deleted, and the node labeled with q' is not in the image, i.e., created. The enriched rule set $\mathcal{E}_A$ is defined analogously.

We explicate the graph classes for joint GTSs. These graphs are of the form G^q for a state q of the control automaton.

Example 3.3. The traffic network in Example 3.1 synchronized with the control automaton A in Example 3.2 yields the joint GTS in Figure 3.2.

We consider "annotated" joint graph transformation systems which also carry the information whether the last applied rule was a system or environment rule. This is realized by a node labeled with "**s**" ("the last applied rule was a system rule") or "**e**" ("the last applied rule was an environment rule"). A node labeled with "$\top$" indicates an INITial graph. This annotation will be used for the reduction to temporal logics.

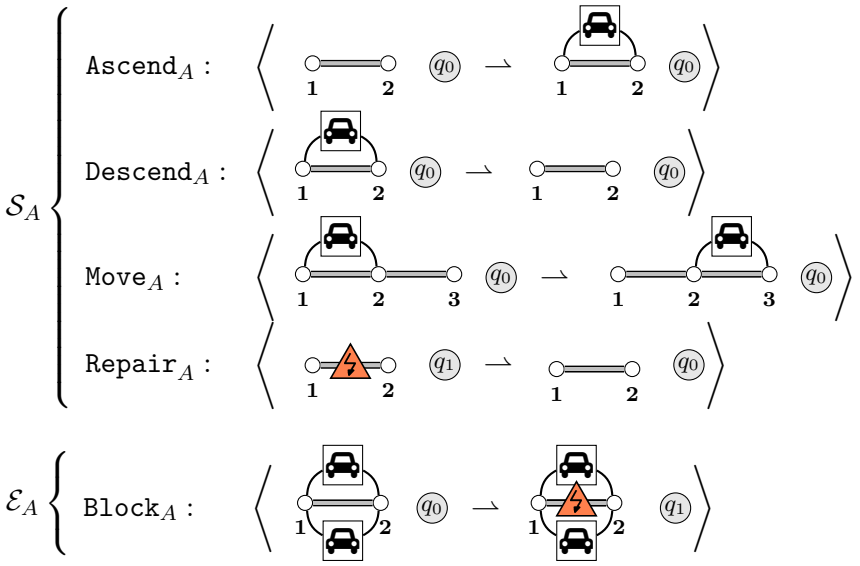


Figure 3.2: The joint GTS of the traffic network.

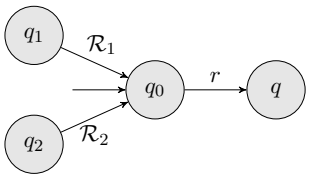
To reduce the number of rules, we use "presymbols" of a state q describing which of the symbols $\mathbf{s}, \mathbf{e}, \top$ can occur before the application of a rule. For a joint GTS $\mathcal{SE}$, the symbol $\sigma(\mathcal{S}) = \mathbf{s}$ or $\sigma(\mathcal{E}) = \mathbf{e}$, is the *annotation symbol* of $\mathcal{S}$ or $\mathcal{E}$, respectively. For a rule $r \in \mathcal{R}$ and $\mathcal{R} \in \{\mathcal{S}, \mathcal{E}\}$, let $\sigma(r) = \sigma(\mathcal{R})$ be the annotation symbol of r. The set of all annotation symbols $\{\mathbf{s}, \mathbf{e}, \top\}$ includes also the symbol $\top$, indicating an INITial graph.

Construction 3.2 (presymbols). The set of *presymbols* presym(q) of a state q of the control automaton is constructed as follows.

(1) For every transition of the control automaton going to the state q, consider its label, i.e., the subset of rules.

(2) For each such rule, include its annotation symbol into the set of presymbols of q, i.e., if a system (environment) rule is contained, include the symbol $\mathbf{s}$ ($\mathbf{e}$). If q is the initial state, include also the symbol $\top$.

Example 3.4. In the situation below, the presymbols of the state q_0 are the presymbols of $\mathcal{R}_1 \cup \mathcal{R}_2$. Since q_0 is the initial state, we also include the symbol $\top$. These presymbols are used to annotate each rule r with $q_0 \to_r q$.



In Example 3.2, the set of presymbols of q_0 of the control automaton consists of $\sigma(\texttt{Move}), \sigma(\texttt{Ascend}), \sigma(\texttt{Descend}), \sigma(\texttt{Repair}), \top$, i.e., the symbols $\mathbf{s}, \top$. The set of presymbols of q_1 consists of $\sigma(\texttt{Block})$), i.e., the symbol $\mathbf{e}$.

Fact 3.1. For every joint GTS, there is an "equivalent" annotated one.

For simplicity, one may replace presym(r) by the set of all annotation symbols in the following construction. However, the number of rules may be larger than in the case using presymbols presym(q).

Construction 3.3 (annotated joint graph transformation system). Let $\mathcal{SE}$ be a joint graph transformation systems w.r.t. a control automaton $A = \langle Q, \delta, q_0 \rangle$ of $\langle \mathcal{S}, \mathcal{E} \rangle$. The *annotated joint graph transformation system* of $\mathcal{S}$ and $\mathcal{E}$ w.r.t. A is $\mathcal{S}'_A \cup \mathcal{E}'_A$, shortly $(\mathcal{SE})'$, where the *annotated rule set* $\mathcal{S}'_A$ is constructed as

$$\mathcal{S}'_A = \{\langle L^{q\varsigma} \rightharpoonup R^{q'\varsigma'} \rangle \mid \langle L^q \rightharpoonup R^{q'} \rangle \in \mathcal{S}_A, \varsigma \in \text{presym}(q), \varsigma' = \mathbf{s}\},$$

where $L^{q\varsigma}$ ($R^{q'\varsigma'}$) denotes the disjoint union of L (R), a node labeled with a state q (q'), and a node labeled with an annotation symbol ς (ς'). In the partial morphism $L^{q\varsigma} \rightharpoonup R^{q'\varsigma'}$, the node labeled with ς is not in the domain, i.e., deleted, and the node labeled with ς' is not in the image, i.e., created. The annotated rule set $\mathcal{E}'_A$ is defined analogously, replacing $\mathcal{S}$ by $\mathcal{E}$ ($\mathbf{s}$ by $\mathbf{e}$).

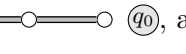
We explicate the graph classes for annotated joint GTSs. These graphs are of the form $G^{q\varsigma}$ for a state q of the control automaton and an annotation symbol ς.

Example 3.5. Consider the system $\mathcal{S}$, the environment $\mathcal{E}$, and the control automaton A given in Example 3.1. The annotated joint GTS $\mathcal{S}'_A \cup \mathcal{E}'_A$ is represented in Figure 3.3.

3.2 Resilience Notions

In this section, we give a notion of correctness for graph transformation systems and introduce three instances of resilience notions for joint graph transformation systems, which generalize correctness.

Definition 3.2 (marked joint GTS). A *marked GTS* is a tuple $\langle S, \mathcal{R}, \text{INIT} \rangle$ where $\mathcal{R}$ is a GTS, S is a set of graphs closed under $\mathcal{R}$, and INIT $\subseteq S$. A *marked joint GTS* is a marked GTS $\langle S, \mathcal{SE}, \text{INIT} \rangle$ where $\mathcal{SE}$ is an joint GTS and S is a set of graphs of the form G^q where q is a state of the control automaton. Marked annotated joint GTS are defined analogously.

Example 3.6. The tuple $\langle S, \mathcal{SE}, \text{INIT} \rangle$ is a marked joint GTS where $\mathcal{SE}$ is the joint GTS of the traffic network in Example 3.3, INIT consists the single graph 

, and S is the set of graphs reachable from the latter graph.

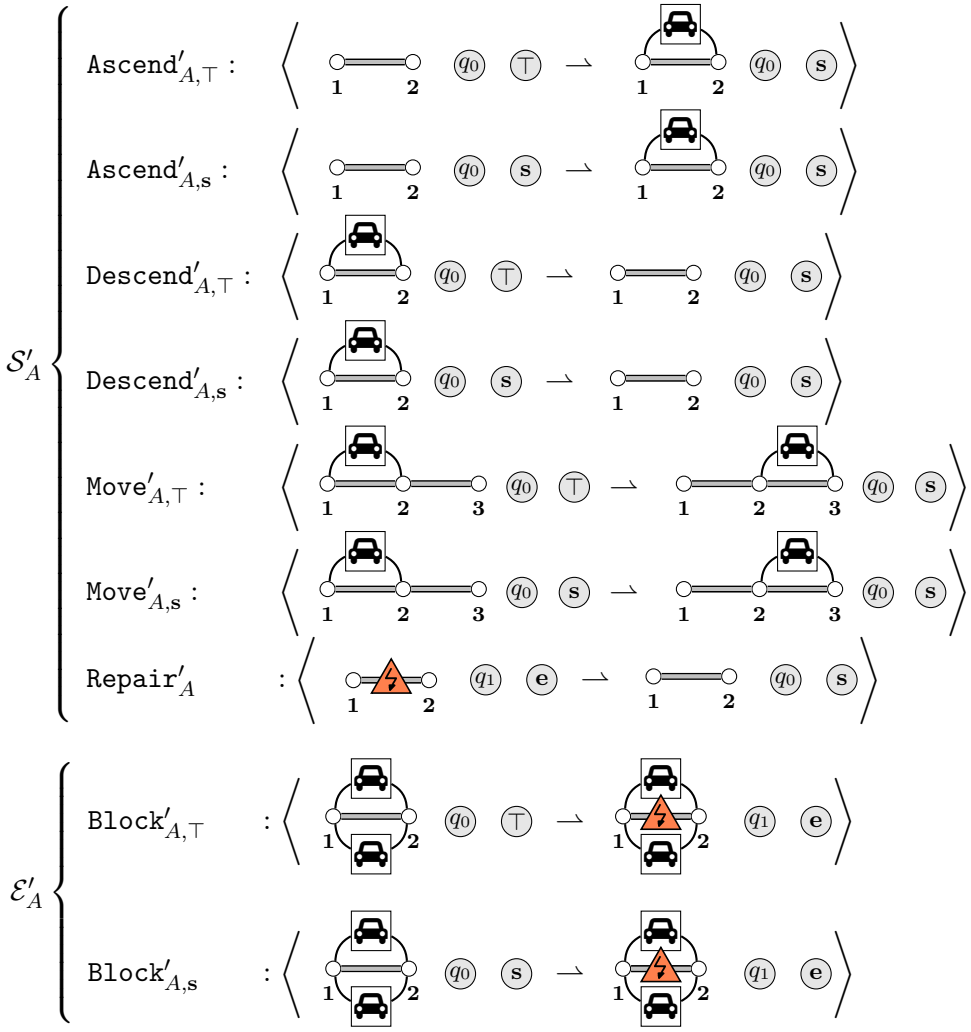


Figure 3.3: The annotated joint GTS of the traffic network.

Correctness of a marked GTS w.r.t. a *safety constraint* Safe means that every graph, reachable from an INITial graph, satisfies Safe. This notion corresponds to safety correctness in the usual sense, see, e.g., Apt et al. [AOdB09, p. 283].

Definition 3.3 (correctness). A marked GTS $\langle S, \mathcal{R}, \text{INIT} \rangle$ is *correct* w.r.t. a graph constraint Safe if for all graphs $G \in \text{INIT}$, $G \Rightarrow^*_{\mathcal{R}} H$ implies $H \models \text{Safe}$.

Example 3.7. The marked joint GTS of the traffic network (Example 3.6) is not correct w.r.t. Safe = NoBlocked = $\neg\exists($  $)$. A graph with a blocked track can be derived by applying `Block`.

For joint GTSs, this correctness notion is too restrictive. Instead of

correctness, we consider resilience. We allow that the validity of the safety constraint is violated for a certain number of steps after an interference of the environment. Nonetheless, the safety constraint must be recovered. We give instances of resilience notions. A natural approach is to limit the maximal number of steps after which the safety constraint must be recovered.

Assumption. In the following, let $\langle S, \mathcal{SE}, \text{INIT}\rangle$ be a marked joint GTS w.r.t. a control automaton $A = \langle Q, \delta, q_0\rangle$, Safe a graph constraint, and $k \geq 0$ a natural number.

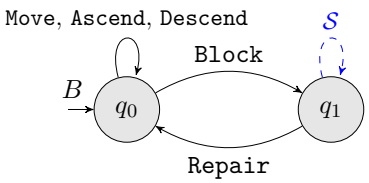
The notion of k-step$_\forall$ resilience demands that after an interference of the environment, in *every* following sequence, recovery of the safety constraint occurs in at most k steps. By contrast, the notion of k-step$_\exists$ resilience demands that after an interference of the environment, in *at least one* following sequence, recovery of the safety constraint occurs in at most k steps.

Definition 3.4 (k-step resilience). A marked joint GTS $\langle S, \mathcal{SE}, \text{INIT}\rangle$ is

(1) *k-step$_\forall$ resilient* w.r.t. Safe if, for every graph $G_0 \in \text{INIT}$ and every transformation sequence $G_0 \Rightarrow^*_{\mathcal{SE}} H \Rightarrow_{\mathcal{E}_A} M \Rightarrow^k_{\mathcal{SE}} N$, there exists a subsequence[6] $M \Rightarrow^{\leq k}_{\mathcal{SE}} N'$ with $N' \models \text{Safe}$, and

(2) *k-step$_\exists$* (or simply *k-step*) *resilient* w.r.t. Safe if, for every graph $G_0 \in \text{INIT}$ and every transformation sequence $G_0 \Rightarrow^*_{\mathcal{SE}} H \Rightarrow_{\mathcal{E}_A} M$, there exists a graph N s.t. $M \Rightarrow^{\leq k}_{\mathcal{SE}} N$ and $N \models \text{Safe}$.

Example 3.8. (1) For Safe = NoBlocked, the marked joint GTS of the traffic network system (Example 3.6) is neither correct w.r.t. Safe nor 0-step$_\forall$ resilient, but 1-step$_\forall$ resilient w.r.t. Safe since after applying `Block`, there is a blocked track and, by applying `Repair`, which is the only option in q_1, the blocked track vanishes.
(2) Consider the marked joint GTS obtained by replacing the control automaton A by B below.



In this case, it is 1-step$_\exists$ resilient w.r.t. Safe since `Repair` can be applied after an application of `Block` and before `Block` is applied again. By the proposition

[6] Our notation implies that the subsequence is connected.

below, it is k-step$_\exists$ resilient for $k \geq 1$. It is not k-step$_\forall$ resilient w.r.t. Safe for any k since, e.g., the system rule `Ascend` may be applied arbitrarily often after an application of `Block`.

k-step$_\forall$ resilience yields a hierarchy: k-step$_\forall$ resilience implies $(k+1)$-step$_\forall$ resilience for $k \geq 0$. A *deadlock* of a marked joint GTS $\langle S, \mathcal{SE}, \text{INIT}\rangle$ is a graph H reachable from a graph $G_0 \in \text{INIT}$ s.t. no rule in $\mathcal{SE}$ is applicable to H. For marked joint GTSs without deadlocks, k-step$_\exists$ resilience is a weaker notion than k-step$_\forall$ resilience. Similar to k-step$_\forall$ resilience, we obtain a hierarchy for k-step$_\exists$ resilience.

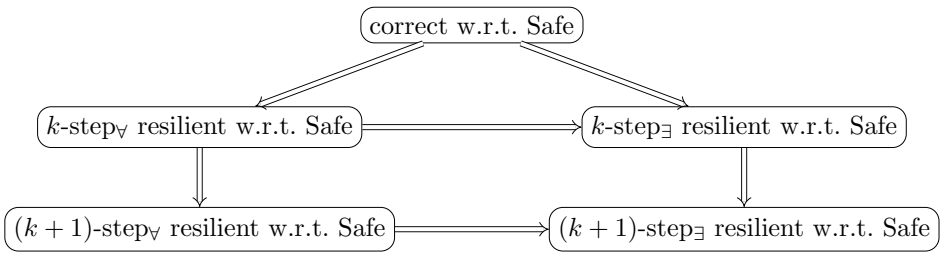
Proposition 3.1 (hierarchy of k-step resilience). Let $\langle S, \mathcal{SE}, \text{INIT}\rangle$ be a marked joint GTS.

(i) If the marked joint GTS is correct w.r.t. Safe,
then it is 0-step$_\forall$ resilient w.r.t. Safe.

(ii) If the marked joint GTS is k-step$_\forall$ resilient w.r.t. Safe,
then it is $(k+1)$-step$_\forall$ resilient w.r.t. Safe.

(iii) If the marked joint GTS has no deadlocks and
is k-step$_\forall$ resilient w.r.t. Safe,
then it is k-step$_\exists$ resilient w.r.t. Safe.

(iv) If the marked joint GTS is k-step$_\exists$ resilient w.r.t. Safe,
then it is $(k+1)$-step$_\exists$ resilient w.r.t. Safe.



Proof. (i) Let $G_0 \in \text{INIT}$. Consider a transformation sequence $G_0 \Rightarrow^*_{\mathcal{SE}} H \Rightarrow_{\mathcal{E}} M$. The correctness of $\langle S, \mathcal{SE}, \text{INIT}\rangle$ implies that $M \models \text{Safe}$.
(ii) If $M \Rightarrow^{\leq k}_{\mathcal{SE}} N \models \text{Safe}$, then $M \Rightarrow^{\leq k+1}_{\mathcal{SE}} N \models \text{Safe}$.
(iii) By deadlock-freeness, every transformation sequence $G \Rightarrow^*_{\mathcal{SE}} H \Rightarrow_{\mathcal{E}_A} M$ can be completed to a sequence $G \Rightarrow^*_{\mathcal{SE}} H \Rightarrow_{\mathcal{E}_A} M \Rightarrow^k_{\mathcal{SE}} N$. By k-step$_\forall$-resilience, there exists a subsequence $M \Rightarrow^{\leq k}_{\mathcal{SE}} N'$ with $N' \models \text{Safe}$.
(iv) If $M \Rightarrow^{\leq k}_{\mathcal{SE}} N \models \text{Safe}$, then $M \Rightarrow^{\leq k+1}_{\mathcal{SE}} N \models \text{Safe}$. □

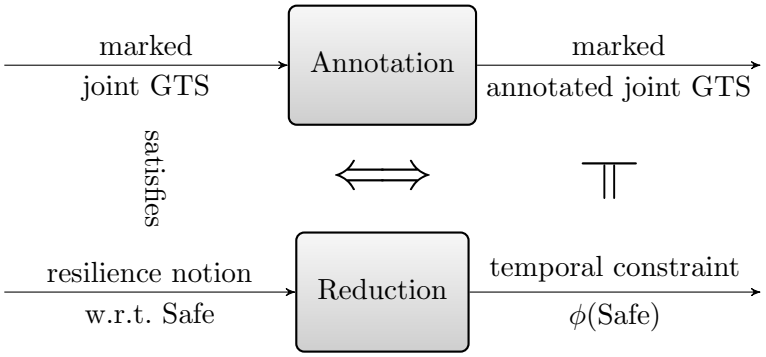
We exemplarily introduce a notion of resilience which demands not recovery after a number of steps but recovery until the next environment step, i.e., the safety constraint must hold before the next environment rule is applied.

Definition 3.5 ($\mathcal{E}$-step resilience). A marked joint GTS $\langle S, \mathcal{SE}, \text{INIT}\rangle$ is *$\mathcal{E}$-step resilient* w.r.t. Safe if, for every infinite transformation sequence $G_0 \Rightarrow^*_{\mathcal{SE}} H \Rightarrow_{\mathcal{E}_A} M \Rightarrow_{\mathcal{S}_A} N \Rightarrow \ldots$ with $G_0 \in \text{INIT}$, there exists a subsequence $M \Rightarrow^*_{\mathcal{S}_A} N'$ with $N' \models \text{Safe}$.

Example 3.9. Let Safe = NoBlocked. After applying `Block` there is a blocked track but immediately, by applying `Repair`, the blocked track vanishes. Thus, the traffic network system is $\mathcal{E}$-step resilient w.r.t. Safe.

3.3 Reduction to Temporal Logics

We propose a method for checking resilience by a reduction to temporal model checking, i.e., we express the resilience notions by means of temporal logics. We can check whether a marked joint GTS is resilient w.r.t. Safe by checking whether the marked annotated joint GTS satisfies a corresponding temporal constraint of Safe.



Temporal formulas such as LTL and CTL formulas well-known in logic, see, e.g., [CE82, Eme90, BK08]. We adapt the notions and consider so-called LTL and CTL graph constraints, i.e., temporal formulas whose atoms equate to graph constraints. A formal definition of LTL and CTL graph constraints can be found in Appendix A.

First, we consider LTL graph constraints. The temporality is interpreted as the changes along a transformation sequence. Every direct transformation correlates to a time step. Besides the common propositional operators there are *temporal operators*, e.g., the operator **X** (*Ne**X**t*) describes the validity of a formula in the next step while **G** (***G**lobally*) describes the validity of a formula in every following step. E.g., the expression

G NoBlocked

describes that there exists no blocked track in any derived graph, i.e., correctness w.r.t. NoBlocked. CTL graph constraints are, like LTL graph constraints, temporal formulas where the atoms equate to the graph constraints. By contrast, the temporality is here branching. Besides the common propositional operators, there are *path-quantified temporal operators* which are pairs of operators: the first one is either **A** (*for **A**ll following paths*) or **E** (*there **E**xists a following path*), the second one is a temporal operator. The operator **AG** means the valdity of a formula in all following sequences, and **EX** means that a graph can be reached in one step where the formula is valid. E.g., the expression

AG NoBlocked

describes that there exists no blocked track in any derived graph, i.e., correctness w.r.t. NoBlocked.

We first consider the formalization of k-step$_\forall$ and $\mathcal{E}$-step resilience as LTL constraints. The following theorem states that checking k-step$_\forall$/$\mathcal{E}$-step resilience of a joint GTS without deadlocks is equivalent to checking whether the annotated joint GTS satisfies a certain LTL constraint.

Theorem 3.1 (reduction to LTL). For every graph constraint Safe and every natural number $k \geq 0$, there exist LTL constraints ϕ(Safe) and ϕ_k(Safe) s.t. for every marked joint GTS without deadlocks:

(1) the marked joint GTS is k-step$_\forall$ resilient w.r.t. Safe iff the marked annotated joint GTS satisfies ϕ_k(Safe),

(2) the marked joint GTS is $\mathcal{E}$-step resilient w.r.t. Safe iff the marked annotated joint GTS satisfies ϕ(Safe).

Construction 3.4. Let

$$\phi_k(\text{Safe}) := \mathbf{G}(\text{Env} \Rightarrow \bigvee_{j=0}^{k} \mathbf{X}^j \text{Safe}),$$

$$\phi(\text{Safe}) := \mathbf{G}((\text{Env} \wedge \mathbf{X}\text{Sys}) \Rightarrow (\text{Safe} \vee \mathbf{X}(\text{Sys}\mathbf{U}\text{Safe})))$$

where k is a natural number, $\mathbf{X}^0$ is "an empty operator", and for $j \geq 0$, $\mathbf{X}^{j+1} := \mathbf{X}^j\mathbf{X}$ denotes the iterated next-operator in LTL. The positive constraint Sys := $\exists$(ⓢ) means that the last applied rule was a system rule. The positive constraint Env := $\exists$(ⓔ) means that the last applied rule was an environment rule.

The LTL constraint ϕ_k(Safe) can be read as "Whenever an environment rule was applied, there exists a $0 \leq j \leq k$ s.t. in j steps Safe holds". The LTL constraint ϕ(Safe) can be read as "Whenever an environment rule was applied and a system rule is applied next, (Safe holds or) only system rules are applied until Safe holds (and Safe holds eventually)".

Proof. See Appendix B. □

We consider the formalization of k-step$_\exists$ resilience as CTL constraint. The following theorem states that checking k-step$_\exists$ resilience of a joint GTS without deadlocks is equivalent to checking whether the annotated joint GTS satisfies a certain CTL constraint.

Theorem 3.2 (reduction to CTL). For every graph constraint Safe and every natural number $k \geq 0$, there exists a CTL constraints $\varphi_k(\text{Safe})$ s.t. for every marked joint GTS without deadlocks:

> the marked joint GTS is k-step$_\exists$ resilient w.r.t. Safe iff
> the marked annotated joint GTS satisfies $\varphi_k(\text{Safe})$.

Construction 3.5. Let

$$\varphi_k(\text{Safe}) := \mathbf{AG}(\text{Env} \Rightarrow \bigvee_{j=0}^{k} \mathbf{EX}^j \text{Safe}),$$

where k is a natural number, $\mathbf{EX}^0$ is "an empty operator", and for $j \geq 0$, $\mathbf{EX}^{j+1} := \mathbf{EX}^j\mathbf{EX}$ denotes the iterated existential next-operator in CTL.

The CTL constraint $\varphi_k(\text{Safe})$ can be read as "For all following sequences, whenever an environment rule was applied, there exists a following sequence and $0 \leq j \leq k$ s.t. Safe holds in j steps".

Proof. See Appendix B. □

3.4 Related Concepts

Correctness and *verification* of programs are addressed by **Apt et al.** [AOdB09] including also verification of *non-deterministic* and *distributed* programs. In **Pennemann** [Pen09] and **Poskitt & Plump** [PP13], correctness of *graph programs* is considered. Graph programs are programs whose basic components are sets of rules, i.e., graph transformation systems, and which are characterized by their closedness under non-deterministic composition, sequential composition, and iteration (as-long-as-possible).

The concept of adverse conditions addresses systems interacting with an adversary environment. Correctness in this sense means that the interaction between system and environment satsifies desired properties. In **Olderog et al.**, [OFTK21], a number of results on system correctness under adverse conditions in diverse contexts are presented.

In **Flick** [Fli16], correctness of *adverse reconfiguration nets* and *graph programs under adverse conditions* is considered. Adverse reconfiguration nets are structure-changing Petri nets. The structure change is realized by context-free graph transformation rules. It is shown that the word

problem, the reachability problem, and the abstract reachability problem are decidable under certain boundedness assumptions (at most one token in every marking, boundedly many reconfigurations). Moreover, a correctness notion for programs under adverse conditions (2-player programs) is introduced. Correctness is defined w.r.t. pre- and postconditions which are realized in form of *μ-conditions*, a generalization of graph conditions. A 2-player program has one system and one environment player. While each step of the environment player is considered, the system player can choose his step. It is shown that the method of the weakest precondition for μ-conditions also applies to this case. By constrast, we consider joint graph transformation systems where the control automaton is the only option to control the application of rules.

In **Peuser** [Peu18], *graph grammars under adverse conditions* are investigated. The considered grammars are hyperedge replacement grammars. The specification is realized by a *temporal (LTL) graph condition*. The interaction of system and environment is modeled by a finite labeled transition system. The two labels correspond to "system" and "environment", respectively. System and environment *play against each other*, i.e., their interaction is transformed into a *hyperedge replacement game*. Correctness is achieved by a *winning strategy*. In more detail, the (ordered) hyperegde replacement grammar and the temporal graph condition are transformed into a *pushdown process*. The temporal graph condition is translated into a parity automaton. Synchronizing the pushdown process with the labeled transition system and the parity automaton yields a pushdown process, a priority function, and a partition function. The combination of the latter three components is a hyperedge replacement game. By contrast, we consider control automata to model the interaction of system and environment. Instead of considering arbitrary temporal constraint, we show how to express meaningful resilience notions as specific LTL or CTL constraints.

In [GMSS19], **Giese et al.** introduce a *metric temporal logic* over *typed attributed graphs*. The atoms are graph conditions. Instead of *discrete* temporality like in the case of LTL, the *continuous* set of non-negative real numbers is considered. The main point of the definition is the until-operator w.r.t. an interval of real numbers. This approach also incorporates the tracking of elements over time, which is not possible with LTL/CTL constraints.

The concept of *self-stabilization*, introduced by **Dijkstra** [Dij74] and considered, e.g., in **Dolev** [Dol00], shows similarities with our resilience notions. A system is self-stabilizing if it fulfills the following two conditions: Starting from an arbitrary state, the system will eventually reach a safe state (*convergence*). If the system is in a safe state, it will stay in a safe state (*closure*). A probabilistic system is *k-self-stabilizing* if it satisfies the closure condition and: starting from an arbitrary state, the system will reach a safe state within an expected number of steps, which is bounded by a constant k

(convergence). Although we do not consider probabilistic systems, the latter notion is very similar to k-step resilience where recovery in at most k steps is demanded. In **Müllner et al.** [MTF13], verification of self-stabilizing probabilistic systems by state space analysis is investigated. It is shown how to compute the so-called limiting window availibility probability for reduced Markov chains of subsystems. By recombination, the latter probability can be computed for the whole system, often avoiding a state space explosion.

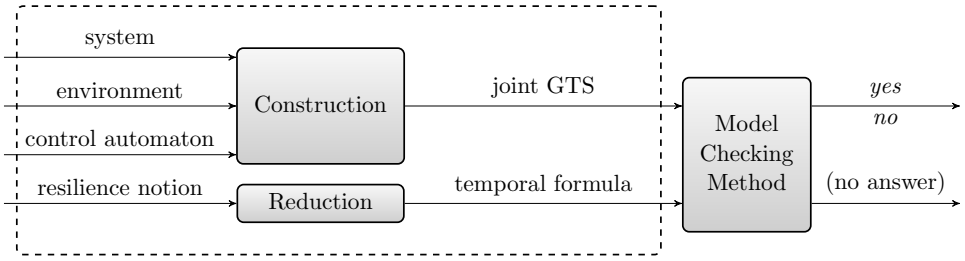
Methods and Tools for Model Checking

We adhered to **Emerson** [Eme90] and **Baier & Katoen** [BK08] for the syntax and semantics of LTL and CTL. In [BK08], Baier & Katoen present formal methods for temporal model checking of finite-state transition systems. Besides LTL and CTL, they consider also CTL* – an extension of both – as well as timed and probabilistic temporal logics. They discuss several model checking techniques, e.g., automata-based model checking for LTL and symbolic model checking for CTL. The tool GROOVE by **Kastenberg & Rensink** [KR06] provides the opportunity of LTL and CTL model checking as well as state space exploration for (single-pushout) graph transformation systems. The atomic proposition are given as rules. By considering identical rules, also restricted graph constraints are included. By contrast, the tool HENSHIN by **Arendt et al.** [ABJ$^+$10] supports the more expressive μ-calculus.

While most model checking methods are limited to finite-state systems, the tools AUGUR 2 by **König & Kozioura** [KK08] and UNCOVER by **Stückrath** [Stü15] provide the possibility to verify infinite-state graph transformation systems. The former is based on methods of approximation by Petri nets, see, e.g., **Baldan et al.** [BCK08], the latter on algorithms for well-structured graph transformation systems, see, e.g., **König & Stückrath** [KS17]. **Steenken** [Ste15] proposes a method of abstraction by graph shapes to verify infinite-state graph transformation systems. Using forward methods, **Blondin et al.** [BFG20] show that LTL model checking is decidable for a subclass of strongly well-structured transition systems and, as a consequence, for so-called ω-Petri nets.

3.5 Summary

Joint GTSs are constructed from a system, an environment (both GTSs), and a control automaton modeling their interaction. This construction yields again a GTS. Meaningful resilience notions (k-step$_\forall$, k-step$_\exists$, and $\mathcal{E}$-step resilience) are presented for this kind of GTSs. These resilience notions capture recovery of a safety constraint after an interference of the environment. We showed how to express these resilience notions in temporal logics.



In the Chapters 4 and 5, we investigate on the verification of k-step$_\exists$ resilience.

Takeaway (Chapter 3).

- GTSs under adverse conditions are modeled by two GTSs, a system, an environment, and a control automaton specifying their interaction. The constructed GTS is called joint GTS.
- Meaningful resilience notions for joint GTSs are presented.
- These resilience notions are expressible in temporal logics.

Chapter 4

Verifying Resilience in a Well-structured Framework

Our goal is the verification of resilience for graph transformation systems. To this aim, we use more abstract methods at the level of transition systems. In the context of graph transformation, states are captured by graphs and transitions by graph transformations. A transition system consists of a set of states of any kind (not necessarily graphs) and a transition relation on the state set.

This chapter deals with the decidability of resilience problems. We investigate on the question whether, starting from any INITial state, a SAFE state can be reached in a bounded number of steps from any BAD state where BAD is not necessarily the complement of SAFE. Usually, the state set of the transition system – also in the context of graph transformation – is infinite. To handle infinite state sets (sets of graphs), we employ the concept of well-structuredness as considered, e.g., in Abdulla et al. [AČJT96], Finkel & Schnoebelen [FS01], König & Stückrath [KS17]. Recall that a well-structured transition system is, informally, a transition system equipped with a well-quasi-order satisfying that "larger" states simulate "smaller" states and that certain predecessor sets can be effectively computed. In this well-structured setting, ideal-based sets (upward- or downward-closed sets) play an important role. They enjoy a number of properties simplifying verification such as finite representation of upward-closed sets and closure properties. For well-structured transition systems, the ideal reachability problem is known to be decidable [AČJT96], which is an integrant of our results.

In Section 4.1, we introduce the resilience problem for well-structured transition systems. In Section 4.2, we show the decidability of the resilience problem for subclasses of well-structured transition systems. In Section 4.3, we consider the algorithms in more detail as well as approximations.

4.1 Resilience Problem

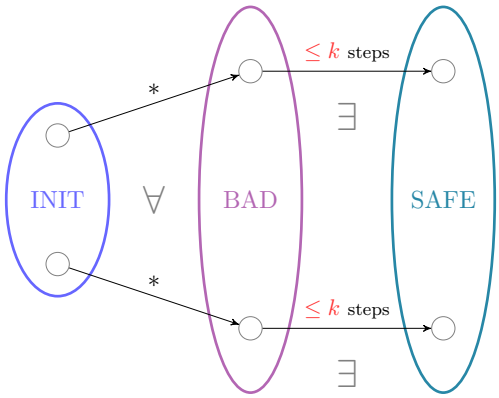
We formulate resilience problems for marked WSTSs, i.e., WSTS with a specified set INIT of initial states starting from which we investigate resilience.

Definition 4.1 (marked WSTS). A *marked well-structured transition system* is a tuple $\langle S, \leq, \rightarrow, \text{INIT}\rangle$ where $\langle S, \leq, \rightarrow\rangle$ is a WSTS and $\text{INIT} \subseteq S$. It is *finite-marked* if INIT is finite.

We consider the *bounded* resilience problem which asks whether there exists such a bound on the number of steps for recovery.

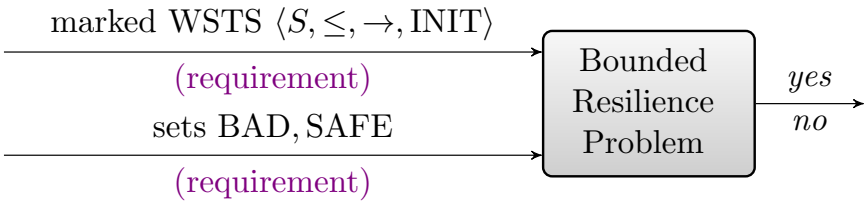
Bounded Resilience Problem for WSTSs	
Given:	A marked WSTS $\langle S, \leq, \rightarrow, \text{INIT}\rangle$ and sets $\text{BAD}, \text{SAFE} \subseteq S$.
Question:	Does there exist a $k \geq 0$ s.t. starting from any state in INIT, whenever we reach a BAD state, we can reach a SAFE state in $\leq k$ steps?[7]

The answer to this question is *yes* if there is a natural number $k \geq 0$ s.t. the marked WSTS is *k-step resilient* w.r.t. BAD and SAFE, i.e., starting from any state in INIT, whenever we reach a BAD state, we can reach a SAFE state in $\leq k$ steps, and *no* otherwise. The set BAD specifies which kind reachable states are of interest. We do not require that BAD is the complement of SAFE.



We are interested in suitable subclasses of marked WSTSs and suitable sets BAD and SAFE, for which the bounded resilience problem is decidable.

[7] A BAD (SAFE) state is a state in BAD (SAFE).



We show that for suitable subclasses of finite-marked WSTSs and so-called ideal-based sets BAD and SAFE, the bounded resilience problem is decidable.

Example 4.1 (supply chain). We consider a marked Petri net (see Appendix A) modeling a simplified scenario of a supply chain. The supply chain, depicted in Figure 4.1, consists of a production site labeled with *product*, a warehouse labeled with *warehouse*, and two stores labeled with $store_1$ and $store_2$, respectively. Products are produced in the production site, transported to the warehouse, and shipped to one of the stores where products can be bought. Products also may get lost by an accident in the warehouse. As usual we depict places as circles, transitions as rectangles, and the flow as weighted directed arcs between them. In the example, all weights are 1 and therefore not indicated. Dots on places indicate the number of tokens on the respective place in the marking. The INITial marking is given in Figure 4.1. The light-red transitions *accident*, buy_1, and buy_2 reduce the number of tokens in the net.

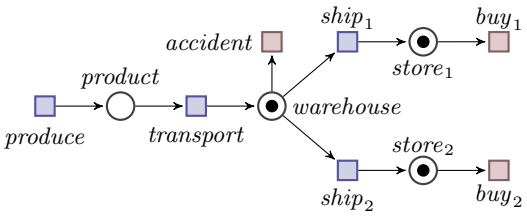


Figure 4.1: A Petri net modeling a supply chain.

For the bounded resilience problem, we are interested in a bound k for the number of steps needed for recovery. Let SAFE =

$$\{M \text{ marking } : M(warehouse), M(store_1), M(store_2) \geq 1\},$$

i.e., in the warehouse and in both stores products are available for shipping or purchase, respectively. We ask whether we can reach a marking in SAFE in $\leq k$ steps whenever we reach marking not in SAFE, i.e., BAD =

$$\{M \text{ marking } : M(warehouse) = 0 \text{ or } M(store_i) = 0 \text{ for some } i\}.$$

One may ask:

1. Does such a k exist?
2. Is there a generic method for problems of this kind?

We will answer these questions in this chapter.

4.2 Decidability

We show the decidability of resilience problems for subclasses of well-structured transition systems with strong compatibility, i.e., we regard requirements. These requirements ensure that we can (i) compute a finite representation of reachable BAD states for (ii) checking inclusion in a predecessor set of the SAFE states. While "post*-effectiveness" or "lossiness" ensures the former, "$\bot$-boundedness" ensures the latter in the case that SAFE is downward-closed.

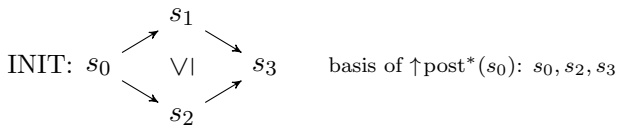
Definition 4.2 (requirements). A marked WSTS $\langle S, \leq, \rightarrow, \text{INIT}\rangle$ is

(1) *post*-effective*
if INIT is finite and a basis of $\uparrow\text{post}^*(\text{INIT})$ is computable,

(2) *lossy* if $\downarrow\text{post}^*(\text{INIT}) = \text{post}^*(\text{INIT})$,

(3) $\bot$-*bounded* if there is a natural number $\ell \geq 0$ s.t. $b \in \text{post}^{\leq \ell}(s)$ for every $s \in S$ and every element b of a basis of S with $s \geq b$.

The requirement of post*-effectiveness describes that a basis of the upward-closure of the reachable states from the initial states is computable, i.e., we can effectively obtain a finite representation of the reachable states (in the sense that we can compute the "smallest" reachable states). In the example below, $\text{post}^*(s_0)$ consists of four states s_0, s_1, s_2, s_3. Since $s_2 \leq s_1$, the basis of $\uparrow\text{post}^*(s_0)$ consists of s_0, s_2, s_3.



The notion of lossiness is an abstraction from the lossiness concept in Finkel & Schnoebelen [FS01, p. 83]. Lossiness in our sense means that the set of reachable states from the initital states is downward-closed. Usually, "lossy" describes the circumstance that every "non-solid component"[8] of every state (in our sense: every state reachable from INIT) may get lost. Hence, the system is unreliable to some extent. Another kind of unreliability is $\bot$-boundedness. The notion of $\bot$-boundedness means that from every state, every "smaller" basis element, i.e., the bottom underneath, is reachable in a

[8] "Solid components" of a state s are the parts of a basis element $b \leq s$.

bounded number of steps. Systems of this kind are unreliable in the sense that all "non-solid components" of a state may get lost in a bounded number of steps.

The following lemma is crucial for many following proofs. It states that the inclusion of an intersection of set with an anti-ideal in an ideal is equivalent to the inclusion of the intersection of the upward-closure of the set with the anti-ideal.

Lemma 4.1 (ideal-inclusion). Let $A \subseteq S$ be a set, $I \subseteq S$ an ideal, and $J \subseteq S$ an anti-ideal. Then,

$$A \cap J \subseteq I \iff \uparrow A \cap J \subseteq I.$$

Proof. "$\Leftarrow$": Holds since $A \subseteq \uparrow A$.
"$\Rightarrow$": Let $s \in \uparrow A \cap J$. Then, $\exists a \in A : s \geq a$. Thus, $a \in \downarrow J = J$. Hence, $a \in I$ and $s \in \uparrow I = I$. □

Applying this lemma to a basis of an ideal, we obtain that the inclusion of an intersection of an ideal and an anti-ideal in an ideal can be checked by computing the intersection of the basis with the anti-ideal and checking inclusion afterwards.

We give a characterization of post*-effectiveness via the anti-ideal reachability problem.

Proposition 4.1 (characterization of post*-effectiveness). For a class of finite-branching WSTSs, a basis of $\uparrow \mathrm{post}^*(s)$ is computable for a given state s iff the anti-ideal reachability problem is decidable, i.e., given a state s of a WSTS $\langle S, \leq, \rightarrow \rangle$ and an anti-ideal $J \subseteq S$ with a given basis of $S \setminus J$, it can be decided whether $\exists s' \in J : s \rightarrow^* s'$.

Proof. On one hand we can decide the anti-ideal reachability problem by computing a basis of $\uparrow \mathrm{post}^*(s)$ and checking whether the intersection with the anti-ideal is empty (Lemma 4.1). On the other hand, we can compute a basis of $\uparrow \mathrm{post}^*(s)$ by computing the sequence of ideals $P_k = \uparrow \mathrm{post}^{\leq k}(s)$ until it becomes stationary (Lemma 2.2). The stop condition, i.e., the condition which guarantees that we can terminate the algorithm, is formalized as anti-ideal reachability:

"$\Leftarrow$": Since the WSTS is finite-branching, for every $k \geq 0$, $\mathrm{post}^{\leq k}(s)$ is finite and computable. By Fact 2.6, for every $k \geq 0$, a basis of $\uparrow \mathrm{post}^{\leq k}(s)$ is computable. We compute the sequence of ideals $P_k = \uparrow \mathrm{post}^{\leq k}(s)$ until it becomes stationary (Lemma 2.2). The decidable stop condition is to ask whether the anti-ideal $S \setminus P_k$ is reachable from s. Namely, it holds: $\mathrm{post}^*(s) \cap S \setminus P_k = \varnothing \iff \mathrm{post}^*(s) \subseteq P_k \iff \uparrow \mathrm{post}^*(s) = P_k$, i.e., we stop the computation when $S \setminus P_k$ is not reachable from s.
"$\Rightarrow$": For deciding whether an anti-ideal J is reachable from s, we check

whether $\mathcal{B}_{\text{post}}(s) \cap J = \varnothing$ where $\mathcal{B}_{\text{post}}(s)$ is a basis of $\uparrow \text{post}^*(s)$. This is equivalent to $\text{post}^*(s) \cap J = \varnothing$ by Lemma 4.1. Membership in J is decidable since a basis of $S \setminus J$ is given. □

The characterization in Proposition 4.1 is used to show that Petri nets are post*-effective. It is well-known that Petri nets constitute SWSTSs, see, e.g., Finkel & Schnoebelen [FS01, Thm. 6.1]. More generally, also reset Petri nets constitute SWSTSs, see, e.g., Dufourd et al. [DFS98]. For a formal definition of reset Petri nets, see Appendix A.

Convention. We say that a marked reset Petri net is post*-effective if the induced marked SWSTS is post*-effective. The same convention applies to "lossy" and "⊥-bounded".

Example 4.2 (variations of Petri nets).

(1) Petri nets (equipped with any finite set of initial states) are post*-effective by Proposition 4.1 and results of Mayr [May84] and Hack [Hac85]: Reachability for Petri nets is decidable [May84] and recursively equivalent to submarking reachability [Hac85]. This corresponds to the anti-ideal reachability problem for Petri nets.

(2) *1-Lossy Petri nets* are Petri nets where in any state, one token may get lost at any place. 1-Lossy Petri nets are lossy for every set of initial states. (A more common notion of lossy Petri nets, which differs slightly from ours, is given in Bouajjani & Mayr [BM99].)

(3) *Reset-lossy (mixed-lossy) Petri nets* are reset Petri nets where in any state, all tokens (and one token) may get lost at any place. Reset-lossy (mixed-lossy) Petri nets are ⊥-bounded (and lossy) for every set of initial states. The bound for returning to the zero-marking is given by the number of places.

For some results, we assume that a basis of all states of the considered WSTS is given. This is relevant when we use the basis elements for computations as in the following proposition.

Notation. For a (S)WSTS $\langle S, \leq, \to \rangle$ with a given basis of S, we write shortly (S)WSTS$^{\mathcal{B}}$.

The next proposition shows how the requirements are related provided that a basis of the set of all states is given.

Proposition 4.2. Lossy (⊥-bounded) finite-marked WSTS$^{\mathcal{B}}$s are post*-effective.

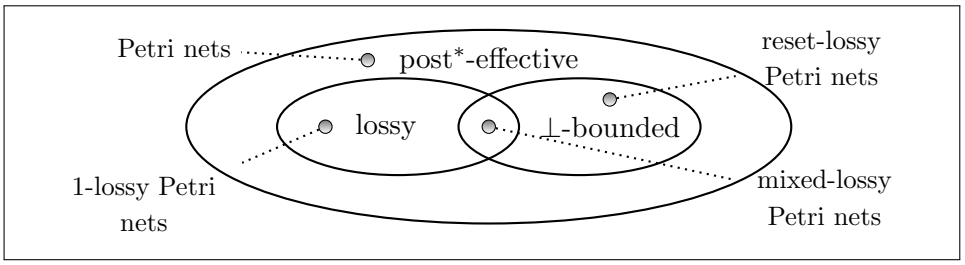


Figure 4.2: Subclasses of finite-marked WSTS$^{\mathcal{B}}$s .

Proof. Let $\langle S, \leq, \rightarrow, \text{INIT}\rangle$ be a lossy ($\bot$-bounded) finite-marked WSTS$^{\mathcal{B}}$. To compute a basis of $\uparrow\text{post}^*(\text{INIT})$ for a finite set INIT, we look at the reachable elements of a basis of the set S of all states. Such a basis element is reachable iff its upward-closure is reachable:

Let $\mathcal{B}$ be a basis of S and $s \in \text{INIT}$. In both cases, a basis of $\uparrow\text{post}^*(s)$ is given by the set $\mathcal{B}_{\text{post}}(s) = \{b \in \mathcal{B} : \uparrow\{b\} \text{ is reachable from } s\}$ which can be computed by Lemma 2.3. We show that $\uparrow\mathcal{B}_{\text{post}}(s) = \uparrow\text{post}^*(s)$:
"$\supseteq$": Let s' be any element reachable from s. By definition of a basis, there exists a basis element $b \leq s'$. It follows that $\uparrow\{b\}$ is reachable from s. We showed $\text{post}^*(s) \subseteq\uparrow\mathcal{B}_{\text{post}}(s)$ and thus, $\uparrow\text{post}^*(s) \subseteq\uparrow\mathcal{B}_{\text{post}}(s)$.
"$\subseteq$": By definition of lossiness and $\bot$-boundedness, if $\uparrow\{b\}$ is reachable from s, also b is reachable from s. We showed $\mathcal{B}_{\text{post}}(s) \subseteq \text{post}^*(s)$ and hence, $\uparrow\mathcal{B}_{\text{post}}(s) \subseteq\uparrow\text{post}^*(s)$.
The set INIT is finite and, by Fact 2.6, $\mathcal{B}_{\text{post}}(s)$ is finite for every $s \in S$. Hence, $\bigcup_{s\in\text{INIT}} \mathcal{B}_{\text{post}}(s)$ is finite. It holds

$$\begin{aligned}\uparrow\text{post}^*(\text{INIT}) =&\uparrow \bigcup_{s\in\text{INIT}} \text{post}^*(s) = \bigcup_{s\in\text{INIT}} \uparrow\text{post}^*(s) = \bigcup_{s\in\text{INIT}} \uparrow\mathcal{B}_{\text{post}}(s)\\ =&\uparrow \bigcup_{s\in\text{INIT}} \mathcal{B}_{\text{post}}(s).\end{aligned}$$

By Fact 2.6, we can compute a basis of $\uparrow\text{post}^*(\text{INIT})$. □

In our setting, ideals and anti-ideals of the set of states play an important role. Ideals and anti-ideals are, in general, infinite sets. However, every ideal can be represented by a finite basis. For anti-ideals, we assume that they are decidable.

Definition 4.3 (ideal-based). A set is *ideal-based* if it is

(a) an ideal with a given basis, or

(b) a decidable anti-ideal.

We denote the set of ideals with a given basis by $\mathcal{I}$ and the set of decidable anti-ideals by $\mathcal{J}$.

A main result of this chapter is the decidability of the bounded resilience problem for subclasses of finite-marked WSTSs with bounded compatibility and the respective requirements. Since our main objective is the decidability for marked GTSs – and marked GTSs naturally satisfy strong comptability w.r.t. the subgraph order –, we formulate and prove the result for SWSTSs.

Theorem 4.1 (bounded resilience for finite-marked SWSTSs). The bounded resilience problem is decidable for

(1) post*-effective finite-marked SWSTSs if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy finite-marked SWSTSs if BAD, SAFE $\in \mathcal{I}$,

(3) lossy, $\bot$-bounded finite-marked SWSTS$^{\mathcal{B}}$s if BAD $\in \mathcal{I}$, SAFE $\in \mathcal{J}$,

(4) $\bot$-bounded finite-marked SWSTS$^{\mathcal{B}}$s if BAD, SAFE $\in \mathcal{J}$.

Key Idea of the Proof. We aim to compute a finite representation of post*(INIT) $\cap$ BAD for checking inclusion in a decidable ideal I. The decidable ideal I is a predecessor set of SAFE.
The proof structure is shown in Figure 4.3: Lemma 4.2 states that for post*-effective (lossy) finite-marked SWSTSs, a finite representation of post*(INIT) $\cap$ BAD is computable, i.e., inclusion in a decidable ideal is decidable. In the case SAFE $\in \mathcal{I}$, the set pre$^{\leq k}$(SAFE) is a decidable ideal for every $k \geq 0$ [AČJT96]. Lemma 4.3 shows the existence of bounds for the set of all predecessors of SAFE $\in \mathcal{J}$ provided that the SWSTS is $\bot$-bounded. Proposition 4.3 shows that pre*(SAFE) constitutes a decidable ideal in the case SAFE $\in \mathcal{J}$ if the SWSTS$^{\mathcal{B}}$ is $\bot$-bounded. □

The following lemma states that the inclusion of the intersection post*(INIT) $\cap$ BAD in an decidable ideal is decidable if we consider post*-effective in the case BAD $\in \mathcal{J}$ or lossy finite-marked WSTSs in the case BAD $\in \mathcal{I}$.

Lemma 4.2 (checking inclusion). Let $\langle S, \leq, \rightarrow, \text{INIT}\rangle$ be a finite-marked WSTS, BAD $\subseteq S$, and $I \subseteq S$ be a decidable ideal. It is decidable whether post*(INIT) $\cap$ BAD $\subseteq I$ provided that the finite-marked WSTS is

(a) post*-effective and BAD $\in \mathcal{J}$,

(b) lossy and BAD $\in \mathcal{I}$.

Proof. The idea is to compute a finite representation of the intersections post*(INIT) $\cap$ BAD for checking inclusion in the decidable ideal I. To this aim, we use Lemma 4.1:

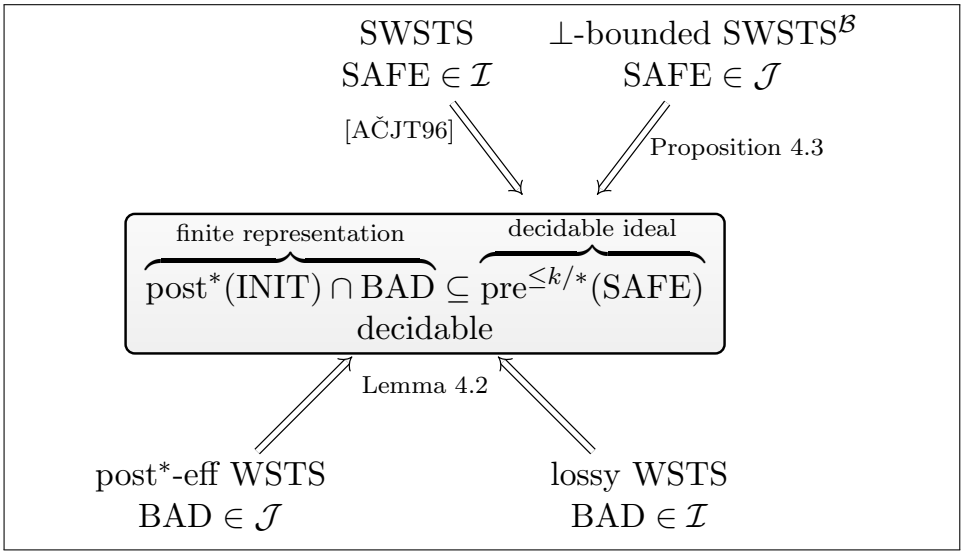


Figure 4.3: Decidability for finite-marked SWSTSs.

(a) We consider post*-effective finite-marked WSTSs and BAD $\in \mathcal{J}$. It holds

$$\begin{aligned} & \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \\ \iff & \uparrow \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \text{(Lemma 4.1)} \\ \iff & B_{\mathrm{post}}(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \text{(Lemma 4.1)} \end{aligned}$$

where the set $B_{\mathrm{post}}(\mathrm{INIT})$ is a basis of $\uparrow \mathrm{post}^*(\mathrm{INIT})$, i.e., $\uparrow B_{\mathrm{post}}(\mathrm{INIT}) = \uparrow \mathrm{post}^*(\mathrm{INIT})$. By post*-effectiveness, $B_{\mathrm{post}}(\mathrm{INIT})$ is computable. By Fact 2.6, $B_{\mathrm{post}}(\mathrm{INIT})$ is finite. The last inclusion is algorithmically checkable. We take out all elements of $B_{\mathrm{post}}(\mathrm{INIT})$ which are not in the decidable anti-ideal BAD and then check inclusion of the remaining elements in the decidable ideal I.

(b) We consider lossy finite-marked WSTSs and BAD $\in \mathcal{I}$. We use the same idea as in the previous case, but we change the roles of post*(INIT) and BAD. It holds

$$\begin{aligned} & \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \\ \iff & \downarrow \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \text{(Def. lossy)} \\ \iff & \downarrow \mathrm{post}^*(\mathrm{INIT}) \cap \mathcal{B}_{\mathrm{BAD}} \subseteq I & \text{(Lemma 4.1)} \end{aligned}$$

where $\mathcal{B}_{\mathrm{BAD}}$ is a basis of BAD, i.e., $\uparrow \mathcal{B}_{\mathrm{BAD}} = \mathrm{BAD}$. We show that $\downarrow \mathrm{post}^*(\mathrm{INIT})$ is a decidable anti-ideal. It holds $s \in \downarrow \mathrm{post}^*(\mathrm{INIT})$ iff $\mathrm{INIT} \cap \mathrm{pre}^*(\uparrow \{s\}) \neq \varnothing$ for any $s \in S$. Since INIT is finite, we check whether

$s' \in \mathrm{pre}^*(\uparrow\{s\})$ for every $s' \in \mathrm{INIT}$. The latter is decidable by Lemma 2.3. Hence, the last inclusion is decidable. We take out all elements of $\mathcal{B}_{\mathrm{BAD}}$ which are not in $\downarrow \mathrm{post}^*(\mathrm{INIT})$ and then check inclusion of the remaining elements in the decidable ideal I. □

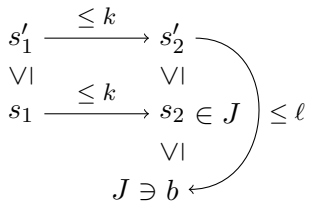
The sets we want to check inclusion in are $\mathrm{pre}^*(\mathrm{SAFE})$ or $\mathrm{pre}^{\leq k}(\mathrm{SAFE})$, $k \geq 0$. In the case $\mathrm{SAFE} \in \mathcal{I}$, for SWSTSs, $\mathrm{pre}^{\leq k}(\mathrm{SAFE})$ for every $k \geq 0$ is a decidable ideal.

By the next lemma, $\perp$-boundedness implies that for any anti-ideal J, $\mathrm{pre}^*(J)$ is an ideal and $\mathrm{pre}^*(J) = \mathrm{pre}^{\leq k}(J)$ for a $k \geq 0$.

Lemma 4.3 (existence of bounds). For every $\perp$-bounded SWSTS and decidable anti-ideal $J \subseteq S$, there exists a $k \geq 0$ s.t. $\mathrm{pre}^*(J) = \uparrow \mathrm{pre}^*(J) = \mathrm{pre}^{\leq k}(J)$.

Proof. Recall that, by Lemma 2.2, for every set A of states of a WSTS, there exists a $k_0 \geq 0$ s.t. $\uparrow \mathrm{pre}^*(A) = \uparrow \mathrm{pre}^{\leq k_0}(A)$. By strong compatibility and $\perp$-boundedness, we obtain $\uparrow \mathrm{pre}^{\leq k}(J) \subseteq \mathrm{pre}^{\leq k+\ell}(J)$ for every anti-ideal J and every $k \geq 0$ where ℓ is a constant:

We show $\uparrow \mathrm{pre}^{\leq k}(J) \subseteq \mathrm{pre}^{\leq k+\ell}(J)$ for every anti-ideal J and every $k \geq 0$ where ℓ is a constant. Let $s_1' \geq s_1$ with $s_1 \rightarrow^{\leq k} s_2 \in J$ and $k \geq 0$. By strong compatibility, there exists $s_2' \geq s_2$ with $s_1' \rightarrow^{\leq k} s_2'$. By definition of a basis, there exists a basis element $b \leq s_2$. It holds $b \in J$. Since $b \leq s_2'$, by $\perp$-boundedness, there exists a bound ℓ s.t. $s_2' \rightarrow^{\leq \ell} b \in J$.



By Lemma 2.2, there exists $k_0 \geq 0$ s.t. $\uparrow \mathrm{pre}^*(J) = \uparrow \mathrm{pre}^{\leq k_0}(J)$. We obtain

$$\mathrm{pre}^*(J) \subseteq \uparrow \mathrm{pre}^*(J) = \uparrow \mathrm{pre}^{\leq k_0}(J) \subseteq \mathrm{pre}^{\leq k_0+\ell}(J) \subseteq \mathrm{pre}^*(J).$$

Hence, $\mathrm{pre}^*(J) = \uparrow \mathrm{pre}^*(J) = \mathrm{pre}^{\leq k_0+\ell}(J)$. □

The following proposition identifies sufficient prerequisites s.t. $\mathrm{pre}^*(\mathrm{SAFE})$ constitutes a decidable ideal in the case $\mathrm{SAFE} \in \mathcal{J}$.

Proposition 4.3 (decidable ideals). For every $\perp$-bounded $\mathrm{SWSTS}^{\mathcal{B}}$ and decidable anti-ideal $J \subseteq S$, the set $\mathrm{pre}^*(J)$ is a decidable ideal.

Proof. By Lemma 4.3, $\uparrow \mathrm{pre}^*(J) = \mathrm{pre}^*(J)$. Thus, it is an ideal. Let $s \in S$. It holds

$$\begin{aligned} & s \notin \mathrm{pre}^*(J) & \\ \iff & \nexists s' \in J : s \to^* s' & \text{(Def. preset)} \\ \iff & \mathrm{post}^*(s) \cap J = \varnothing & \text{(Def. postset)} \\ \iff & \uparrow \mathrm{post}^*(s) \cap J \subseteq \varnothing & \text{(Lemma 4.1)} \\ \iff & B_{\mathrm{post}}(s) \cap J \subseteq \varnothing & \text{(Lemma 4.1)} \end{aligned}$$

where $B_{\mathrm{post}}(s)$ is a basis of $\uparrow \mathrm{post}^*(s)$. By Proposition 4.2, $\bot$-boundedness implies post*-effectiveness w.r.t. any finite set of initial states provided that a basis of S is given. Hence, $B_{\mathrm{post}}(s)$ is computable. Thus, the last inclusion is decidable. □

We compile our preparatory results to prove Theorem 4.1.

Proof of Theorem 4.1. We want to verify the property

$$\exists k \geq 0 : \ \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^{\leq k}(\mathrm{SAFE}).$$

This property is indeed equivalent to the resilience property since it can be formalized as

$$\exists k \geq 0 : \ \forall (\mathrm{INIT} \ni s \to^* s' \in \mathrm{BAD}) : \ \exists (s' \to^{\leq k} s'' \in \mathrm{SAFE}).$$

Cases (1) & (2). By Fact 2.8, $\mathrm{pre}^{\leq k}(\mathrm{SAFE})$ is an ideal for every $k \geq 0$ since $\mathrm{SAFE} \in \mathcal{I}$ and, by Definition 2.12, it is decidable. By Lemma 4.2, we can decide whether $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^{\leq k}(\mathrm{SAFE})$ for (1) post*-effective finite-marked SWSTSs and (2) lossy finite-marked SWSTSs, respectively. By Lemma 2.2, the sequence $\mathrm{SAFE} \subseteq \mathrm{pre}^{\leq 1}(\mathrm{SAFE}) \subseteq \mathrm{pre}^{\leq 2}(\mathrm{SAFE}) \subseteq \ldots$ becomes stationary, i.e., there is a minimal $k_0 \geq 0$ s.t. $\mathrm{pre}^{\leq k_0}(\mathrm{SAFE}) = \mathrm{pre}^*(\mathrm{SAFE})$. By Lemma 2.3, we can also determine this k_0. Thus, we can determine the minimal number $k = k_{\min}$ s.t. $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^{\leq k}(\mathrm{SAFE})$ (if it exists) and also whether it exists. Hence, we can decide the bounded resilience problem. To sum up, the bounded resilience problem is decidable for (1) post*-effective finite-marked SWSTSs and (2) lossy finite-marked SWSTSs, respectively.

Cases (3) & (4). By Lemma 4.3, for $\bot$-bounded SWSTSs, there exists a $k \geq 0$ s.t. $\mathrm{pre}^*(\mathrm{SAFE}) = \mathrm{pre}^{\leq k}(\mathrm{SAFE})$. Hence, checking bounded resilience is equivalent to testing inclusion in $\mathrm{pre}^*(\mathrm{SAFE})$. By Proposition 4.3, for $\bot$-bounded SWSTS$^{\mathcal{B}}$s, $\mathrm{pre}^*(\mathrm{SAFE})$ is a decidable ideal since $\mathrm{SAFE} \in \mathcal{J}$. By Lemma 4.2, we obtain that checking $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^*(\mathrm{SAFE})$ is decidable for (3) lossy, $\bot$-bounded finite-marked SWSTS$^{\mathcal{B}}$s and (4) post*-effective, $\bot$-bounded finite-marked SWSTS$^{\mathcal{B}}$s, respectively. By Proposition 4.2, $\bot$-boundedness implies post*-effectiveness provided that a basis of

all states is given. Hence, we can decide the bounded resilience problem for (3) lossy, $\bot$-bounded SWSTS$^{\mathcal{B}}$s and (4) $\bot$-bounded SWSTS$^{\mathcal{B}}$s, respectively. □

Reset Petri nets can be seen as a special kind of graph transformation systems. By replacing "SWSTSs" with "reset Petri nets", we obtain the decidability of the bounded resilience problem for finite-marked reset Petri nets (including finite-marked Petri nets).[9]

Corollary 4.1 (bounded resilience for finite-marked reset Petri nets). The bounded resilience problem is decidable for

(1) post*-effective finite-marked reset Petri nets if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy finite-marked reset Petri nets if BAD, SAFE $\in \mathcal{I}$,

(3) lossy, $\bot$-bounded finite-marked reset Petri nets if BAD $\in \mathcal{I}$, SAFE $\in \mathcal{J}$,

(4) $\bot$-bounded finite-marked reset Petri nets if BAD, SAFE $\in \mathcal{J}$.

Remark. The bounded resilience is decidable for finite-marked Petri nets if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$ since finite-marked Petri nets are post*-effective.

Besides the bounded resilience problem, one may be interested in the so-called *explicit* resilience problem where, additionally, a bound k is given as input. We formulate the explicit resilience problem for marked WSTSs.

Explicit Resilience Problem for WSTSs	
Given:	A marked WSTS $\langle S, \leq, \rightarrow, \text{INIT}\rangle$, sets BAD, SAFE $\subseteq S$, a natural number $k \geq 0$.
Question:	Starting from any state in INIT, whenever we reach a BAD state, can we reach a SAFE state in $\leq k$ steps?

We are interested in suitable subclasses of marked WSTSs and suitable sets BAD and SAFE, for which the explicit resilience problem is decidable.

Remark. For the explicit resilience problem, "precision" is required, i.e., we require that for every ideal I and every $k \geq 0$, the sets $\text{pre}^{\leq k}(I)$ are ideals. Strong compatibility is crucial in this case.

For SAFE $\in \mathcal{I}$, the decidability of the bounded resilience problem yields the decidability of the explicit resilience problem: If the algorithm deciding the bounded resilience problem gives the answer *yes*, it provides also the minimal bound k. If the answer to the bounded resilience problem is *no*, the answer to the explicit resilience problem is *no* for every k.

[9] In the literature, finite-marked (reset) Petri nets are just called "marked (reset) Petri nets". In contrast to Petri nets, reset Petri nets are, in general, not post*-effective, i.e., there is no generic procedure to compute the basis of $\uparrow\text{post}^*(\text{INIT})$, see, Appendix B.

Theorem 4.2 (explicit resilience for finite-marked SWSTSs). The explicit resilience problem is decidable for finite-marked SWSTSs which are

(1) post*-effective if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy if BAD, SAFE $\in \mathcal{I}$.

Proof. Inspecting the proof of Theorem 4.1, we find that we can compute the minimal $k = k_{\min}$ s.t. post*(INIT) $\cap$ BAD $\subseteq$ pre$^{\leq k}$(SAFE) if it exists. Thus, given k, we can check $k_{\min} \leq k$ to decide the explicit resilience problem. If $k_{\min}$ does not exist, the answer to the explicit resilience problem is *no* for every k. □

By replacing "SWSTSs" with "reset Petri nets", we obtain the decidability of the explicit resilience problem for finite-marked reset Petri nets (including finite-marked Petri nets).

Corollary 4.2 (explicit resilience for finite-marked reset Petri nets).

The explicit resilience problem is decidable for

(1) post*-effective finite-marked reset Petri nets if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy finite-marked reset Petri nets if BAD, SAFE $\in \mathcal{I}$.

Remark. The explicit resilience is decidable for finite-marked Petri nets if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$ since finite-marked Petri nets are post*-effective.

Example 4.3 (supply chain). Consider again the marked Petri net modeling a simplified scenario of a supply chain, depicted in Figure 4.4.

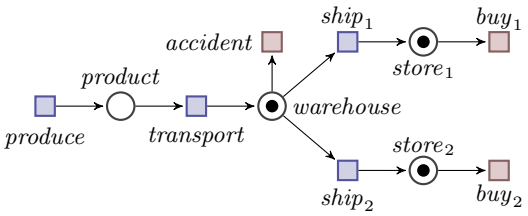


Figure 4.4: A Petri net modeling a supply chain.

We considered the bounded resilience problem for

$$\text{BAD} = \{M \text{ marking } : M(\mathit{warehouse}) = 0 \text{ or } M(\mathit{store}_i) = 0 \text{ for some } i\},$$
$$\text{SAFE} = \{M \text{ marking } : M(\mathit{warehouse}), M(\mathit{store}_1), M(\mathit{store}_2) \geq 1\}.$$

SAFE is upward-closed by definition. The basis of SAFE is given by the marking $\langle 0, 1, 1, 1\rangle$ representing the number of tokens in the production

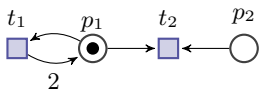
site, warehouse, the first store, and the second store. BAD is an anti-ideal since it is the complement of SAFE (Fact 2.8). It is decidable by definition. We can answer the raised question: By Corollary 4.1 (1), we can solve the bounded resilience problem and, by Corollary 4.2 (1), we can solve the explicit resilience problem. There exists a bound k s.t. the finite-marked Petri net is k-step resilient w.r.t. BAD and SAFE. In fact, the minimal bound is $k_{\min} = 8$. For the computations, see Appendix C.

Remark. The finite-marked Petri net in Figure 4.4 is lossy.

Infinite Set of Initial States

One may ask for the decidability of the bounded and the explicit resilience problem in the case where INIT is infinite and ideal-based. This case, is, in general, not captured in Theorem 4.1 and 4.2. The following example shows that k-step resilience w.r.t. a finite set INIT can be satisfied while it is not satisfied w.r.t. the upward-closure of INIT.

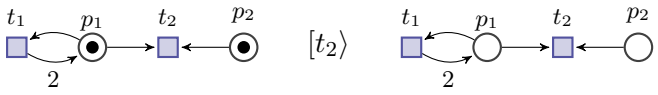
Example 4.4. Consider the following finite-marked Petri net with two places p_1, p_2 and two transitions t_1, t_2.



The transition t_1 generates tokens in the place p_1 provided that there exists a token in p_1. The transition t_2 deletes a token in each of both places. Let

$$\text{BAD} = \{M \text{ marking } : M(p_1) < 2\},$$
$$\text{SAFE} = \{M \text{ marking } : M(p_1) \geq 2\}.$$

For $\text{INIT} = \{\langle 1, 0\rangle\}$, the finite-marked Petri net is 1-step resilient w.r.t. BAD and SAFE. One may ask whether this still holds if we include all markings "larger" than $\langle 1, 0\rangle$. For $\text{INIT} = \{M \text{ marking } : M(p_1) \geq 1\}$, there exists no k s.t. the marked Petri net is k-step resilient w.r.t. BAD and SAFE: we can reach a deadlock from the marking $\langle 1, 1\rangle \in \text{INIT}$.



If INIT is an ideal with a given basis, a statement of the flavour of Theorem 4.1 and 4.2 with adapted requirements holds.

A marked WSTS $\langle S, \leq, \to, \text{INIT}\rangle$ is called *$\mathcal{I}$-marked* if $\text{INIT} \in \mathcal{I}$. For the requirements, we regard computability of a basis of $\uparrow\text{post}^*(\text{INIT})$ where $\text{INIT} \in \mathcal{I}$, lossiness, $\bot$-boundedness, and "weak intersection-effectiveness", i.e., decidability of non-emptiness of $\text{INIT} \cap I$ for every $I \in \mathcal{I}$.

Definition 4.4 (requirements for INIT $\in \mathcal{I}$**).** An $\mathcal{I}$-marked well-structured transition system $\langle S, \leq, \to, \text{INIT}\rangle$ is

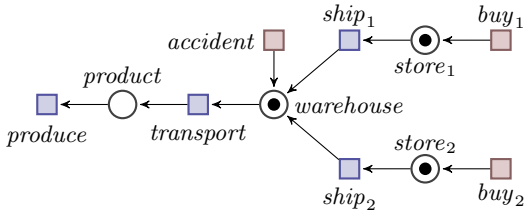
(1) *weakly invertible* if a basis of $\uparrow\text{post}^*(\text{INIT})$ is computable,

(2) *weakly* $\cap$*-effective* if it is decidable whether $\text{INIT} \cap I = \varnothing$ for every $I \in \mathcal{I}$.

Weak invertibility of $\mathcal{I}$-marked WSTSs is the pendant to post*-effectiveness of finite-marked WSTSs. It can be shown that $\mathcal{I}$-marked Petri nets are weakly invertible. This is achieved by inverting the direction of the firing (transition relation). The result is again a Petri net. By Lemma 2.3, the basis of $\text{pre}^*(\uparrow\text{INIT})$ in the inverted Petri net is computable, which is the basis of $\uparrow\text{post}^*(\uparrow\text{INIT})$ in the original Petri net.

Weak $\cap$-effectiveness generalizes $\cap$-effectiveness which means that a basis of $I \cap I'$ is computable for all ideals I, I' with given bases, see, e.g., Abdulla et al. [AČJT96, p. 318]. This is rather a property of the state set than of the WSTS. For ideals of markings, the basis of the intersection is computable by using their representation as tuples of natural numbers.

Example 4.5. Consider the finite-marked Petri net of the supply chain (Figure 4.4). To compute the basis of $\uparrow \text{post}^*(\text{INIT})$ with INIT as the upward-closure of the marking $\langle 0, 1, 1, 1\rangle$, we invert the Petri net. The result is the Petri net represented below.



The basis of the intersection of the upward-closure of the single marking $\langle 0, 1\rangle$ with the upward-closure of $\langle 2, 0\rangle$ consists of the single marking $\langle 2, 1\rangle$.

For the bounded resilience problem for $\mathcal{I}$-marked SWSTSs, – by replacing the requirements – we obtain a result of the flavour of Theorem 4.1. More precisely, we replace

- finite-marked by $\mathcal{I}$-marked,
- post*-effective by weakly invertible,
- lossy by lossy and weakly $\cap$-effective,

and add weakly invertible in case (4).

Theorem 4.3 (bounded resilience for $\mathcal{I}$-marked SWSTSs).
The bounded resilience problem is decidable for

(1) weakly invertible $\mathcal{I}$-marked SWSTSs if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy, weakly $\cap$-effective $\mathcal{I}$-marked SWSTSs if BAD, SAFE $\in \mathcal{I}$,

(3) lossy, weakly $\cap$-effectve, $\bot$-bounded $\mathcal{I}$-marked SWSTS$^{\mathcal{B}}$s if BAD $\in \mathcal{I}$, SAFE $\in \mathcal{J}$,

(4) weakly invertible, $\bot$-bounded $\mathcal{I}$-marked SWSTS$^{\mathcal{B}}$s if BAD, SAFE $\in \mathcal{J}$.

Remark. The difference to the proof of Theorem 4.1 is the computation of a finite respresentation of post*(INIT) $\cap$ BAD. We obtain a statement of the flavour of Lemma 4.2. For the full proof, see Appendix B.

For the explicit resilience problem for $\mathcal{I}$-marked SWSTSs, – by replacing the requirements – we obtain a result of the flavour of Theorem 4.2.

Theorem 4.4 (explicit resilience for $\mathcal{I}$-marked SWSTSs).
The explicit resilience problem is decidable for

(1) weakly invertible $\mathcal{I}$-marked SWSTSs if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy, weakly $\cap$-effectve $\mathcal{I}$-marked SWSTSs if BAD, SAFE $\in \mathcal{I}$.

Proof. As in the proof of Theorem 4.2, we derive the decidability of the explicit resilience problem from the decidability of the bounded resilience problem (Theorem 4.3). □

Remark. For the case that INIT is a decidable anti-ideal, one could obtain a result of the flavour of Theorem 4.3 and 4.4. However, it is unclear how to compute a basis of $\uparrow$post*(INIT) for a decidable anti-ideal INIT (except using the reachability of basis elements).

Concluding: This section provided a generic method for solving the bounded and explicit resilience problem for subclasses of marked WSTSs (in particular, for reset Petri nets) and ideal-based sets BAD and SAFE.

4.3 Algorithms and Approximations

From a more practical perspective, one is not only interested in decidability but also in the algorithms. The proof of Theorem 4.1 includes a decidability algorithm which can be used for automatic verification of resilience. An overview is given in Figure 4.5. In the first step, a finite representation finRepr of post*(INIT) $\cap$ BAD is computed. The requirement for BAD $\in \mathcal{J}$ is post*-effectiveness and the requirement for BAD $\in \mathcal{I}$ is lossiness. In the second step, inclusion of finRepr in the of predecessors of SAFE is checked.

Here, the algorithm splits in the cases SAFE $\in \mathcal{I}$ and SAFE $\in \mathcal{J}$. For SAFE $\in \mathcal{I}$, in the case that there exists a bound k, the algorithm returns also the minimal k, called $k_{\min}$. For SAFE $\in \mathcal{J}$, the algorithm decides the bounded resilience problem without returning a k. The requirement for SAFE $\in \mathcal{J}$ is $\bot$-boundedness.

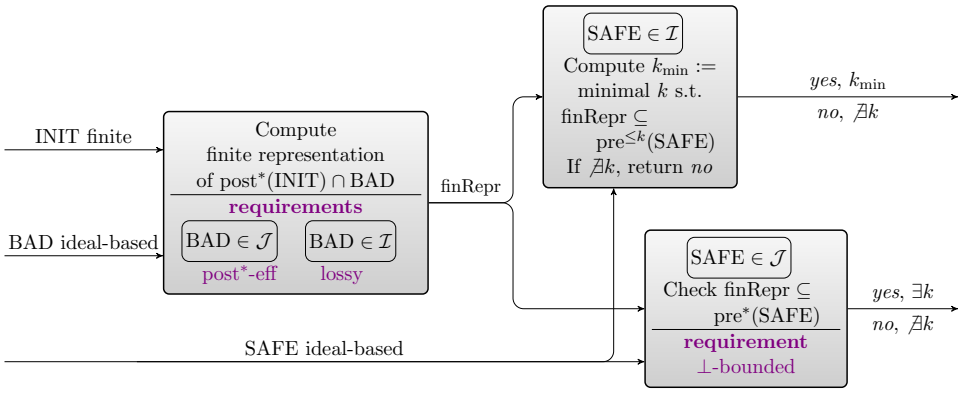


Figure 4.5: Decidability algorithm for the bounded resilience problem.

The decidability algorithm for the explicit resilience problem in Figure 4.6 works for the case SAFE $\in \mathcal{I}$. It can be derived from the proof of Theorem 4.2 and uses the decidability algorithm for the bounded resilience problem. More precisely, in the case SAFE $\in \mathcal{I}$, the decidability algorithm for the bounded resilience problem delivers also $k_{\min}$. Given any k, the algorithm checks whether $k_{\min} \leq k$ and returns the answer accordingly. If the decidability algorithm for the bounded resilience problem detects that there is no such k, the decidability algorithm for the explicit resilience problem returns *no*.

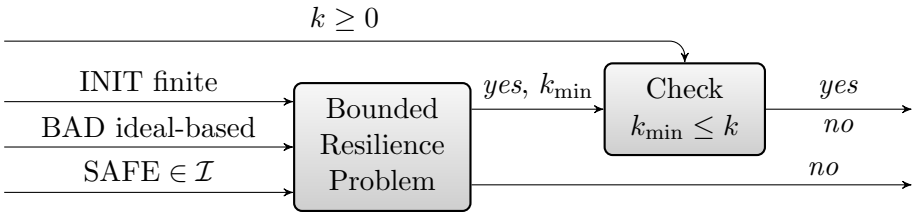


Figure 4.6: Decidability algorithm for the explicit resilience problem.

In Figure 4.7, the second step for the case SAFE $\in \mathcal{I}$ is depicted in more detail. Given a finite representation finRepr and a basis of SAFE, it is first checked whether finRepr is included in pre$^{\leq k}$(SAFE). If so, the algorithm returns *yes* and the current k as $k_{\min}$. Otherwise, it is checked whether pre$^{\leq k+1}$(SAFE) is included in pre$^{\leq k}$(SAFE) using the equation pre$^{\leq k+1}$(SAFE) = SAFE $\cup$ pre(pre$^{\leq k}$(SAFE)). If so, the algorithm returns

no meaning that there exists no such k. Otherwise, k is increased and the iteration continues.

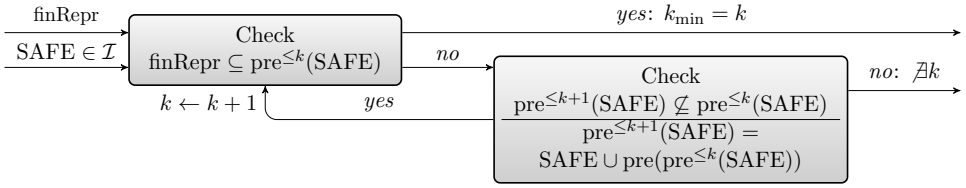


Figure 4.7: Checking inclusion for SAFE $\in \mathcal{I}$.

We give an algorithmic description in pseudocode.

In Algorithm 1, the decidability algorithm for the bounded resilience problem, respectively is depicted. It uses two main procedures: FINREPR which returns a finite representation of post*(INIT) $\cap$ BAD and CHECKINC which tests inclusion in pre*(SAFE) or iteratively in pre$^{\leq k}$(SAFE) for an increasing $k \geq 0$.

Algorithm 1 Decidability Algorithm (Bounded Resilience)

Input: $\langle S, \leq, \rightarrow, \text{INIT}\rangle$, ideal-based sets BAD, SAFE
Output: $k_{\min}$ (minimal k)/ true/ false

```
1: procedure DecideRes(⟨S, ≤, →, INIT⟩, BAD, SAFE)    ▷ decide bounded resilience
2:     return CheckInc(FinRepr(INIT, BAD), SAFE)
3: end procedure
```

Algorithm 2 depicts the algorithm for the explicit resilience problem.

Algorithm 2 Decidability Algorithm (Explicit Resilience)

Input: $\langle S, \leq, \rightarrow, \text{INIT}\rangle$, ideal-based set BAD, SAFE $\in \mathcal{I}$, $k \geq 0$
Output: true/ false

```
1: procedure DecideRes(⟨S, ≤, →, INIT⟩, BAD, B_SAFE, k)   ▷ decide explicit resilience
2:     if CheckInc(FinRepr(INIT, BAD), B_SAFE) = false then          ▷ SAFE ∈ 𝓘
3:         return false
4:     else
5:         if CheckInc(FinRepr(INIT, BAD), B_SAFE) ≤ k then
6:             return true
7:         else
8:             return false
9:         end if
10:    end if
11: end procedure
```

Algorithm 3 depicts the computation of a finite representation of $\text{post}^*(\text{INIT}) \cap \text{BAD}$. In the case $\text{BAD} \in \mathcal{J}$, by post^*-effectiveness, we can use a procedure PostBasis for computing a basis of $\uparrow\text{post}^*(\text{INIT})$. In the case $\text{BAD} \in \mathcal{I}$, we compute $\downarrow\text{post}^*(\text{INIT}) \cap \mathcal{B}_{\text{BAD}}$ where $\mathcal{B}_{\text{BAD}}$ is a basis of BAD. This is expedient by lossiness. The computation of $\downarrow\text{post}^*(\text{INIT}) \cap \mathcal{B}_{\text{BAD}}$ is based on the ideal reachability algorithm [AČJT96]. The procedure Min computes a basis of $\uparrow A$ for a given finite set of states A by minimizing A.

Algorithm 3 Computation of a finite representation

Input: INIT (finite set of states), ideal-based set BAD
Output: finRepr (finite representation)

```
 1: procedure FinRepr(INIT, BAD)                          ▷ for the case BAD ∈ 𝒥
 2:     return PostBasis(INIT) ∩ BAD                      ▷ PostBasis(INIT) computes a basis of
    ↑post*(INIT)
 3: end procedure
 4:
 5: procedure FinRepr(INIT, 𝓑_BAD)  ▷ for the case BAD ∈ ℐ; 𝓑_BAD is a basis of BAD
 6:     finRepr ← ∅
 7:     for b ∈ 𝓑_BAD do                                  ▷ compute ↓post*(INIT) ∩ 𝓑_BAD
 8:         𝓑_1 ← {b}                                     ▷ basis of the current pre^≤k(↑{b})
 9:         𝓑_2 ← ∅                                       ▷ basis of the current pre^≤k+1(↑{b})
10:         while true do                                 ▷ check whether INIT ∩ pre*(↑{b}) ≠ ∅
11:             if INIT∩ ↑𝓑_1 ≠ ∅ then
12:                 finRepr ← finRepr ∪ {b}               ▷ ↑{b} is reachable from INIT
13:                 break
14:             else
15:                 𝓑_2 ← {b} ∪ PreBasis(𝓑_1)             ▷ PreBasis(𝓑_1) computes a basis of ↑𝓑_1
16:                 𝓑_2 ← Min(𝓑_2)                        ▷ Min(𝓑_2) minimizes the set 𝓑_2
17:                 if 𝓑_2 ⊆↑𝓑_1 then
18:                     break                             ▷ ↑{b} is not reachable from INIT
19:                 else
20:                     𝓑_1 ← 𝓑_2                         ▷ continue
21:                 end if
22:             end if
23:         end while
24:     end for
25:     return finRepr
26: end procedure
```

By Proposition 4.2, we can use lossiness or $\bot$-boundedness also to derive a procedure PostBasis for computing a basis of $\uparrow \mathrm{post}^*(\mathrm{INIT})$. In both cases, the procedure is the same. We require a basis $\mathcal{B}_S$ of the whole state set S. In Algorithm 4, this procedure is depicted. It makes use of the ideal reachability algorithm [AČJT96].

Algorithm 4 Computation of a Basis of $\uparrow \mathrm{post}^*(s)$ using a Basis of S

Input: s (state), $\mathcal{B}_S$ (basis of S)
Output: postBasis

```
 1: procedure PostBasis(s, B_S)                    ▷ returns of a basis of ↑post*(s)
 2:     postBasis ← ∅
 3:     for b ∈ B_S do
 4:         B_1 ← {b}                              ▷ basis of the current pre^≤k(↑{b})
 5:         B_2 ← ∅                                ▷ basis of the current pre^≤k+1(↑{b})
 6:         while true do                          ▷ check whether s ∈ pre*(↑{b})
 7:             if s ∈↑B_1 then
 8:                 postBasis ← postBasis ∪ {b}    ▷ ↑{b} is reachable from s
 9:                 break
10:             else
11:                 B_2 ← {b} ∪ PreBasis(B_1)      ▷ PreBasis(B_1) computes a basis of ↑B_1
12:                 B_2 ← Min(B_2)                 ▷ Min(B_2) minimizes the set B_2
13:                 if B_2 ⊆↑B_1 then
14:                     break                      ▷ ↑{b} is not reachable from s
15:                 else
16:                     B_1 ← B_2                  ▷ continue
17:                 end if
18:             end if
19:         end while
20:     end for
21:     return postBasis
22: end procedure
```

The procedure CHECKINC of Algorithm 5 tests inclusion in $\mathrm{pre}^*(\mathrm{SAFE})$ (in $\mathrm{pre}^{\leq k}(\mathrm{SAFE})$ for an increasing $k \geq 0$, and returns the minimal k if it exists). In the case $\mathrm{SAFE} \in \mathcal{I}$, inclusion is checked iteratively and hence, the minimal k is returned if it exists. This procedure (in this case) is based on the ideal reachability algorithm [AČJT96]. In the case $\mathrm{SAFE} \in \mathcal{J}$, the procedure only decides whether there exists such a k. Here we use that $\bot$-boundedness implies post*-effectiveness provided that a basis of all states is given (cp. Proposition 4.2).

Algorithm 5 Checking Inclusion (Minimal k)

Input: finRepr (finite representation), ideal-based set SAFE
Output: $k_{\min}$ (minimal k)/ true/ false

```
1: procedure CheckInc(finRepr, B_SAFE)   ▷ for the case SAFE ∈ I; returns k_min/false
2:     k ← 0                                                    ▷ increasing counter
3:     B_1 ← B_SAFE ▷ basis of the current pre^≤k(SAFE); B_SAFE is a given basis of SAFE
4:     B_2 ← ∅                                  ▷ basis of the current pre^≤k+1(SAFE)
5:     while true do
6:         if finRepr ⊆ ↑B_1 then
7:             return k                                              ▷ we found k_min
8:         else
9:             B_2 ← B_SAFE ∪ PreBasis(B_1)      ▷ PreBasis(B_1) computes a basis of ↑B_1
10:            B_2 ← Min(B_2)                          ▷ Min(B_2) minimizes the set B_2
11:            if B_2 ⊆ ↑B_1 then
12:                return false                                  ▷ there exists no such k
13:            else
14:                B_1 ← B_2                                                ▷ continue
15:                k ← k + 1
16:            end if
17:        end if
18:    end while
19: end procedure
20:
21: procedure CheckInc(finRepr, SAFE)   ▷ for the case SAFE ∈ J; returns true/false
22:    for s ∈ finRepr do
23:        if PostBasis(s, B_S) ∩ SAFE = ∅ then
24:                                          ▷ PostBasis(s, B_S) computes a basis of
25:            return false                  ▷ ↑post*(s) using a given basis B_S of S
26:        end if
27:    end for
28:    return true
29: end procedure
```

Complexity

The algorithms in Section 4.3 exploit the fact that every infinite, ascending sequence of ideals become stationary (cp. Lemma 2.2). Their complexity highly depends on the k for which the considered sequence of ideals becomes stationary. **Schmitz & Schnoebelen** [SS13] obtain non-primitive recursive upper bounds on the k (on the length of sequences without increasing pairs). Their general assumption is that the well-structured transition system cannot "jump" to states of arbitrary "size". They also identify principles to obtain similar lower bounds in the sense of hardness results for the termination and the coverability problem.

Despite all that, in specific (graph transformation) systems, the complexity of the coverability (ideal reachability) problem is often manageable, see **Stückrath** [Stü16, p. 16 and Chapter 7]. The computations we performed are feasible without complications on a standard device, see Appendix C.

Another issue is that the post*-effectiveness of Petri nets relies on the reachability problem for Petri nets. The complexity of the reachability problem for Petri nets has non-primitive recursive upper bounds and non-elementary lower bounds, see, e.g., **Czerwiński et al.** [CLL$^+$21]. This establishes that, for Petri nets, the reachability problem is much harder than the coverability problem.

In conclusion, the complexity of the algorithms in Section 4.3 highly depends on the concrete system and the input. However, we can distinguish between two kinds of procedures: forward and backward procedures. The complexity of our backward procedures, i.e., ideal reachability, seems to be more manageable. Our forward procedures use the post*-effectiveness or lossiness of a well-structured transition system. In the case of lossiness or in the case where post*-effectiveness is established by the reachability of basis elements (cp. Proposition 4.2), one uses, in fact, also a backward procedure. Proper forward procedures only occur when using the post*-effectiveness of Petri nets. The complexity of the latter may be considerable.

Approximation

In our setting, proper forward procedures are inequally costly compared to backward procedures. The computation of a basis of $\uparrow \mathrm{post}^*(\mathrm{INIT})$ is of considerable complexity for Petri nets since it relies on the reachability algorithm for Petri nets. As suggested by König[10], approximations are a way to handle costly computations. We discuss which approximations for k-step resilience are feasible in the case where $\mathrm{BAD} \in \mathcal{J}$ and $\mathrm{SAFE} \in \mathcal{I}$.

Assumption. Let $\langle S, \leq, \to, \mathrm{INIT} \rangle$ be a marked SWSTS, $\mathrm{BAD} \in \mathcal{J}$, and $\mathrm{SAFE} \in \mathcal{I}$.

[10] Private communication at Graph Computation Models 2021.

The so-called μ-function approximates the minimal number of steps to reach a SAFE state whenever a BAD state is reached. It is defined for all subsets of states.

Definition 4.5 (μ-function). The function $\mu : \mathfrak{P}(S) \to \mathbb{N} \cup \{\infty\}$ from the power set of S into the natural numbers extended by the symbol ∞ is given by $\mu(A) = \min(\{k \in \mathbb{N} : A \cap \mathrm{BAD} \subseteq \mathrm{pre}^{\leq k}(\mathrm{SAFE})\} \cup \{\infty\})$. Let $k^{\ell}_{\mathrm{un}} = \mu(\mathrm{post}^{\leq \ell}(\mathrm{INIT}))$ for every $\ell \geq 0$, $k_{\mathrm{ov}} = \mu(\mathrm{post}^{*}(\uparrow \mathrm{INIT}))$, and $k_{\mathrm{min}} = \mu(\mathrm{post}^{*}(\mathrm{INIT}))$.

$k_{\mathrm{min}} = \infty$ means that there exists no k s.t. the marked SWSTS is k-step resilient w.r.t. BAD and SAFE. By definition, the μ-function is monotonic and, by Lemma 4.1, it respects the upward-closure.

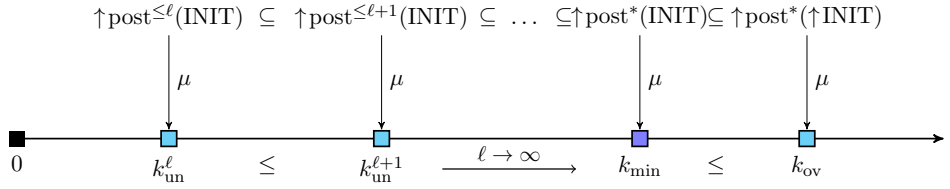
Fact 4.1. For subsets $A \subseteq A' \subseteq S$, $\mu(A) \leq \mu(A')$ and $\mu(\uparrow A) = \mu(A)$.

While the under-approximation is feasible for finite-branching SWSTSs, the over-approximation is feasible for "weakly invertible" finite-marked SWSTSs. A finite-marked WSTS $\langle S, \leq, \rightarrow, \mathrm{INIT}\rangle$ is *weakly invertible* if a basis of $\uparrow \mathrm{post}^{*}(\uparrow \mathrm{INIT})$ is computable. By Fact 4.1, we obtain the following approximations for k-step resilience.

Proposition 4.4 (approximation). Let $\langle S, \leq, \rightarrow, \mathrm{INIT}\rangle$ be a marked SWSTS, $\mathrm{BAD} \in \mathcal{J}$, and $\mathrm{SAFE} \in \mathcal{I}$.

(1) $k^{\ell}_{\mathrm{un}} \leq k^{\ell+1}_{\mathrm{un}} \leq k_{\mathrm{min}}$ for every $\ell \geq 0$ and the sequence $k^{0}_{\mathrm{un}}, k^{1}_{\mathrm{un}}, \ldots$ converges to k_{min} eventually stabilizing. For finite-marked finite-branching SWSTSs, k^{ℓ}_{un} is computable for every $\ell \geq 0$.

(2) $k_{\mathrm{ov}} \geq k_{\mathrm{min}}$ and for weakly invertible finite-marked SWSTSs, k_{ov} is computable.



Proof. (1) By Lemma 2.2, the sequence $\uparrow \mathrm{INIT} \subseteq \uparrow \mathrm{post}^{\leq 1}(\mathrm{INIT}) \subseteq \ldots$ becomes stationary, i.e., there exists $\ell_0 \geq 0$ (minimal) s.t. $\uparrow \mathrm{post}^{*}(\mathrm{INIT}) = \uparrow \mathrm{post}^{\leq \ell_0}(\mathrm{INIT})$. By Fact 4.1, $k^{\ell}_{\mathrm{un}} = \mu(\uparrow \mathrm{post}^{\leq \ell}(\mathrm{INIT})) \leq \mu(\mathrm{post}^{\leq \ell+1}(\mathrm{INIT})) = k^{\ell+1}_{\mathrm{un}}$ and $k^{\ell}_{\mathrm{un}} = \mu(\uparrow \mathrm{post}^{\leq \ell}(\mathrm{INIT})) \leq \mu(\mathrm{post}^{\leq \ell_0}(\mathrm{INIT})) = k_{\mathrm{min}}$ for every $\ell \geq 0$. Thus, the sequence $k^{1}_{\mathrm{un}}, k^{2}_{\mathrm{un}}, \ldots$ converges to k_{min} eventually stabilizing. Since the SWSTS is finite-branching, we can compute a basis of $\uparrow \mathrm{post}^{\leq \ell}(\mathrm{INIT})$ for every $\ell \geq 0$. By the proof of Theorem 4.1 (1), we can

compute k^{ℓ}_{un} for every $\ell \geq 0$.
Statement (2) follows similarly: by definition of weak invertibility, Fact 4.1, and the proof of Theorem 4.1 (1). □

Weak invertibility for Petri nets is achieved by inverting the direction of the firing.

Fact 4.2. Finite-marked Petri nets are weakly invertible.

As a consequence, we can perform under- and over-approximations for finite-marked Petri nets.

Corollary 4.3 (approximation for Petri nets). For finite-marked Petri nets, k_{ov} and k^{ℓ}_{un} are computable for every $\ell \geq 0$.

Example 4.6. For the supply chain (Example 4.1), the over-approximation yields $k_{\text{ov}} = 8$. This implies that $k_{\min} \leq 8$. In particular, $k_{\min} < \infty$ which means that there exists such a k. Thus, we solved the bounded resilience problem for the supply chain. The under-approximations yield $k^0_{\text{un}} = 0$, $k^1_{\text{un}} = 3$, $k^2_{\text{un}} = 6$, $k^3_{\text{un}} = 8$. This implies that $k_{\min} \geq 8$. We conclude that $k_{\min} = 8$, i.e., the over(under)-approximation is exact (after 3 steps). Thus, we solved the explicit resilience problem for the supply chain.

4.4 Related Concepts

The concept of resilience is broadly used in different areas with varying definitions, ee, e.g., **Trivedi et al.** [TKG09] **Jackson & Ferris** [JF13]. Our interpretation of resilience captures recovery in a bounded number of steps. In Akshay et al. [AGH$^+$21], a similar notion of resilience problems for timed automata is investigated. Decidability and complexity results are obtained.

Abdulla et al. [AČJT96] show the decidability of ideal reachability, eventuality properties and simulation in (labeled) strongly well-structured transition systems. We use the presented algorithm as an integrant of our decidability proof. **Finkel & Schnoebelen** [FS01] show that the concept of well-structuredness is ubiquitous in computer science by providing a large class of example models, e.g., Petri nets and their extensions, communicating finite state machines, lossy systems, basic process algebras. Moreover, they give several decidability results for well-structured systems with varying notions of compatibility, e.g., decidability of the termination problem. They also generalize the algorithm of [AČJT96] to not necessarily strongly well-structured transition systems (without labels) to show decidability of the coverability problem.

Karp & Miller [KM69] show the decidability of the ideal reachability problem Petri nets. They construct a coverability tree which gives rise to a "top basis" of the downward-closure of reachable states. **Blondin et**

al. [BFG20] generalize this forward construction to a subclass of strongly well-structured transition systems, concluding that LTL model checking is decidable for this subclass. They show that ω-Petri nets satisfy their requirements. For our purpose, "top bases" do not seem to be useful. Graph transformation systems satisfy upward-compatibility but usually not downward-compatibility which may fit to the notion of a "top basis" of a downward-closed set.

Mayr [May84] shows that the reachability problem is decidable for Petri nets. In his PhD thesis, **Hack** [Hac85] investigates decidability questions for Petri nets. It is shown that submarking reachability is recursively equivalent to reachability. We use the decidability of submarking reachability to show that Petri nets are post*-effective.

4.5 Summary

We provided a systematic investigation on resilience problems obtaining a decidability theorem for subclasses of WSTSs. The key idea is to compute a finite representation of the reachable BAD states for checking inclusion in a predecessor set of the SAFE states. The stated requirements for decidability are, besides strong compatibility, an effectiveness requirement (post*-effective) or a kind of unreliability (lossy, $\bot$-bounded). The requirement of post*-effectiveness is associated with the case that the set BAD is downward-closed. Lossiness is associated with the case that the set BAD is upward-closed. The requirement of $\bot$-boundedness is associated with the case that the set SAFE is downward-closed. As an immediate consequence, we obtain decidability results for reset Petri nets.

Takeaway (Chapter 4).

- The bounded resilience problem for WSTSs asks: Given a WSTS, sets INIT, BAD, and SAFE, is there a bound k s.t. starting from any INITial state, whenever we reach a BAD state, we can reach a SAFE state in $\leq k$ steps? This problem is decidable for WSTSs with strong compatibility satisfying additional requirements: post*-effective, lossy, $\bot$-bounded; INIT finite; BAD, SAFE ideal-based.

BAD decidable anti-ideal	post*-effective
BAD ideal with given basis	lossy
SAFE decidable anti-ideal	$\bot$-bounded

- The explicit resilience problem for WSTSs is formulated as the bounded resilience problem, except that a bound $k \geq 0$ is given. It is decidable for the cases where INIT is finite and SAFE is an ideal. The requirements are: post*-effective, lossy; BAD ideal-based.

Chapter 5

Verifying Resilience of Graph Transformation Systems

In this chapter, we apply the concepts and decidability results from Chapter 4 to graph transformation systems. The basic prerequisite is the well-structuredness of the transition system induced by a set of graphs and a graph transformation system. Well-structuredness of graph transformation systems is investigated by König & Stückrath [KS17] for several well-quasi-orders. The well-quasi-order we use is the subgraph order which permits strong compatibility but comes with the restriction of the boundedness of the path length on the graph class. Boundedness of the path length means that the length of any path in any graph of the class is bounded. Our general theory applies to graph transformation systems of bounded path length satisfying additional requirements.

Our results on strongly well-structured transition systems can be translated into the graph-transformational setting as follows.

marked SWSTS	marked GTS
set of states	set of graphs
well-quasi-order	subgraph order
transition $\rightarrow$	transformation $\Rightarrow_{\mathcal{R}}$
finite set INIT of states	finite set INIT of graphs
ideal-based sets BAD, SAFE	proper constraints Bad, Safe

In the context of graph transformation systems, the states are graphs and the transitions are transformations. We consider the subgraph order as well-quasi-order. At the level of well-structured transition systems, we assumed that BAD and SAFE are ideal-based. Regarding the subgraph order, ideals and anti-ideals of graphs are expressively equivalent to proper constraints, i.e., graph constraints describing the existence and absence of specified subgraphs, respectively. Instead of ideal-based sets, we input proper constraints Bad and Safe.

In Section 5.1, we introduce the resilience problem for graph transformation systems. In Section 5.2, we show the decidability of the resilience problem for subclasses of graph transformation systems of bounded path length. In Section 5.5, we develop sufficient, rule-specific criteria for the applicability of our results.

5.1 Resilience Problem

A marked GTS is a GTS together with a graph class closed under rule application and a subset INIT of graphs. It is *finite-marked* if INIT is finite.

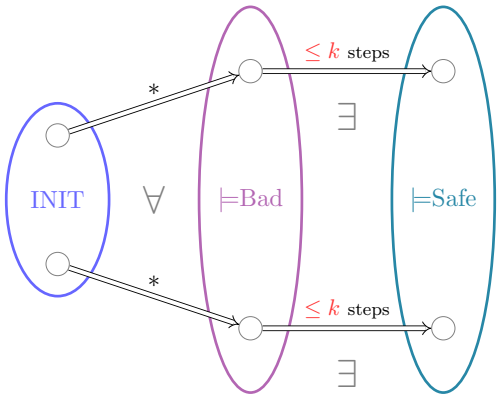
We formulate the bounded resilience problem for marked GTSs.

BOUNDED RESILIENCE PROBLEM FOR GTSS

Given: A marked GTS $\langle S, \mathcal{R}, \text{INIT}\rangle$, graph constraints Bad, Safe.

Question: Does there exist a $k \geq 0$ s.t. starting from any graph in INIT, whenever we reach a Bad graph, we can reach a Safe graph in $\leq k$ steps?[11]

The answer to this question is *yes* if there is a natural number $k \geq 0$ s.t. the marked GTS is *k-step resilient* w.r.t. Bad and Safe, i.e., starting from any graph in INIT, whenever we reach a Bad graph, we can reach a Safe graph in $\leq k$ steps, and *no* otherwise. The constraint Bad specifies which kind reachable graphs are of interest. We do not require that Bad is the negation of Safe.

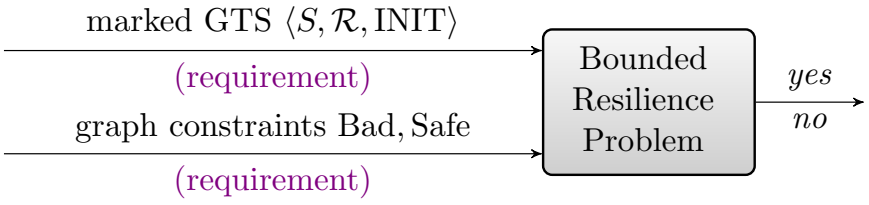


Note: While in Chapter 3, Bad graphs are obtained by application of an environment rule, in this chapter, Bad graphs are defined by a graph constraint. While in Chapter 3, k-step resilience is defined for marked joint GTS, the resilience notion considered in this chapter is a generalization to marked GTSs and arbitrary proper constraints Bad. The relation is as follows.

[11] A Bad (Safe) graph is a graph satisfying the graph constraint Bad (Safe).

Fact 5.1. A marked joint GTS is k-step resilient w.r.t. Safe iff the marked annotated joint GTS is k-step resilient w.r.t. Bad $= \exists(\textcircled{e})$ and Safe.

We are interested in suitable subclasses of marked GTSs and graph constraints Bad and Safe, for which the bounded resilience problem is decidable.



We show that for suitable subclasses of finite-marked GTSs and proper constraints Bad and Safe, the bounded resilience problem is decidable.

Example 5.1 (circular process protocol). To illustrate the idea of our resilience concept, we give an example. Consider a ring of three processes P_0, P_1, P_2 each of which has an unordered waiting collection (multiset) containing commands. Each command belongs to a process and is labeled accordingly as c_0, c_1, or c_2. The protocol is described below.

1. The process P_0 is *liberal*, i.e., it can *initiate* (generate and forward) a command c_0.
2. Every process P_i can *forward* a command c_j, $i \neq j$, not belonging to itself.
3. If a process P_i receives a command c_i, it is *enabled* and can
   - *execute* its specific process action, or
   - *clear* all commands in its waiting collection and forward a command of the next process, or
   - *leave* the process ring (if $i \neq 0$) and forward a command of the next process.

   Afterwards, the command c_i is deleted.
4. Any command may get *lost* in any state.

The process action of P_0 is to forward two commands, c_1 and c_2. The process action of P_1 (P_2) is to forward a command c_2 (c_1). The topology of the process ring changes when a process leaves the ring. Processes P_1 and P_2 can leave the ring only if the other process has not left the ring before. In Figure 5.1, the initial state where every process P_i has one command c_i in its collection and the three possible topologies are shown. A process P_i is represented by an edge labeled with P_i. The collections are represented

by white nodes which may have loops labeled with c_i corresponding to the contained commands.

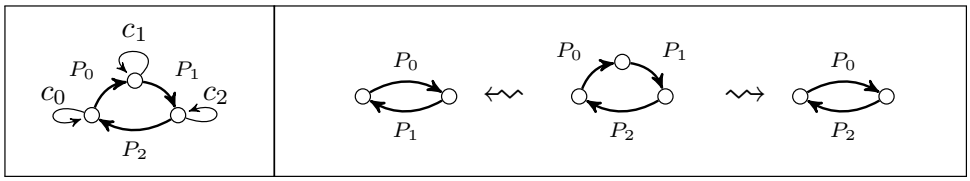


Figure 5.1: Initial state and topologies of the circular process protocol.

A formalization as WSTS induced by a GTS can be found in Figure 5.5. Consider the following instances of the bounded resilience problem with the initial state as in Figure 5.1:

instance	1	2	3	4	5
BAD	¬AllEnabled	Command(c_2)	AllEnabled	NoCommand	¬3Processes
SAFE	AllEnabled	Collection(c_0, c_1)	¬AllEnabled	¬3Processes	3Processes

For every instance of the bounded resilience problem, we are interested in a bound k for the number of steps needed for recovery. In the first instance, we ask whether we can reach a state where every process is enabled in $\leq k$ steps whenever we reach a state where this is not the case. In the second instance, we ask whether we can reach a state with a collection containing commands c_0, c_1 in $\leq k$ steps whenever we reach a state with a collection containing c_2. The third instance is the "dual" problem to the first one where the constraints for BAD and SAFE are exchanged. In the fourth instance, we ask whether we can reach a state not containing three processes in $\leq k$ steps whenever we reach a state without commands. In the fifth instance, we ask whether we can reach a state containing three processes in $\leq k$ steps whenever we reach a state containing not three processes. One may ask:

1. Does such a k exist?

2. Is there a generic method for problems of this kind?

We will answer these questions in this chapter.

5.2 Decidability

We translate the results for strongly well-structured transition systems into the graph-transformational setting. Our approach makes use of the subgraph order.

Convention. We equip every graph class with the subgraph order.

Remark. The subgraph order is anti-symmetric.[12] As a consequence, bases of ideals are unique up to isomorphism. For the subgraph order, we speak of *the* basis of an ideal.

The following result of König & Stückrath terms a sufficient criterion for GTSs to be well-structured. It is essential for our translation. König & Stückrath consider labeled hypergraphs.

Lemma 5.1 (well-structured GTS [KS17, Prop. 7]).

(1) For every graph class S of bounded path length, the restriction of the subgraph order to S is a wqo.

(2) Every GTS together with a graph class closed under rule application induces a strongly compatible transition system.

(3) For every GTS, the basis of $\mathrm{pre}(\uparrow\{G\})$ in the class of all graphs is computable for every given graph G.

In a nutshell, this means that every GTS together with a graph class of bounded path length induces a SWSTS. All graphs satisfying the bounded path length condition are included. We generalize and adapt the result of König & Stückrath for our purpose. In the following table, the conditions to be met for marked SWSTSs and marked GTSs are listed.

marked SWSTS	marked GTS
set of states S	set of graphs S
wqo on S	subgraph order on S
	bounded path length [KS17]
strong compatibility	satisfied, see [KS17]
$\forall s \in S$: basis of $\mathrm{pre}(\uparrow\{s\})$ is computable	pre-eff: $\forall G \in S$: basis of $\mathrm{pre}(\uparrow\{G\})$ is computable
basis of S given	basis of S given
ideal with given basis	c-eff: $\forall$ positive constraint c: basis of $[\![c]\!]_S$ is computable
decidable anti-ideal	$\models$ is decidable for negative constraints

[KS17]

In the following, we restrict the upward-closure and the predecessor set to a graph class S: By $\uparrow_S A$ we denote the upward-closure in S of $A \subseteq S$. By $\mathrm{pre}_S(A)$ we denote the predecessor set in S of $A \subseteq S$. For the class of all graphs, we omit the index.

Definition 5.1 (pre-effective). A marked GTS with graph class S is pre-*effective* if there exists an effective procedure to obtain the basis of $\mathrm{pre}_S(\uparrow_S\{G\})$ for every graph $G \in S$.

[12] anti-symmetric *up to isomorphism*, i.e., $\forall G, G' : (G \leq G' \text{ and } G \geq G')$ implies $G \cong G'$.

Pre-effectiveness corresponds directly to the pre-effectiveness condition for WSTSs. We use the result of König & Stückrath to obtain an induced SWSTS. A prerequisite is the boundedness of path length on the graph class.

Definition 5.2 (marked GTS$^{\mathrm{bp}}$). A marked GTS is *of bounded path length*, shortly a marked GTS$^{\mathrm{bp}}$, if its graph class is of bounded path length.

Example 5.2 (starry sky). The rules $\langle \varnothing \rightharpoonup \textcircled{A} \rangle$ and $\langle 1\textcircled{A} \rightharpoonup 1\textcircled{A}\!\!\rightarrow\!\!\bullet \rangle$ together with the set of disjoint unions of unboundedly many star-shaped graphs and any subset form a marked GTS$^{\mathrm{bp}}$.

Proposition 5.1. Every pre-effective marked GTS$^{\mathrm{bp}}$ induces a marked SWSTS.

Proof. This follows by the definition of SWSTSs (Definition 2.12) and Lemma 5.1 (1) and (2). □

Convention. We say that a pre-effective marked GTS$^{\mathrm{bp}}$ is post*-effective if the induced marked SWSTS is post*-effective. The same convention applies to "lossy" and "$\bot$-bounded".

A *positive*$^+$ constraint is a positive constraint c with a given basis of $[\![c]\!]_S$. If a basis of the graph class is given, we write shortly marked GTS$_{\mathcal{B}}$. By replacing "marked SWSTSs" with "pre-effective marked GTS$^{\mathrm{bp}}$s"in Theorem 4.1, we obtain the following result.

Lemma 5.2. The bounded resilience problem is decidable
for pre-effective finite-marked GTS$^{\mathrm{bp}}$s which are

(1) post*-effective if Bad is negative and Safe is positive$^+$,

(2) lossy if Bad and Safe are positive$^+$,

for pre-effective finite-marked GTS$^{\mathrm{bp}}_{\mathcal{B}}$s which are

(3) lossy and $\bot$-bounded if Bad is positive$^+$ and Safe is negative,

(4) $\bot$-bounded if Bad and Safe are negative.

Proof. By Proposition 5.1, every pre-effective marked GTS$^{\mathrm{bp}}$ induces a marked SWSTSs. For every positive$^+$ (negative) constraint c, $[\![c]\!]_S \in \mathcal{I}$ ($\in \mathcal{J}$). Thus, we can apply Theorem 4.1, setting BAD $= [\![\mathrm{Bad}]\!]_S$ and SAFE $= [\![\mathrm{Safe}]\!]_S$. □

This result is the direct translation of the decidability results for SWSTSs into the graph-transformational setting. However, it has two drawbacks: (i) We do not input solely a positive constraint c but require also a given basis of $[\![c]\!]_S$, and (ii) it does not give us a procedure for pre-effectiveness. To improve this result, we consider requirements on the graph class.

Definition 5.3 (con-effective, ∩-based). A graph class S is *con-effective (constraint-effective)* if the basis of $[\![c]\!]_S$ is computable for every pure positive constraint c. It is *∩-based* if there exist $I \in \mathcal{I}$, $J \in \mathcal{J}$ s.t. $S = I \cap J$ in the class of all graphs.

Since we input proper graph constraints, a reasonable assumption for the graph class is that we can compute the basis of the ideal $[\![c]\!]_S$ for every positive constraint c. However, the definition of a con-effectiveness does not give us a concrete procedure for the computation of the basis of $[\![c]\!]_S$ for a given positive constraint c. Similarly, the definition of pre-effectiveness does not give us a concrete procedure for the computation of the basis of $\mathrm{pre}(\uparrow\{G\})$ for a given graph G. We illustrate what these procedures look like in many use cases by considering ∩-basedness. While pre-effectiveness is a property of a marked GTS, con-effectiveness and ∩-basedness are properties of a graph class. We will show the implications depicted in Figure 5.2.

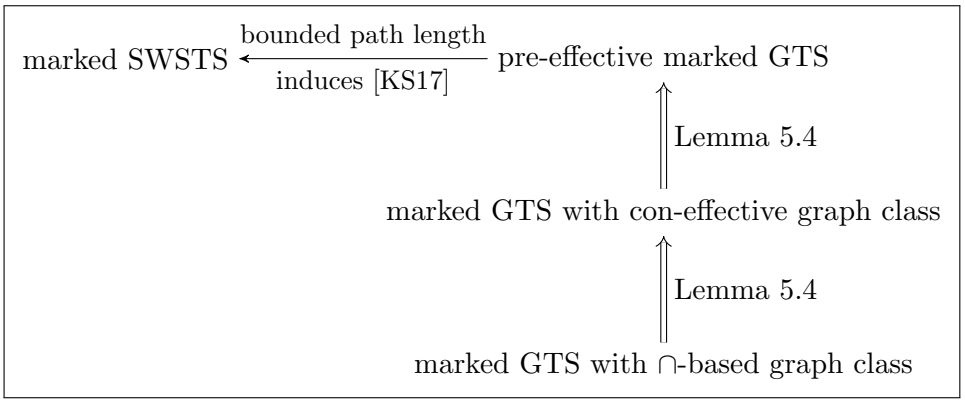


Figure 5.2: Hierarchy of the effectiveness requirements.

First, we show that con-effectiveness is sufficient for computing the basis of $[\![c]\!]_S$ for every positive constraint c.

Lemma 5.3. A graph class S is con-effective iff the basis of $[\![c]\!]_S$ is computable for every positive constraint c.

Proof. "⇒": Let S be a con-effective graph class and c a positive constraint. By Lemma 2.1, we can effectively transform c into an equivalent positive constraint c' of the form $\bigvee_{1\leq i\leq n} \exists C_i$. By con-effectiveness, we compute the basis $\mathcal{B}_i$ of $[\![\exists C_i]\!]_S$. We obtain

$$\uparrow_S \bigcup_{1\leq i\leq n} \mathcal{B}_i = \bigcup_{1\leq i\leq n} \uparrow_S \mathcal{B}_i = \bigcup_{1\leq i\leq n} [\![\exists C_i]\!]_S = [\![\bigvee_{1\leq i\leq n} \exists C_i]\!]_S$$
$$= [\![c']\!]_S = [\![c']\!] \cap S = [\![c]\!] \cap S = [\![c]\!]_S.$$

By Fact 2.6, each basis $\mathcal{B}_i$ is finite. Hence, by Fact 2.6, we can compute the basis of $\uparrow_S \bigcup_{1 \leq i \leq n} \mathcal{B}_i$.
"$\Leftarrow$": Holds since every pure positive constraint is a positive constraint. □

Example 5.3 (con-effective). Let $S^{2\text{lp}}$ be the class of graphs consisting of one node and $2n$ loops for every $n \geq 0$. We show that this graph class is con-effective. For the graph D_2 consisting of two isolated nodes, $[\![\exists D_2]\!]_{S^{2\text{lp}}} = \varnothing$. For every graph $C \geq D_2$, $[\![\exists C]\!]_{S^{2\text{lp}}} \subseteq [\![\exists D_2]\!]_{S^{2\text{lp}}}$. Hence, $[\![\exists C]\!]_{S^{2\text{lp}}} = \varnothing$. Let D_1^n be the graph consisting of one node and n loops. For every $n \geq 0$, the set $\{D_1^{2n}\}$ is the basis of $[\![\exists D_1^{2n}]\!]_{S^{2\text{lp}}}$ and of $[\![\exists D_1^{2n+1}]\!]_{S^{2\text{lp}}}$. The set $\{D_1^0\}$ is the basis of $[\![\exists \varnothing]\!]_{S^{2\text{lp}}}$.

Lemma 5.4 (effectiveness lemma).

(i) Every marked GTS with con-effective graph class is pre-effective.

(ii) Every $\cap$-based graph class is con-effective.

Proof. (i) We show that there exists an effective procedure to obtain the basis of $\text{pre}_S(\uparrow_S \{G\})$ for every graph $G \in S$. König & Stückrath [KS17] give an effective procedure which works in the class of all graphs (cp. Lemma 5.1 (3)). Let $G \in S$ be a graph and $\mathcal{B}_{\text{pre}}$ the basis of $\text{pre}(\uparrow\{G\})$ in the class of all graphs. Let c be the positive constraint $\bigvee_{B \in \mathcal{B}_{\text{pre}}} \exists B$. For every graph H,

$$
\begin{aligned}
& & H \in [\![c]\!]_S & \\
\iff & & H \in S \text{ and } H \models \bigvee_{B \in \mathcal{B}_{\text{pre}}} \exists B & \quad \text{(Def. } [\![\cdot]\!]_S\text{)} \\
\iff & & H \in S \text{ and } \exists B \in \mathcal{B}_{\text{pre}} : H \models \exists B & \quad \text{(Def. } \models\text{)} \\
\iff & & H \in S \text{ and } \exists B \in \mathcal{B}_{\text{pre}} : B \leq H & \quad \text{(Def. } \leq\text{)} \\
\iff & & H \in S \text{ and } H \in \text{pre}(\uparrow\{G\}) & \quad \text{(Def. basis)} \\
\iff & & H \in S \text{ and } \exists G' \in \uparrow\{G\} : H \Rightarrow G' & \quad \text{(Def. pre)} \\
\iff & & H \in S \text{ and } \exists G' : H \Rightarrow G' \geq G & \quad \text{(Def. } \uparrow\text{)} \\
\iff & & H \in S \text{ and } \exists G' \in S : H \Rightarrow G' \geq G & \quad \text{(closed under } \Rightarrow\text{)} \\
\iff & & H \in S \text{ and } \exists G' \in \uparrow_S \{G\} : H \Rightarrow G' & \quad \text{(Def. } \uparrow_S\text{)} \\
\iff & & H \in \text{pre}_S(\uparrow_S \{G\}). & \quad \text{(Def. pre}_S\text{)}
\end{aligned}
$$

Thus, $[\![c]\!]_S = \text{pre}_S(\uparrow_S\{G\})$. By Lemma 5.3 and con-effectiveness, the basis of $[\![c]\!]_S$ is computable.
(ii) Let $S = I \cap J$ be a graph class where $I \in \mathcal{I}$ and $J \in \mathcal{J}$. Let $b = \bigvee_{B \in \mathcal{B}} \exists B$ where $\mathcal{B}$ is the basis of I and c a positive constraint. Then, $b \wedge c$ is a positive constraint. By Lemma 2.1, we can compute a positive constraint c' s.t. $[\![c']\!] = [\![b \wedge c]\!]$ (in the class of all graphs) and c' is of the form $\bigvee_{1 \leq i \leq n} \exists B_i$. Since $J \in \mathcal{J}$, we can compute the set $\{B_i \in J : 1 \leq i \leq n\}$ which is the basis of $[\![c]\!]_S = \{G \in S : G \models c\}$. □

For the case where Bad and Safe are negative, con-effectiveness is not required and we consider a pre-effective marked GTS and a given basis of the graph class, shortly a pre-effective marked GTS$_{\mathcal{B}}$.

Theorem 5.1 (bounded resilience for finite-marked GTSs).
The bounded resilience problem is decidable
for finite-marked GTS$^{\text{bp}}$s which are

(1) post*-effective if Bad is negative and Safe is positive,

for finite-marked GTS$^{\text{bp}}$s with **con-effective** graph class, which are

(2) lossy if Bad and Safe are positive,

(3) lossy and $\perp$-bounded if Bad is positive and Safe is negative,

for pre-**effective** finite-marked GTS$^{\text{bp}}_{\mathcal{B}}$s which are

(4) $\perp$-bounded if Bad and Safe are negative,

and, in particular, for finite-marked GTS$^{\text{bp}}$s with $\cap$**-based** graph class which satisfy the respective requirement in the cases (2), (3), and (4).

Proof. Case (1). We show that (1) holds for post*-effective finite-marked GTS$^{\text{bp}}$s with $\cap$-based graph class. By Lemma 5.4, every finite-marked GTS$^{\text{bp}}$ with $\cap$-based graph class is a pre-effective finite-marked GTS$^{\text{bp}}$ and every $\cap$-based graph class is con-effective. Thus, the basis of $[\![\text{Safe}]\!]_S$ is computable. By Lemma 5.2 (1), the bounded resilience problem is decidable for post*-effective finite-marked GTS$^{\text{bp}}$s with $\cap$-based graph class. For every post*-effective finite-marked GTS$^{\text{bp}}$s with graph class S, we solve the bounded resilience problem for the post*-effective finite-marked GTS$^{\text{bp}}$s with $\cap$-based graph class $\downarrow\text{post}^*(\text{INIT}) = \uparrow\{\varnothing\} \cap \downarrow\text{post}^*(\text{INIT})$. By Lemma 2.3, $\downarrow\text{post}^*(\text{INIT})$ is a decidable anti-ideal. It is of bounded path length since $\text{post}^*(\text{INIT}) \subseteq S$.
Cases (2) & (3). By con-effectiveness, the basis $\mathcal{B}$ of S is computable as the basis of $[\![\exists\varnothing]\!]_S$. Also by con-effectiveness, the basis of $[\![\text{Bad}]\!]_S$ (and in case (3), also the basis of $[\![\text{Safe}]\!]_S$) is computable. By Lemma 5.4, every finite-marked GTS$^{\text{bp}}$ with $\cap$-based graph class is a pre-effective finite-marked GTS$^{\text{bp}}$. By Lemma 5.2 (2) and (3), the bounded resilience problem is decidable for lossy (and $\perp$-bounded) finite-marked GTS$^{\text{bp}}$s with con-effective graph class.
Case (4). This follows by Lemma 5.2 (4). □

Joint GTSs are GTSs and for joint GTSs, Bad = $\exists(\textcircled{\text{e}})$ is fixed. Thus, we obtain the decidability of the bounded resilience problem for finite-marked joint graph transformation systems considered in Chapter 3.

Corollary 5.1 (bounded resilience for finite-marked joint GTSs). The bounded resilience problem with Bad = $\exists(\textcircled{\mathbf{e}})$ is decidable for finite-marked annotated joint GTS$^{\mathrm{bp}}$s which are

(1) post*-effective if Safe is positive,

for pre-**effective** finite-marked annotated joint GTS$^{\mathrm{bp}}_{\mathcal{B}}$s which are

(2) $\bot$-bounded if Safe is negative,

and, in particular, for $\bot$-bounded finite-marked annotated joint GTS$^{\mathrm{bp}}$s with $\cap$**-based** graph class if Safe is negative.

Proof. By construction of joint GTSs, the positive constraint Bad = $\exists(\textcircled{\mathbf{e}})$ is equivalent to the negative constraint $\neg\exists(\textcircled{\mathbf{s}}) \wedge \neg\exists(\textcircled{\top})$. Since marked annotated joint GTSs are marked GTSs, we can apply the cases (1) and (4) of Theorem 5.1. □

Besides the bounded resilience problem, one may be interested in the explicit resilience problem where, additionally, a bound k is given as input. We formulate the explicit resilience problem for marked GTSs.

EXPLICIT RESILIENCE PROBLEM FOR GTSS	
Given:	A marked GTS $\langle S, \mathcal{R}, \mathrm{INIT}\rangle$, graph constraints Bad, Safe, a natural number $k \geq 0$.
Question:	Starting from any graph in INIT, whenever we reach a Bad graph, can we reach a Safe graph in $\leq k$ steps?

We are interested in suitable subclasses of marked GTSs and graph constraints Bad, Safe, for which the explicit resilience problem is decidable.

Theorem 5.2 (explicit resilience for finite-marked GTSs). The explicit resilience problem is decidable for finite-marked GTS$^{\mathrm{bp}}$s which are

(1) post*-effective if Bad is negative and Safe is positive,

for finite-marked GTS$^{\mathrm{bp}}$s with **con-effective** graph class, which are

(2) lossy if Bad and Safe are positive,

and, in particular, for lossy finite-marked GTS$^{\mathrm{bp}}$s with $\cap$**-based** graph class in the case (2).

Proof. We derive the decidability of the explicit resilience problem from the decidability of the bounded resilience problem (Theorem 5.2). The argumentation is as in the proof of Theorem 4.2. □

Since joint GTSs are GTSs, we obtain the decidability of the explicit resilience problem for finite-marked joint graph transformation systems considered in Chapter 3.

Corollary 5.2 (explicit resilience for finite-marked joint GTSs). The bounded resilience problem with Bad = $\exists(\textcircled{e})$ is decidable
for finite-marked annotated joint GTS$^{\text{bp}}$s which are

post*-effective if Safe is positive,

and, in particular, for post*-effective finite-marked annotated joint GTS$^{\text{bp}}$s with **$\cap$-based** graph class if Safe is positive.

Example 5.4 (path game). Consider two fixed locations represented by nodes labeled with L. The middle points are represented by black nodes. The system tries to construct directed paths of lenght 2 between them (one path forth and one back) using the rules in Figure 5.3.

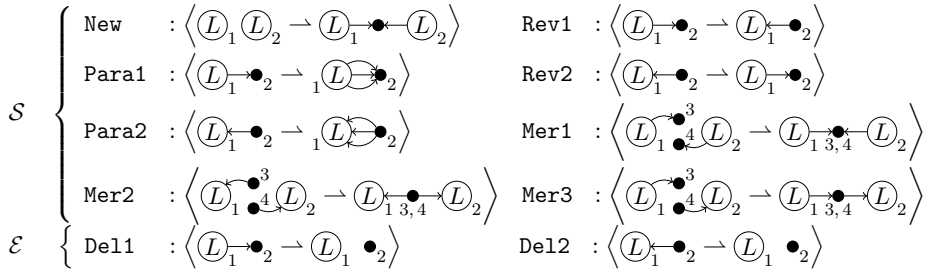
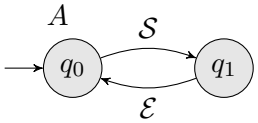


Figure 5.3: System and environment of the path game.

The system can (i) create a new middle point connected to the locations by the `New`-rule, (ii) create two parallel edges provided that there is one by the `Para`-rules, (iii) reverse the direction of an edge by the `Rev`-rules, and (iv) merge two middle points each of which are connected to a different location by the `Mer`-rules. The environment deletes an edge. The control automaton is alternating, i.e., given by:

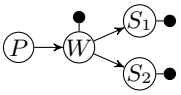


The joint GTS consists of enriched system rules of the form $\langle L^{q_1} \rightharpoonup R^{q_0} \rangle$ and enriched environment rules of the form $\langle L^{q_0} \rightharpoonup R^{q_1} \rangle$. We consider the INITial graph $\textcircled{L}\!\!\rightarrow\!\!\bullet\!\!\leftarrow\!\!\textcircled{L}$ $\textcircled{q_0}$. The positive constraint Safe is given by

$$\text{Safe} = \exists(\textcircled{L}\,\bullet\,\textcircled{L}) \vee \exists(\textcircled{L}\,\bullet\,\textcircled{L}).$$

The graph class consists of all graphs with exactly two locations and arbitrarily many middle points connected to the locations by arbitrary edges. The graphs are also equipped with a node labeled with q_0 or q_1. This graph class is of path length ≤ 2. We can reach the graph $\textcircled{L}\,\textcircled{L}\,\textcircled{q_0}$ (modulo isolated black nodes) when the system is only changing the direction of edges. Hence, the finite-marked joint GTS^{bp} is post*-effective. By Corollary 5.1 and 5.2, we can solve both resilience problems: There exists a bound k s.t. the finite-marked joint GTS^{bp} is k-step resilient w.r.t. Safe. In fact, the minimal bound is $k_{\min} = 13$. For the computations, see Appendix C.

Example 5.5 (supply chain II). Consider again the simplified scenario of a supply chain, depicted in Example 4.1. We model the latter example by a marked joint GTS. The topology is given in the following INITial graph:



A production site (P) is connected to a warehouse (W) which again is connected to two stores S_1 and S_2. Each of the black nodes indicates one product at the corresponding connected location. The behavior in this supply chain is modeled by the rules in Figure 5.4. The system rules consist of `Pr` (the completion of a product at the production site P), `Tr` (transporting a product from P to the warehouse W), and $\texttt{Sh}_1$ and $\texttt{Sh}_2$ (shipping a product from W to one of the two stores S_1, S_2). The environment rules describe adverse effects: `Ac` describes an accident in the warehouse which leads to the loss of one product, and $\texttt{Bu}_1$, $\texttt{Bu}_2$ describe that a product is bought from S_1 or S_2, respectively.

$$\mathcal{S}\left\{\begin{array}{ll} \texttt{Pr}: & \langle 1(P) \rightharpoonup 1(P)\bullet\rangle \\ \texttt{Tr}: & \langle 3\bullet(P)_1\!\to\!(W)_2 \rightharpoonup (P)_1\!\to\!(W)_2\bullet 3\rangle \\ \texttt{Sh}_1: & \langle 3\bullet(W)_1\!\to\!(S_1)_2 \rightharpoonup (W)_1\!\to\!(S_1)_2\bullet 3\rangle \\ \texttt{Sh}_2: & \langle 3\bullet(W)_1\!\to\!(S_2)_2 \rightharpoonup (W)_1\!\to\!(S_2)_2\bullet 3\rangle \end{array}\right., \mathcal{E}\left\{\begin{array}{ll} \texttt{Ac}: & \langle 1(W)\bullet \rightharpoonup 1(W)\rangle \\ \texttt{Bu}_1: & \langle 1(S_1)\bullet \rightharpoonup 1(S_1)\rangle \\ \texttt{Bu}_2: & \langle 1(S_2)\bullet \rightharpoonup 1(S_2)\rangle \end{array}\right.$$

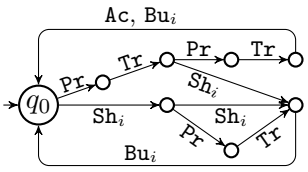


Figure 5.4: System, environment, and control automaton of supply chain II.

The control automaton in Figure 5.4 describes the possible order of rule applications. We are interested in the question when the product is again "in stock", i.e., at least 1 product in the warehouse and in each of both stores, whenever a customer buys a product or when an accident in the warehouse happens. After each such step, the automaton is in the state q_0. Formally, we consider Bad = $\exists(\textcircled{\mathbf{e}})$ ("the last applied rule was an environment rule"), as usual for marked joint GTSs, and the positive constraint

$$\text{Safe} = \exists(\textcircled{W}\text{-}\bullet) \wedge \exists(\textcircled{S_1}\text{-}\bullet) \wedge \exists(\textcircled{S_2}\text{-}\bullet).$$

We ask whether there exists a bound k s.t. starting from the INITial graph, whenever an environment rule was applied, we can reach a graph satisfying Safe in $\leq k$ steps. As $\cap$-based graph class of bounded path length we consider all graphs of the same topology as the INITial graph, with an arbitrary number of products at the warehouse and the stores. One can show that the finite-marked (annotated) joint GTS^{bp} is post*-effective, e.g., by reducing the respective computation to the post*-effectiveness of finite-marked Petri nets (Example 4.2). By Corollary 5.1 and 5.2, we can solve both resilience problems: There exists a bound k s.t. the finite-marked joint GTS^{bp} is k-step resilient w.r.t. Safe. In fact, the minimal bound is $k_{\min} = 6$. For the computations, see Appendix C.

Concluding: This section provided a generic method for solving the bounded and explicit resilience problem for subclasses of marked GTSs and proper constraints Bad and Safe.

Remark. For marked GTSs without deadlocks, deciding the explicit resilience problem is equivalent to checking the CTL constraint

$$\mathbf{AG}(\text{Bad} \Rightarrow \bigvee_{j=0}^{k} \mathbf{EX}^j \text{Safe}).$$

Moreover, the presented methods may be used to check a class of temporal constraints. E.g., by Proposition 4.1, for post*-effective marked GTS^{bp}, we can check the CTL constraint **EF** Safe where Safe is a negative constraint.

More General Constraints

More general graph constraints as, e.g., in Rensink [Ren04] and [HP09] do not constitue (anti-)ideals w.r.t. the subgraph order, in general.

Example 5.6. Consider the constraint AllLoop = $\forall(\ \circ, \exists(\ \circ\!\!\rightleftharpoons\ ))$ expressing that every node has a loop. The graph consisting of one node and one loop satisfies the latter constraint. However, the "larger" graph consisting of two nodes and one loop does not satisfy it. Thus, the constraint AllLoop does not constitute an ideal. By a similar argument, it does not constitute an anti-ideal.

Moreover, we cannnot mix negative and positive constraints.

Example 5.7. Consider the constraint NodeNoLoop $= \exists(\circ) \wedge \neg\exists(\circlearrowleft)$ expressing that there is a node and there is no loop. The graph consisting of one node satisfies the latter constraint. However, the "larger" graph consisting of one node and one loop does not satisfy it. Thus, the constraint NodeNoLoop does not constitute an ideal. By a similar argument, it does not constitute an anti-ideal.

Regarding the induced subgraph order, considered, e.g., by König & Stückrath [KS17], some nested constraints constitute ideals.

Example 5.8. The constraint $\exists(\ \circ, \neg\exists(\ \circlearrowleft))$ expresses that the graph consisting of a single node is an induced subgraph of the considered graph. More generally, consider the constraint $\exists(C, \bigwedge_{C^+ \in \mathrm{Ext}(C)} \neg\exists(C \hookrightarrow C^+))$ where $\mathrm{Ext}(C)$ is the set of "extended" graphs C^+ obtained from C by adding one edge. The latter constraint expresses that the graph C is an induced subgraph of the considered graph.

Unbounded Path Length

In some cases, resilience problems for marked GTS of unbounded path length can be reduced to resilience problems for marked GTS$^{\mathrm{bp}}$ (of bounded path length) and therefore be decided.

Example 5.9 (constellations in the sky). The rules $\langle 1\textcircled{A} \rightharpoonup 1\textcircled{A}\!\!\rightarrow\!\!\textcircled{A}\rangle$ and $\langle 1\textcircled{A} \rightharpoonup 1\textcircled{A}\!\!\rightarrow\!\!\bullet\rangle$ together with the class of all graphs and the single INITial graph $\textcircled{A}$ form a marked GTS of unbounded path length. We consider both resilience problems for

$$\mathrm{BrightStar} = \exists(\ \bullet \leftarrow \textcircled{A} \rightarrow \bullet\ ), \quad \mathrm{Bad} = \neg\mathrm{BrightStar}, \quad \mathrm{Safe} = \mathrm{BrightStar}.$$

We notice that Bad Safe are "independent" from the paths connecting the star-shaped graphs. Both resilience problems can be solved by considering the following marked GTS$^{\mathrm{bp}}$: the rules $\langle 1\textcircled{A} \rightharpoonup 1\textcircled{A}\quad\textcircled{A}\rangle$ and $\langle 1\textcircled{A} \rightharpoonup 1\textcircled{A}\!\!\rightarrow\!\!\bullet\rangle$ together with the set of disjoint unions of unboundedly many star-shaped graphs and the single INITial graph $\textcircled{A}$. We consider the same constraints for Bad and Safe as before. Since the INITial graph is the basis of $\uparrow\mathrm{post}^*(\mathrm{INIT})$, the marked GTS$^{\mathrm{bp}}$ is post*-effective. By Theorem 5.1 (1) and 5.2 (1), we can solve both resilience problems. There exists a bound k s.t. the (original) marked GTS is k-step resilient. In fact, the minimal bound is $k_{\min} = 8$.

5.3 Example: Circular Process Protocol

We illustrate our decidability results by an example. Consider a ring of three processes P_0, P_1, P_2 each of which has an unordered waiting collection (multiset) containing commands. Each command belongs to a process and is labeled accordingly as c_0, c_1, or c_2. These commands are used for communication between the processes. An informal description of the circular process protocol can be found in Section 4.1. The formalization as GTS is given in Figure 5.5. Each `Clear`-rule is undefined on the node which has a c_i-labeled loop. In a rule application, this node will be deleted and recreated. Each `Leave`-rule identifies two nodes.

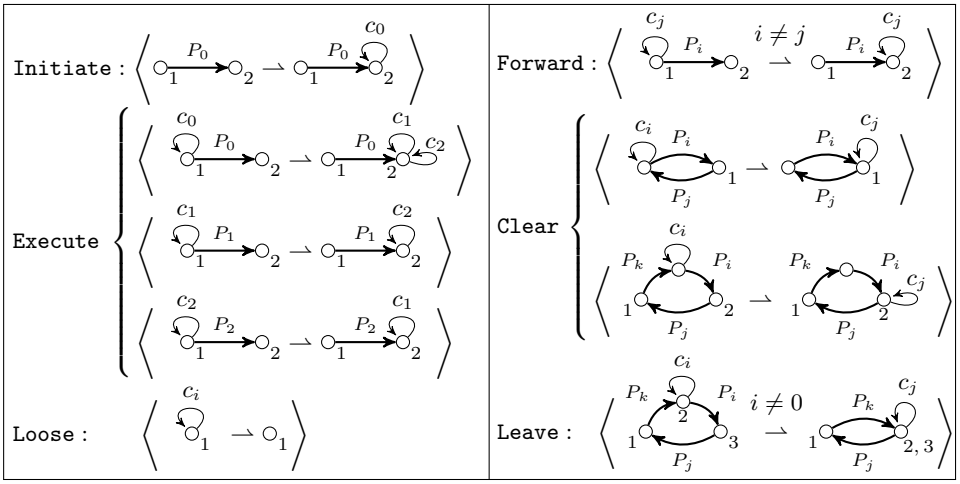


Figure 5.5: Rules of the circular process protocol.

To obtain a marked GTS, we specify the set INIT: It consists of the single graph on the left in Figure 5.6.

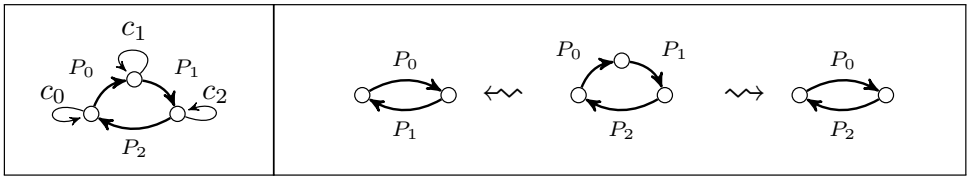
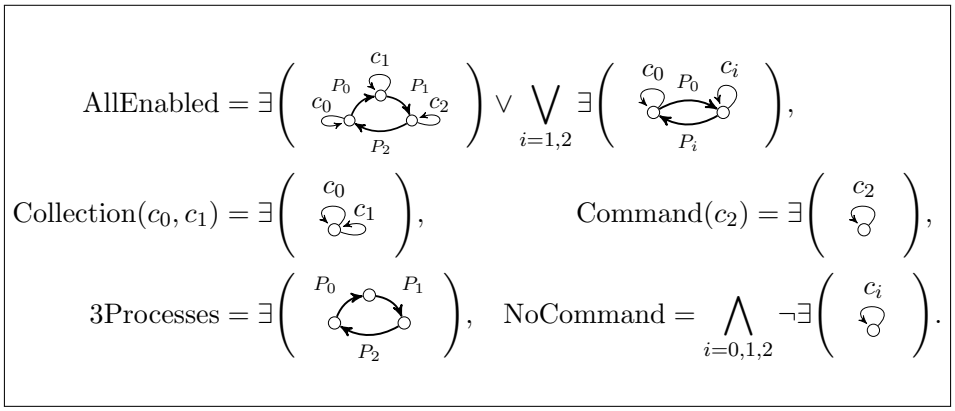


Figure 5.6: Initial graph and basis graphs.

Consider the following proper constraints:



For the graph class S, we consider all graphs with arbitrarily many commands of any kind labeled with c_0, c_1, or c_2 in any collection, fitting to one of the topologies shown in Figure 5.6.

Proposition 5.2. The rules in Figure 5.5 together with the depicted graph class and the initial graph in Figure 5.6 form a finite-marked $\mathrm{GTS}^{\mathrm{bp}}$ satisfying all requirements in Theorem 5.1 and 5.2.

Proof. (1) Finite-marked GTS. The latter graph class is closed under rule application: For all rules except the `Clear`- the `Leave`-rules, the basis graph is preserved. For the `Clear`-rules, a node is deleted, but the basis graph is restored. For the `Leave`-rules, we transit from the underlying basis graph with three nodes to an underlying basis graph with two nodes. For all rules, if the original graph differs from a basis graph only by c_i-labeled loops, also the obtained graph differs from a basis graph only by c_i-labeled loops. Hence, the graph class is closed under rule application. The set INIT consists of a single graph.
(2) Bounded path length. The latter graph class contains only graphs with at most three nodes. Thus, it is of bounded path length.
(3) $\cap$-Basedness. We show that $S = I \cap J$ where

$$I = {\uparrow}\mathcal{B} \text{ with } \mathcal{B} = \Big\{ \quad , \quad , \quad \Big\},$$

$$J = [\![\text{ NPL } \wedge \text{ NCE } \wedge \text{ NPP } \wedge \text{ NOPN } \wedge \text{ N4N }]\!].$$

The anti-ideal J is the semantics of a conjunction of negative constraints[13] where the negative constraints are as follows.

[13] The conjunction of negative constraints is again a negative constraint.

abbreviation (name)	description
NPL (NoProcessLoop)	A process is not a loop.
NCE (NoCommandEdge)	A command can only be a loop.
NPP (NoParallelProcesses)	There are no parallel processes.
NOPN (NoOppositeProcessesNode)	There are no opposite processes with an additional node.
N4N (No4Nodes)	There are no 4 nodes.

$$\text{NPL} = \bigwedge_{i=0,1,2} \neg\exists\Big(\circlearrowright^{P_i} \Big), \qquad \text{NCE} = \bigwedge_{i=0,1,2} \neg\exists\Big(\circ \xrightarrow{c_i} \circ \Big),$$

$$\text{NPP} = \bigwedge_{i,j=0,1,2} \neg\exists\Big(\circ \underset{P_j}{\overset{P_i}{\rightleftarrows}} \circ \Big), \quad \text{NOPN} = \bigwedge_{i,j=0,1,2} \neg\exists\Big(\circ \underset{P_j}{\overset{P_i}{\rightleftarrows}} \circ \ \circ \Big),$$

$$\text{N4N} = \neg\exists\Big(\circ \ \circ \ \circ \ \circ \Big).$$

The set $\mathcal{B}$ is the basis of S since all three basis graphs are included in the anti-ideal J. Indeed, this description of the graph class is equivalent to the previous definition: For every graph G over the same label alphabet, which is "larger" than a basis graph B, $G \in S$ iff G differs from B only by loops representing commands. The graph G cannot contain an additional node by the negative constraints N4N and NOPN. It can also not contain an additional P_i-labeled loop or a c_i-labeled edge which is not a loop by the negative constraints NPL and NCE. It remains the possibility that it contains an additional P_i-labeled edge which is not a loop. However, this is prohibited by the negative constraints NPP and NOPN.

(4) Lossiness & $\bot$-boundedness. The marked GTS$^{\text{bp}}$ is lossy since the rules for losing a command c_i may be applied to any graph containing a command c_i. It is $\bot$-bounded since we can reach a graph with the same topology but containing no commands by (i) initiating a command c_0, (ii) forwarding it to the collection of P_0, (iii) clearing all collections one after another, and (iv) losing the only remaining command. By Proposition 4.2, it is post*-effective. Thus, all cases of Theorem 5.1 and 5.2 apply. □

We can answer the raised questions: Using the algorithms in Section 4.3, we can decide the bounded resilience problem for all instances and compute $k_{\min}$ for the instances 1 and 2, see the tabular below. For the computations, see Appendix C.

instance	1	2	3	4	5
Bad Safe	¬AllEnabled AllEnabled	Command(c_2) Collection(c_0, c_1)	AllEnabled ¬AllEnabled	NoCommand ¬3Processes	¬3Processes 3Processes
bounded (answer)	yes	yes	yes	yes	no
explicit ($k_{\min}$)	6	4	(1)	(5)	–

If Safe is a negative constraint, our method provides only the answer whether there is a bound k. For the third instance, we can infer that $k_{\min} = 1$: By applying a `Clear`-rule to a graph in which all processes are enabled, we can reach a graph in which at least one process is not enabled in one step. Thus, $k_{\min} \leq 1$. Since $\neg$AllEnabled is the negation of AllEnabled, $k_{\min} \neq 0$. For the fourth instance, we use that $\neg$3Processes can be expressed as the positive constraint 2Processes

$$=\exists\left(\begin{array}{c}P_0\\ \circ\rightleftarrows\circ\\ P_1\end{array}\right) \vee \exists\left(\begin{array}{c}P_0\\ \circ\rightleftarrows\circ\\ P_2\end{array}\right).$$

5.4 Example: Logistic System

We give examples of a logistic system each of which is more "typical" for the requirements post*-effectiveness, lossiness, and $\bot$-boundedness.

Consider a logistic system with a unique *logistic center* C from where goods represented by □ are shipped to their *destination* D. The shipment can be direct or via an *interim storage* S. When an order is set, the good enters the system at the logistic center together with the planned route (edges labeled with *rou*) and the information of its destination and current location. This information is represented by edges labeled with *loc* and *dest*, respectively. Its destination D, a possible interim storage S, and routes may already be registered in the system or will enter when the order is set. The `SetOrder`-rules in Figure 5.7 depict the process of setting an order with the planned route and information.

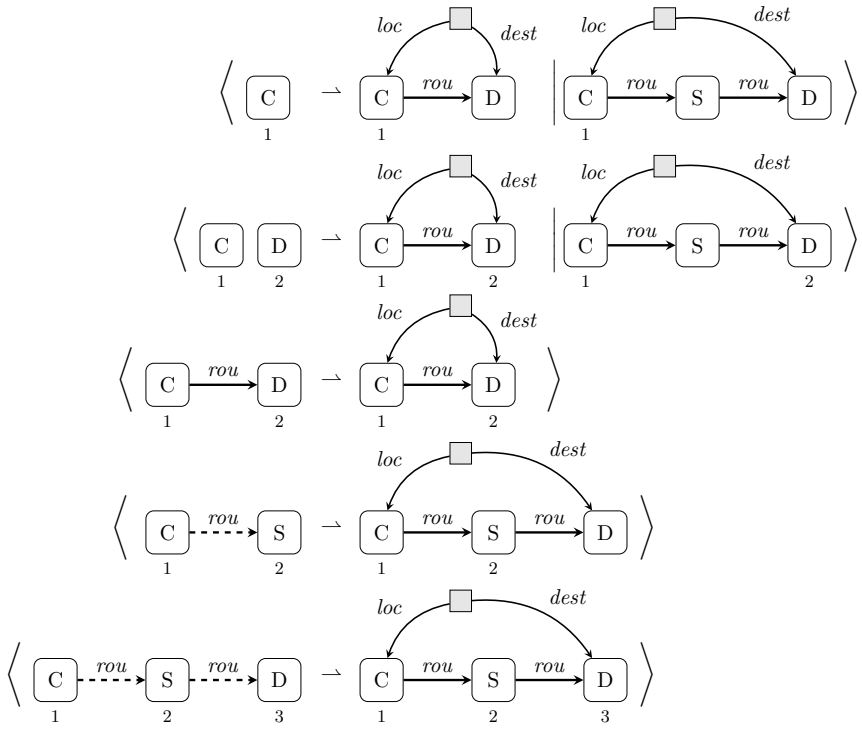


Figure 5.7: The `SetOrder`-rules.

Some rules are presented in the Backus-Naur form, i.e., $\langle L \rightharpoonup R_1 | R_2 \rangle$ depicts the rules $\langle L \rightharpoonup R_1 \rangle$ and $\langle L \rightharpoonup R_2 \rangle$. For every dashed edge, one rule containing the edge in the left-hand side and one rule not containing it is included, e.g., the last scheme depicts four rules. The goods can only be shipped along routes. A good can only be delivered to a destination if it fits to the destination information. The shipment is formalized by the `Ship`-rules shown in Figure 5.8.

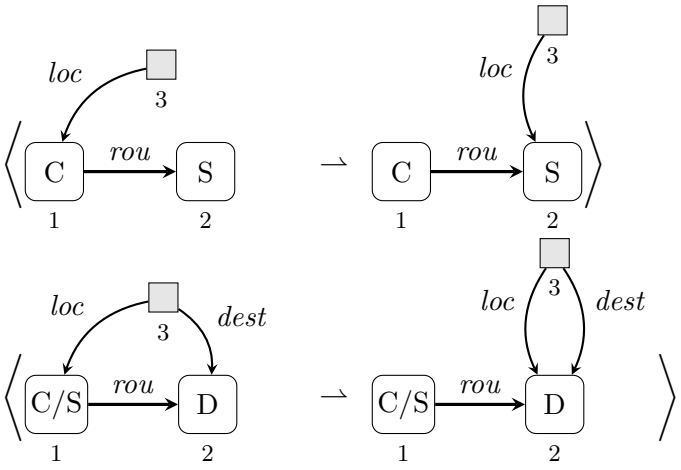


Figure 5.8: The `Ship`-rules.

Some nodes are "C/S"-labeled meaning that both rules are included. This system is erroneous in the sense that a good can be shipped to an interim storage which is not the interim store on the planned route. Another "error" is the loss of a good which is depicted by the `GoodLost`-rule in Figure 5.9.

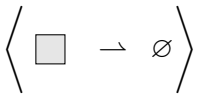


Figure 5.9: The `GoodLost`-rule.

Consider the logistic system $\mathcal{R}_1 = \texttt{SetOrder} \cup \texttt{Ship} \cup \{\texttt{GoodLost}\}$. To obtain a marked GTS, we consider the intersection S of the downward-closure of all graphs reachable from the graph $\boxed{\text{C}}$ and all graphs "larger" than the latter graph. The only basis graph is the graph $\boxed{\text{C}}$. The $\cap$-based graph class S is of path length ≤ 3. By definition, it is closed under application of $\mathcal{R}_1$.

Proposition 5.3 (properties of $\mathcal{R}_1$).

(1) The GTS $\mathcal{R}_1$ together with the $\cap$-based graph class S and any INIT $\subseteq S$ forms a marked GTS$^{\text{bp}}$.

(2) Every finite-marked GTS$^{\text{bp}}$ with rule set $\mathcal{R}_1$ and graph class S is post*-effective.

Proof. Statement (1) follows by definition.
(2) We show $\uparrow \text{post}^*(\text{INIT}) = \uparrow \text{post}^*_{\texttt{GoodLost}}(\text{INIT})$ where $\text{post}^*_{\texttt{GoodLost}}(\text{INIT})$ consists of all graphs reachable from INIT via the `GoodLost`-rule.
"$\supseteq$": It holds $\text{post}^*_{\texttt{GoodLost}}(\text{INIT}) \subseteq \text{post}^*(\text{INIT})$.

"$\subseteq$": Let $G \in \text{INIT}$, $G \Rightarrow_{\mathcal{R}}^{*} H$, and $G' \leq G$ s.t. G' contains no good ($\square$). It holds $G' \in \text{post}^{*}_{\texttt{GoodLost}}(\text{INIT})$ and $H \geq G'$. Thus, $H \in \uparrow\text{post}^{*}_{\texttt{GoodLost}}(\text{INIT})$. Hence, $\text{post}^{*}(\text{INIT}) \subseteq \uparrow\text{post}^{*}_{\texttt{GoodLost}}(\text{INIT})$.
Since INIT is finite, the set $\text{post}^{*}_{\texttt{GoodLost}}(\text{INIT})$ is finite. By Fact 2.6, we can compute a basis of $\uparrow\text{post}^{*}_{\texttt{GoodLost}}(\text{INIT})$. Thus, the finite-marked GTS^{bp} is post*-effective. $\square$

In a more realistic scenario, there may occur "errors" in the logistic system, e.g., a good or information may be lost. The `Error`-rules depict the loss of (i) a good (`GoodLost`), the information about the (ii) location or (iii) destination of a good. Moreover, it can occur that (iv) a route is inaccessible, (v) an interim storage is unusable, or (vi) a destination is not detectable. The `Error`-rules also delete these defects from the system. Formally, the `Error`-rules are defined as in Figure 5.10.

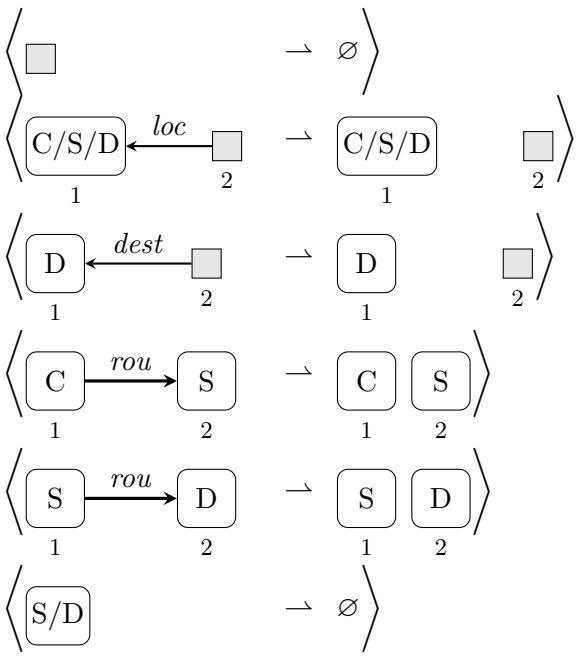


Figure 5.10: The `Error`-rules.

For the logistic system $\mathcal{R}_2 = \texttt{SetOrder} \cup \texttt{Ship} \cup \texttt{Error}$, we obtain the following properties.

Proposition 5.4 (properties of $\mathcal{R}_2$).

(1) The GTS $\mathcal{R}_2$ together with the $\cap$-based graph class S and any $\text{INIT} \subseteq S$ forms a marked GTS^{bp}.

(2) Every finite-marked GTS^{bp} with rule set $\mathcal{R}_2$ and graph class S is lossy.

Proof. Statement (1) follows by definition.
(2) Using the `Error`-rules, we can delete any item except the logistic center C which is the only basis graph. Thus, the marked $\mathrm{GTS}^{\mathrm{bp}}$ is lossy for every set INIT. □

We slightly adapt the model by representing a good as an □-labeled edge pointing from its location to its destination. Thus the information about a good is compressed. We leave out the `Error`-rules except the last ones for deleting an S-labeled (D-labeled) node which we call `Inaccess`. The rules essentially stay the same. However, a `Ship`-rule has to be modified s.t. the destination D is present in the rule. Let $\mathcal{R}_3 = \texttt{SetOrder'} \cup \texttt{Ship'} \cup \texttt{Inaccess'}$ be the modified rule set.

Provided that the number of customers, i.e., destinations is "small", we can assume that number of D-labeled (and S-labeled) nodes is bounded. (This is the case if the logistic company has a small number of major customers.) In a realistic model, this does not apply to the number of goods which may be "very large" over the whole process.

In order to restrict the number of created nodes, we split the `SetOrder'`-rules into `SetOrder1`-rules which do not create nodes and `SetOrder2`-rules which create nodes.[14] For $n \geq 0$, let A_n be the rule-labeled automaton with states $q_0, \ldots, q_n$, transitions from q_i to q_{i+1} labeled with `SetOrder2`, and loops at q_i lableled with $\mathcal{R}_3 \setminus \texttt{SetOrder2}$. In Figure 5.11, the automaton A_n is exemplarily shown for $n = 2$.

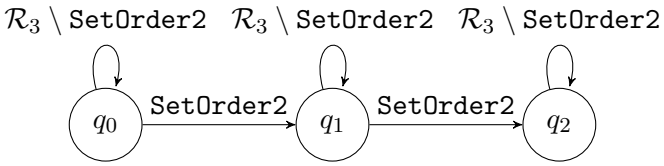


Figure 5.11: The rule-labeled automaton A_n for $n = 2$.

Let $\mathcal{R}_3^n$ denote the synchronization of $\mathcal{R}_3$ with A_n (see Construction 3.1). We have to consider graphs each of which contains exactly one q_i-labeled node for a number $1 \leq i \leq n$. Thus, a basis of this graph class is given by $\mathcal{B}_n = \{\text{C}\ (q_i) : 1 \leq i \leq n\}$. Let S_n be the intersection of $\uparrow\mathcal{B}_n$ with the downward-closure of all graphs reachable from the graph C (q_0). Intuitively, the induced transition system stays essentially the same as before with the restriction that the node-creating `SetOrder`-rules can be applied only boundedly often.

For the logistic system $\mathcal{R}_3^n$, we obtain the following properties.

[14] It is possible to consider only `SetOrder1`-rules. However, this would yield a "static" topology.

Proposition 5.5 (properties of $\mathcal{R}_3^n$).

(1) For every $n \geq 0$, the GTS $\mathcal{R}_3^n$ together with the $\cap$-based graph class S_n and any INIT $\subseteq S_n$ forms a marked GTS$^{\text{bp}}$.

(2) For every $n \geq 0$, the number of nodes in any graph in S_n is bounded.

(3) Every marked GTS$^{\text{bp}}$ with rule set $\mathcal{R}_3^n$ and graph class S_n is $\bot$-bounded.

Proof. Statement (1) follows by definition.
(2) Each `SetOrder2`-rule creates ≤ 2 nodes. Thus, the number of nodes is bounded by $2n + 2$.
(3) The included `Inaccess`-rules ensure that any "smaller" basis graph $\boxed{\text{C}}$ $\textcircled{q_i}$ is reachable in $\leq 2n$ transformation steps. Thus, the marked GTS$^{\text{bp}}$ is $\bot$-bounded (for every set INIT $\subseteq S_n$). □

We summarize which resilience problems are, by Theorem 5.1 and 5.2, decidable for the logistic system.

Fact 5.2.

(1) For every finite-marked GTS$^{\text{bp}}$ with rule set $\mathcal{R}_1$ and graph class S, both resilience problems are decidable in the case where Bad is negative and Safe is positive.

(2) For every finite-marked GTS$^{\text{bp}}$ with rule set $\mathcal{R}_2$ and graph class S, both resilience problems are decidable in the cases where Safe is positive.

(3) For every finite-marked GTS$^{\text{bp}}$ with rule set $\mathcal{R}_3^n$ and graph class S_n, the bounded resilience problem is decidable in the cases where Bad is negative and the explicit resilience problem is decidable in the case where Bad is negative and Safe is positive.

5.5 Rule-specific Criteria

In this section, we investigate sufficient criteria of GTSs for the decidability results in Chapter 5 which are: (1) post*-effectiveness, (2) lossiness, and (3) $\bot$-boundedness. For each requirement, we search for easily checkable, rule-specific conditions.

Post*-effectiveness

For the requirement of post*-effectiveness, we consider properties of the rules.

Definition 5.4 (preserving & node-bijective). A rule is

(1) *preserving* if it is total and injective,

(2) *node-bijective* if it is bijective on the nodes.

These rule properties yield sufficient criteria for post*-effectiveness.

Theorem 5.3 (criteria for post*-effectiveness). A finite-marked GTS^{bp} is post*-effective if all rules are preserving or all rules are node-bijective.

Proof. Let $\langle S, \mathcal{R}, \text{INIT}\rangle$ be a finite-marked GTS^{bp}. If every rule in $\mathcal{R}$ is preserving, the statement follows by Fact 2.6 since $\uparrow\text{post}^*(\text{INIT}) = \uparrow\text{INIT}$. If every rule in $\mathcal{R}$ is node-bijective, the statement follows by the reduction in the proof of [BDK+12, Prop. 10]. For any graph G, a Petri net with initial marking is constructed s.t. reachability and the order of markings correspond to reachability via $\Rightarrow^*_{\mathcal{R}}$ from G and the subgraph order, respectively. In our case[15], the places are given by $V_G \times V_G \times \Lambda_E$. The number of tokens in a place $\langle v, v', \lambda\rangle$ is given by

$$|\{e \in E : \text{lab}^E(e) = \lambda, \text{src}(e) = v, \text{tgt}(e) = v'\}|.$$

Each rule may give rise to a multiple number of transitions since a rule may have multiple matches. On the balance sheet, a transition takes tokens out of a place $\langle v, v', \lambda\rangle$ if the corresponding rule r deletes λ-labeled edges between v (source) and v' (target), and adds tokens in a place if r creates λ-labeled edges between v and v'. See [BDK+12] for further details.
Petri nets are post*-effective, see Example 4.2. Thus, the basis of $\uparrow\text{post}^*(G)$ can be computed by computing the corresponding basis in the constructed Petri net. □

Using that joint GTSs are GTSs, we obtain sufficient criteria for post*-effectiveness of marked joint GTS.

Proposition 5.6. A finite-marked *annotated joint* GTS^{bp} of $\mathcal{S}$ and $\mathcal{E}$ is post*-effective if all rules in $\mathcal{S} \cup \mathcal{E}$ are preserving or all rules in $\mathcal{S} \cup \mathcal{E}$ are node-bijective.

Proof sketch. The statement for preserving rules follows by Lemma 2.3. The statement for node-bijective rules follows by Theorem 5.3. We use the fact that joint GTSs are GTSs. For the full proof, see Appendix B. □

These conditions are favorable in the sense of decidability.

Fact 5.3 (decidable conditions). Given a GTS, it is decidable whether all rules are preserving or all rules are node-bijective.

[15] In [BDK+12], hypergraphs are considered.

Lossiness

To obtain a sufficient criterion for lossiness, we consider special rules.

Decision. We construct lossy rules suited to $\cap$-based graph classes.

Lossiness in the context of $\cap$-based graph classes means the following: if a graph G is reachable from INIT, all subgraphs of G, each of which is "larger" than a basis graph, are also reachable from INIT. In Figure 5.12, the notion of lossiness is illustrated. For every graph G' which is "smaller" than a graph G reachable from a graph $G_1 \in \text{INIT}$ and "larger" than a basis element $B \in \mathcal{B}$, there exists a graph $G_2 \in \text{INIT}$ from which G' is reachable.

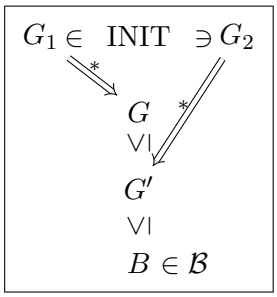


Figure 5.12: Illustration of lossiness.

As shown in Figure 5.13, we assume that we continue the sequence of transformations from G to reach its subgraphs, i.e., every graph G', which is "smaller" than a graph G reachable from a graph $G_1 \in \text{INIT}$ and "larger" than a basis element $B \in \mathcal{B}$, is reachable from G via "lossy rules".

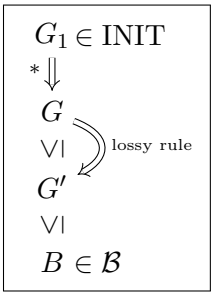


Figure 5.13: Basic idea of lossy rules.

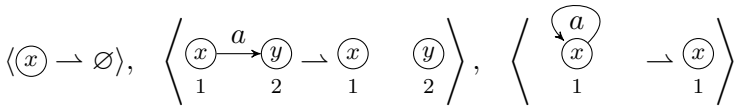
Therefore we have to be able to delete every item in G which is not in (the image of) any basis graph. This can be done by "lossy rules" deleting such items, i.e., every node and every edge outside of a basis graph.[16]

We aim to reach "smaller" graphs but not "smaller" than basis graphs.

[16] In [KS17], "lossy rules" w.r.t. the minor order, i.e., edge contraction rules, are considered in order to obtain well-structuredness for GTSs.

Construction 5.1 (lossy rules). Let $\langle S, \mathcal{R}, \text{INIT}\rangle$ be marked GTS where S is a graph class over Λ and $\mathcal{B}$ the basis of S. The set of *lossy rules* $\mathcal{R}_{\text{loss}}(\text{INIT})$ w.r.t. a set $\text{INIT} \subseteq S$ is constructed as follows.

(1) The set of *deletion rules* $\mathcal{R}_\Lambda$ consists of all rules for deleting a node, an edge, and a loop, i.e.,



where $x, y \in \Lambda_V$, $a \in \Lambda_E$.

(2) Given a deletion rule $\langle L \rightharpoonup R\rangle$, a basis graph $B \in \mathcal{B}$, and a graph A with $A \hookrightarrow L$ (no isomorphism) and $A \hookrightarrow B$, construct the rules $\langle L_B \rightharpoonup R_B\rangle$ where the pushout objects L_B, R_B, and the morphism $L_B \rightharpoonup R_B$ are obtained by the following pushouts:

$$\begin{array}{ccccc} A & \overset{\not\cong}{\hookrightarrow} & L & \longrightarrow & R \\ \downarrow & \text{(PO)} & \downarrow & \text{(PO)} & \downarrow \\ B & \hookrightarrow & L_B & \longrightarrow & R_B \end{array}$$

$\langle L_B \rightharpoonup R_B\rangle$ is a lossy rule w.r.t. INIT if $L_B \in{\downarrow}\text{post}^*(\text{INIT})$.

$\mathcal{R}_{\text{loss}}(S)$ is constructed analogously replacing INIT by S.

Example 5.10. The lossy rules of the circular process protocol in Section 5.3 are shown in Figure 5.14.

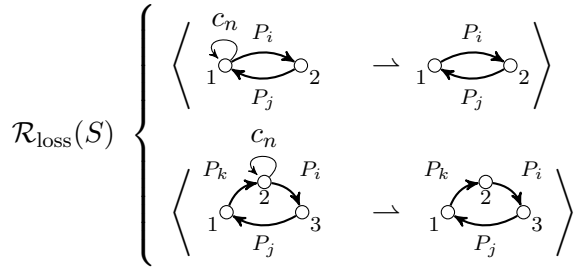
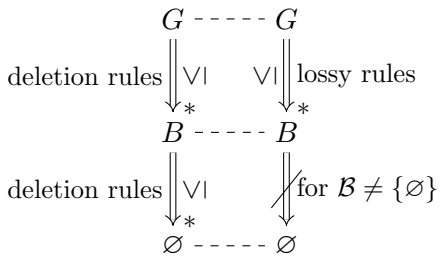


Figure 5.14: The lossy rules of the circular process protocol.

Intuitively speaking, lossy rules delete items outside of a basis graph. The difference between deletion rules and lossy rules is depicted below.



Via deletion rules from any graph the empty graph is reachable. Via lossy rules from any graph any "smaller" basis graph is reachable. However, via lossy rules we cannot reach a graph "smaller" than a basis graph.

Proposition 5.7.

(a) $\mathcal{R}_{\text{loss}}(\text{INIT})$ is effectively constructible from a finite-marked GTS^{bp} with $\cap$-based graph class.

(b) $\mathcal{R}_{\text{loss}}(S)$ is effectively constructible from a $\cap$-based graph class S.

Proof. (a) First, we construct the deletion rules which are finitely many. Second, we construct all the pushouts in (2). We construct only finitely many pushouts since the number of basis graphs B, deletion rules $\langle L \rightharpoonup R \rangle$, and subgraphs (up to isomorphism) of L and B is finite. For a constructed rule $\langle L_B \rightharpoonup R_B \rangle$, we check whether $L_B \in{\downarrow} \text{post}^*(\text{INIT})$ in the following way. By Lemma 5.4 and Proposition 5.1, every finite-marked GTS^{bp} with $\cap$-based graph class induces a finite-marked SWSTS. We can decide $L_B \in \downarrow \text{post}^*(\text{INIT})$ by checking $G \in \text{pre}^*(\uparrow\{L_B\})$ for every $G \in \text{INIT}$ (Lemma 2.3). (b) For the lossy rules $\mathcal{R}_{\text{loss}}(S)$, the same construction is feasible. Since $\text{post}^*(S) = S$, we check $L_B \in S$ for a constructed rule $\langle L_B \rightharpoonup R_B \rangle$. □

For lossiness, it is sufficient that the lossy rules are contained in the GTS. More generally, we regard an "appropriate simulation" of lossy rules.

Definition 5.5 (track morphism & simulation). For a transformation $G \Rightarrow H$, the *track morphism* is the obtained morphism $G \rightharpoonup H$. For a transformation sequence[17] $G \Rightarrow^* H$, the track morphism is the composition of the single track morphisms. A GTS $\mathcal{R}$ is *simulated* by a GTS $\mathcal{R}'$, written as $\mathcal{R} \sqsubseteq \mathcal{R}'$, if for every rule $\langle p : L \rightharpoonup R \rangle \in \mathcal{R}$, there exists a sequence $L \Rightarrow^*_{\mathcal{R}'} R$ s.t. its track morphism $\widetilde{p} : L \rightharpoonup R$ is equal to p.

Example 5.11. The lossy rules of the circular process protocol in Figure 5.14 are simulated by the `Loose`-rules (Section 5.3) in one step.

Lemma 5.5 (simulation). If $\mathcal{R} \sqsubseteq \mathcal{R}'$, then $\Rightarrow_{\mathcal{R}} \subseteq \Rightarrow^*_{\mathcal{R}'}$.

[17] From now on, we may shortly speak of a transformation when considering transformation sequences.

Proof. Let $G \Rightarrow_r H$ via $r = \langle p : L \rightharpoonup R \rangle \in \mathcal{R}$ according to the match $m : L \hookrightarrow G$. By definition of a simulation, there exists a transformation $L \Rightarrow^*_{\mathcal{R}'} R$ s.t. its track morphism $\widetilde{p} : L \rightharpoonup R$ is equal to p. By Lemma A.2, the composition of pushouts is a pushout. Hence, the transformation $L \Rightarrow^*_{\mathcal{R}'} R$ via rules $r'_1 = \langle L_1 \rightharpoonup R_1 \rangle$, ..., $r'_\ell = \langle L_\ell \rightharpoonup R_\ell \rangle$ with track morphism $\widetilde{p} = \widetilde{p}_\ell \circ \ldots \circ \widetilde{p}_1$ induces a transformation $G \Rightarrow^*_{\mathcal{R}'} \widetilde{H}$, as illustrated below for $\ell = 2$.

$$\begin{array}{ccccc}
L_1 & \rightarrowtail & R_1 & & L_2 \rightarrowtail R_2 \\
& \searrow & \text{(PO)} \; \downarrow & \swarrow & \text{(PO)} \; \downarrow \\
& & L \xrightarrow{\widetilde{p}_1} R' & \xrightarrow{\widetilde{p}_2} & R \\
& & m \downarrow \; \text{(PO)} & \downarrow \; \text{(PO)} & \downarrow \\
& & G & \rightarrowtail H' & \rightarrowtail \widetilde{H}
\end{array}$$

By the composition of the lower pushouts, we obtain the following pushout:

$$\begin{array}{ccc}
L & \xrightarrow{p = \tilde{p}} & R \\
m \downarrow & \text{(PO)} & \downarrow \\
G & \rightarrowtail & \widetilde{H}
\end{array}$$

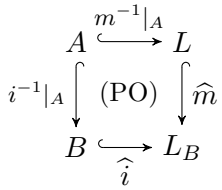
Since pushout objects are unique up to isomorphism (Lemma A.1), $\widetilde{H} \cong H$. Thus, we obtain $G \Rightarrow^*_{\mathcal{R}'} H$. □

Theorem 5.4 (criterion for lossiness). A marked $\mathrm{GTS}^{\mathrm{bp}}$ $\langle S, \mathcal{R}, \mathrm{INIT} \rangle$ with $\cap$-based graph class is lossy if $\mathcal{R}_{\mathrm{loss}}(\mathrm{INIT}) \sqsubseteq \mathcal{R}$ or $\mathcal{R}_{\mathrm{loss}}(S) \sqsubseteq \mathcal{R}$.

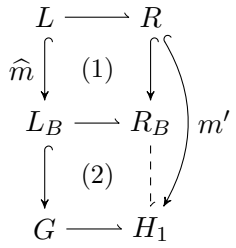
Proof. First, we show that containedness of lossy rules implies lossiness. Let $G \in \mathrm{post}^*(\mathrm{INIT})$, $H \leq G$ and $B \leq H$ for a basis graph B. There exists a sequence $G \Rightarrow^* H$ via deletion rules. Consider the first transformation $G \Rightarrow H_1$ in the latter sequence. For the deletion rule $\langle L \rightharpoonup R \rangle$, we obtain the following pushout.

$$\begin{array}{ccc}
L & \rightarrowtail & R \\
m \downarrow & \text{(PO)} & \downarrow m' \\
G & \rightarrowtail & H_1 \hookleftarrow B
\end{array}$$

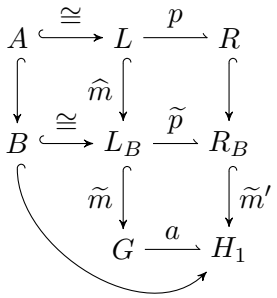
The morphism $G \rightharpoonup H_1$ is injective and surjective since $L \rightharpoonup R$ is injective and surjective (Lemma A.3). By inverting, we obtain a total, injective morphism $H_1 \hookrightarrow G$ and hence, $i : B \hookrightarrow G$. For $A = m(L) \cap i(B)$ and $L_B = m(L) \cup i(B)$, the diagram



is a pushout where $\widehat{i} : B \hookrightarrow i(B) \hookrightarrow L_B$ and $\widehat{m} : L \hookrightarrow m(L) \hookrightarrow L_B$. We split the first pushout as follows.



R_B is the pushout object of (1). The outer rectangle is a pushout and thus, commutes. By definition of a pushout, there exists a morphism $R_B \rightharpoonup H_1$ s.t. the morphisms on the right side and (2) commute. Since (1) and (1)+(2) are pushouts, also (2) is a pushout (Lemma A.2). Since $L_B \hookrightarrow G$ (the identity) is total and injective, also $R_B \hookrightarrow H_1$ (Lemma A.3). Assume that $A \hookrightarrow L$ is an isomorphism, i.e., surjective. Then, $L_B \cong B$. We obtain the following diagram.



Since $p : L \rightharpoonup R$ is injective and surjective, also $\widetilde{p} : L_B \rightharpoonup R_B$ is injective and surjective (Lemma A.3). Thus, the domain of $\widetilde{p}$ is isomorphic to R_B. Hence, also the domain of $\widetilde{m}' \circ \widetilde{p}$ is isomorphic to R_B. The domain of $a \circ \widetilde{m}$ is isomorphic to B and equal to the domain of $\widetilde{m}' \circ \widetilde{p}$. Thus, $B \cong L_B \cong R_B$. However, the upper right square depicts the transformation $L_B \Rightarrow R_B$ via the deletion rule $\langle L \rightharpoonup R \rangle$. Thus, L_B cannot be isomorphic to R_B (either the number of nodes or edges of R_B is strictly smaller than the number of nodes or edges of L_B). Since $L_B \leq G$, $L_B \in{\downarrow}\, \mathrm{post}^*(\mathrm{INIT})$. Hence, $\langle L_B \rightharpoonup R_B \rangle \in \mathcal{R}_{\mathrm{loss}}(\mathrm{INIT})$. We showed $G \Rightarrow_{\mathcal{R}_{\mathrm{loss}}(\mathrm{INIT})} H_1$. It follows that $G \Rightarrow^*_{\mathcal{R}_{\mathrm{loss}}(\mathrm{INIT})} H$.

We generalize this argument to the case $\mathcal{R}_{\text{loss}}(\text{INIT}) \sqsubseteq \mathcal{R}$. Consider a transformation $G \Rightarrow^*_{\mathcal{R}_{\text{loss}}(\text{INIT})} H$. By Lemma 5.5, $G \Rightarrow^*_{\mathcal{R}} H$. Thus, the marked GTS$^{\text{bp}}$ is lossy. By construction, $\mathcal{R}_{\text{loss}}(\text{INIT}) \subseteq \mathcal{R}_{\text{loss}}(S)$. Hence, $\mathcal{R}_{\text{loss}}(S) \sqsubseteq \mathcal{R}$ implies $\mathcal{R}_{\text{loss}}(\text{INIT}) \sqsubseteq \mathcal{R}$. □

⊥-Boundedness

Similar to lossy rules, we consider special rules to obtain a sufficient criterion for ⊥-boundedness.

Decision. We construct bottom rules suited to ∩-based graph classes.

The notion of ⊥-boundedness means that for every graph $G \in S$, we can reach every "smaller" basis graph $B \leq G$ in a bounded number of transformation steps, as depicted below.

$$\leq \ell \text{ transformations} \left(\begin{array}{c} G \in S \\ \vee \mid \\ B \in \mathcal{B} \end{array} \right.$$

First we consider necessary condition for ⊥-boundedness. A graph class is *node-bounded* if the number of nodes in any graph of the class is bounded. To achieve ⊥-boundedness, it is necessary that the graph class is node-bounded.

Proposition 5.8 (node-bounded). For ⊥-bounded marked GTS$^{\text{bp}}$s, the graph class is node-bounded.

Proof. By assumption, the marked GTS$^{\text{bp}}$ is ⊥-bounded. Let ℓ be the bound for reaching any basis graph. Since the set of rules is finite and every rule may delete only boundedly many nodes, the number of nodes deleted in a transformation of length $\leq \ell$ is bounded. Let n be the latter bound. For every graph $G \in S$, there exists a basis graph $B \leq G$. Thus, by definition of ⊥-boundedness, we can reach B in $\leq \ell$ transformation steps. The number of nodes in any graph of the basis is bounded since, by Fact 2.6, every basis is finite. We obtain that

$$|V_G| = |V_B| + \underbrace{|V_G| - |V_B|}_{\leq n} \leq |V_B| + n \leq \max_{B \in \mathcal{B}} |V_B| + n$$

where $\mathcal{B}$ is the basis of S. □

The idea is to delete the nodes outside of basis graphs. In any case these nodes have to be deleted by "outer" bottom rules. Thus, the remaining graph only consists of a basis graph with additional edges. For these "inner" edges, edge deletion rules may be useful. However, there may be unboundedly many additional edges. To delete these edges in a bounded number of

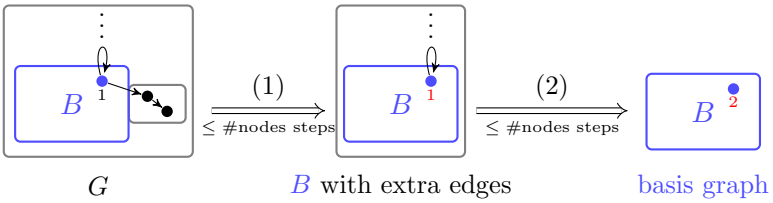


Figure 5.15: From a graph G to a basis graph B via bottom rules.

transformation steps, we need to be able to delete and recreate each node of unbounded "inner" node degree in a basis graph.

In Figure 5.15, it is exemplarily illustrated how to reach a basis graph B via bottom rules. First the two nodes ouside the basis graph B are deleted. Then one node of the basis graph is deleted and recreated.

Definition 5.6 (inner node degree). A node v of a graph B in a graph class S is of *bounded inner node degree* if for any morphism $i : B \hookrightarrow G \in S$, the number of edges $e \in E_G$, which are incident to $i(v)$ and a node in $i(V_B)$, is bounded. We denote this bound by $d(v)$ and the set of nodes of unbounded inner node degree by U_B.

For the other nodes of a basis graph, we can choose whether we delete the remaining edges by edge deletion rules or by deleting and recreating the respective node.

A rule $\langle L \rightharpoonup R \rangle$ can be written in a more explicit form as $\langle L \hookleftarrow K \to R \rangle$ where K is the domain of the partial morphism $L \rightharpoonup R$. We use this form in the construction of the bottom rules.

Construction 5.2 (bottom rules). Let S be a graph class over Λ and B the basis of S. The *bottom rules* w.r.t. S are constructed as follows. For a basis graph $B \in \mathcal{B}$,

(1) $\langle B + \textcircled{x} \hookleftarrow B \hookrightarrow B \rangle$ is called an *outer bottom rule* if $B + \textcircled{x} \in S$,

(2) $\langle B \hookleftarrow B \setminus \{v\} \hookrightarrow B \rangle$ is called an *inner bottom rule of type 1* for $v \in U_B$,

(3) for $v \in V_B \setminus U_B$, we include (either)

   (a) the inner bottom rule of type 1 as constructed in (2) or

   (b) all *inner bottom rules of type 2*: $\langle B^+ \hookleftarrow B \hookrightarrow B \rangle$ where $B^+ \in S$ is obtained from B by adding one edge incident to v and a node $v' \in V_B \setminus U_B$ (including $v' = v$).

For every basis graph $B \in \mathcal{B}$, the choice for the type of the inner bottom rules is represented by a function $f_B : V_B \setminus U_B \to \{1, 2\}$, i.e., $f_B(v) = i$

means that we choose the inner bottom rule of type i for the node v of B. The set of all functions f_B is denoted by f. If we choose only inner bottom rules of type 1, we write $f \equiv 1$. We denote the bottom rules w.r.t. S and f by $\mathcal{R}_\perp(S, f)$.

The outer bottom rules allow to delete nodes outside (the image of) a basis graph. The inner bottom rues of type 1 allow to delete incident edges of inner nodes (of bounded or unbounded inner node degree). The inner bottom rules of type 2 allow to delete incident edges of inner nodes of bounded inner node degree.

Example 5.12. The bottom rules of the logistic system $\mathcal{R}_3^n$ with a bounded number of nodes consist only of the rules for deleting a node $\boxed{\text{S/D}}$. These are the outer bottom rules. Since no node of a basis graph $\boxed{\text{C}}$ (q_i) has an unbounded inner node degree, we can choose the inner bottom rules of type 2 for $\boxed{\text{C}}$ and for (q_i). However, since there are no edges or loops attached to any of both nodes in any graph of the graph class, there exist no inner bottom rules of type 2.

Proposition 5.9. If $f \equiv 1$, the bottom rules $\mathcal{R}_\perp(S, f)$ are effectively constructible from a $\cap$-based graph class S.

Proof. First, we construct all outer bottom rules which are finitely many. There is no node v of a basis graph B with $f_B(v) = 2$. Thus, for every node of a basis graph, we construct the inner bottom rule of type 1. □

For $\perp$-boundedness, it is sufficient that the bottom rules are contained in the GTS. More generally, we regard a "weak simulation" of bottom rules.

Definition 5.7 (weak simulation). A GTS $\mathcal{R}$ is *weakly simulated* by a GTS $\mathcal{R}'$, written as $\mathcal{R} \boxdot \mathcal{R}'$, if for every rule $\langle p : L \rightharpoonup R \rangle \in \mathcal{R}$, there exists a transformation $L \Rightarrow_{\mathcal{R}'}^* R$ s.t. its track morphism $\tilde{p} : L \rightharpoonup R$ is equal to p on the domain of $\tilde{p}$ (this implies that the domain of $\tilde{p}$ is a subset of the domain of p).

A weak simulation generalizes a simulation, i.e., nodes can be deleted and recreated. E.g., an inner bottom rule $\langle B \hookleftarrow B \setminus \{v\} \hookrightarrow B \rangle$ can be weakly simulated by the rule $\langle B \hookleftarrow \varnothing \hookrightarrow B \rangle$.

Example 5.13. The bottom rules of the logistic system in Example 5.12 are (weakly) simulated by the `Inaccess`-rules (Section 5.4) in one step.

Lemma 5.6 (weak simulation). For $\mathcal{R} \boxdot \mathcal{R}'$, there exists a bound $\ell \geq 0$ s.t. for all graphs G, H: $G \Rightarrow_{\mathcal{R}} H$ implies that there exists $\widetilde{H} \leq H$ with $G \Rightarrow_{\mathcal{R}'}^{\leq \ell} \widetilde{H}$.[18]

[18] The notion of a *weak simulation* is justified: One can derive from the proof of Lemma 5.6 that there exists a total, injective, and node-bijective morphism from $\widetilde{H}$ to H.

Proof. Let $G \Rightarrow_r H$ according to the match $m : L \hookrightarrow G$ and $r = \langle p : L \rightharpoonup R \rangle \in \mathcal{R}$. By definition of a weak simulation, there exists a transformation $L \Rightarrow^{\ell}_{\mathcal{R}'} R$ s.t. its track morphism $\widetilde{p} : L \rightharpoonup R$ is equal to p on the domain of $\widetilde{p}$. Similar to the proof of Lemma 5.5, we obtain a transformation $G \Rightarrow^{\ell}_{\mathcal{R}'} \widetilde{H}$. Consider the pushouts

$$\begin{array}{ccc} L & \xrightarrow{p} & R \\ {\scriptstyle m}\downarrow & (\mathrm{PO}) & \downarrow \\ G & \longrightarrow & H \end{array} \qquad\qquad \begin{array}{ccc} L & \xrightarrow{\widetilde{p}} & R \\ {\scriptstyle m}\downarrow & (\mathrm{PO}) & \downarrow \\ G & \longrightarrow & \widetilde{H} \end{array}$$

whereby we assume that H and $\widetilde{H}$ are given as in the canonical construction (Construction A.2). The equivalence classes of items in H and $\widetilde{H}$ are denoted by $[\cdot]_p$ and $[\cdot]_{\widetilde{p}}$, respectively. We define

$$\begin{aligned} h : \widetilde{H} &\to H, \\ [x]_{\widetilde{p}} &\mapsto [x]_p \end{aligned}$$

and check the well-definedness, morphism property, and injectivity.

Well-definedness. (i) For a valid equivalence class $[x]_{\widetilde{p}}$, also $[x]_p$ is valid: Assume that $[x]_p$ is not valid, i.e., there exists $m(\hat{x}) \in [x]_p$ s.t. $p(\hat{x})$ is undefined. It follows that $[m(\hat{x})]_p = [x]_p = \{x\}$ is a singleton. Thus, $x = m(\hat{x})$ and $m(\hat{x}) \in [x]_{\widetilde{p}}$. For every item $\hat{x}$ of L, $\widetilde{p}(\hat{x})$ is undefined if $p(\hat{x})$ is undefined. Thus, $[x]_{\widetilde{p}}$ is not valid, a contradiction. Hence, $[v]_{\widetilde{p}} \in V_{\widetilde{H}}$ implies $[v]_p \in V_H$. It follows directly that $[e]_{\widetilde{p}} \in E_{\widetilde{H}}$ implies $[e]_p \in E_H$. (ii) Let $[x]_{\widetilde{p}} = [y]_{\widetilde{p}}$. If $[x]_{\widetilde{p}} = \{x\}$ is a singleton, then $x = y$, and thus, $[x]_p = [y]_p$. Consider the remaining case $[x]_{\widetilde{p}} = \{\widetilde{p}(\hat{x}), m(\hat{x})\}$. Since $\widetilde{p}(\hat{x}) = p(\hat{x})$, also $[x]_{\widetilde{p}} = [x]_p$. Analogously, $[y]_{\widetilde{p}} = [y]_p$. Thus, $[x]_p = [x]_{\widetilde{p}} = [y]_{\widetilde{p}} = [y]_p$.

Morphism. We show that h preserves the source function. For every equivalence class $[e]_{\widetilde{p}} \in E_{\widetilde{H}}$ of edges, $h(\mathrm{src}_{\widetilde{H}}([e]_{\widetilde{p}})) = h([\mathrm{src}_{G+R}(e)]_{\widetilde{p}}) = [\mathrm{src}_{G+R}(e)]_p = \mathrm{src}_H([e]_p) = \mathrm{src}_H(h([e]_{\widetilde{p}}))$. The statement for the target function follows analogously. We show that labels are preserved. For every equivalence class $[x]_{\widetilde{p}}$, $\mathrm{lab}_H(h([x]_{\widetilde{p}})) = \mathrm{lab}_H([x]_p) = \mathrm{lab}_{G+R}(x) = \mathrm{lab}_{\widetilde{H}}([x]_{\widetilde{p}})$.

Injectivity. Let $h([x]_{\widetilde{p}}) = h([y]_{\widetilde{p}})$, i.e., $[x]_p = [y]_p$ where $[x]_{\widetilde{p}}, [y]_{\widetilde{p}} \in V_{\widetilde{H}} \cup E_{\widetilde{H}}$. If $[x]_p = \{x\}$ is singleton, then $x = y$, and thus, $[x]_{\widetilde{p}} = [y]_{\widetilde{p}}$. Consider the remaining case $[x]_p = \{p(\hat{x}), m(\hat{x})\} = [y]_p$. W.l.o.g., assume $x = p(\hat{x})$ and $y = m(\hat{x})$. Assume that $\widetilde{p}(\hat{x})$ is undefined. Then, $m(\hat{x}) \in [m(\hat{x})]_{\widetilde{p}} = [y]_{\widetilde{p}}$. Hence, $[y]_{\widetilde{p}}$ is not valid, i.e., not in $V_{\widetilde{H}} \cup E_{\widetilde{H}}$, a contradiction. Thus, $\widetilde{p}(\hat{x})$ is defined and $p(\hat{x}) = \widetilde{p}(\hat{x})$. It follows that $[x]_{\widetilde{p}} = [y]_{\widetilde{p}}$. Since pushout objects are unique up to isomorphism (Lemma A.1), it follows that $\widetilde{H} \leq H$ for general pushout objects H and $\widetilde{H}$.

We have shown that for every rule r, there exists a bound $\ell \geq 0$ s.t. $\forall G, H : G \Rightarrow_r H$ implies $\exists \widetilde{H} \leq H : G \Rightarrow^{\leq \ell}_{\mathcal{R}'} \widetilde{H}$. Since the GTS $\mathcal{R}$ is by definition finite, there exists also a bound on the whole GTS. □

Theorem 5.5 (criterion for $\bot$-boundedness). A marked GTS$^{\text{bp}}$ with $\cap$-based graph class S is $\bot$-bounded if S is node-bounded and $\mathcal{R}_\bot(S,f) \boxdot \mathcal{R}$ where f is an arbitrary choice function.

Proof. First, we show that node-boundedness and containedness of bottom rules implies $\bot$-boundedness. Let $G \in S$ be graph and $B \leq G$ a basis graph. We apply the outer bottom rules to get rid of the nodes outside of (the image of) the basis graph B. The remaining graph consists of the graph B with additional edges. The additional edges of a node $v \in V_B$ can be deleted by inner bottom rules of type 1 or type 2: For $v \in U_B$, we use the inner bottom rule of type 1. For $v \in V_B \setminus U_B$ and $f_B(v) = 1$, we use the inner bottom rule of type 1. Each of these rules deletes and recreates v. Hence, in the (image of the) obtained graph, v has no additional edges. For $v \in V_B \setminus U_B$ and $f_B(v) = 2$, we use all inner bottom rules of type 2. Each of these rules deletes a single additional edge. We show that this is feasible in a bounded number of transformation steps. Let n the maximal number of nodes and d be the maximal inner node degree, i.e., $d = \max_{v \in V_B \setminus U_B, B \in \mathcal{B}} d(v)$ where $\mathcal{B}$ is the basis of S. We obtain that the number of transformation steps is

$$\leq \overbrace{\underbrace{(|V_G| - |V_B|)}_{\leq n}}^{\text{outer } \bot \text{ rules}} + \overbrace{\underbrace{|U_B|}_{\leq n}}^{\text{type 1}} + \overbrace{\underbrace{|f_B^{-1}(\{1\})|}_{\leq |V_B \setminus U_B|}}^{\text{type 1}} + \overbrace{\underbrace{|f_B^{-1}(\{2\})|}_{\leq |V_B \setminus U_B|} \cdot d}^{\text{type 2}}$$

$$\leq 2n + |V_B \setminus U_B| + |V_B \setminus U_B| \cdot d$$

$$\leq 3n + nd =: n'.$$

We generalize this argument to the case $\mathcal{R}_\bot(S,f) \boxdot \mathcal{R}$. Consider a transformation $G \Rightarrow_{\mathcal{R}_\bot}^{\leq n'} B$ where $\mathcal{R}_\bot := \mathcal{R}_\bot(S,f)$. By Lemma 5.6, there exists a bound $\ell \geq 0$ s.t. $G \Rightarrow_{\mathcal{R}}^{\leq n'\ell} H$ for a graph $H \leq B$. By closedness of S under rule application, $H \in S$. There exists a basis graph $B' \leq H$. Hence, $B' \leq B$ which implies that $B' \cong H \cong B$. We obtain that we can reach any "smaller" basis graph in $\leq n'\ell = n(3+d)\ell$ steps. This bound only depends on the graph class S and the GTS $\mathcal{R}$. □

Concluding: This section provided sufficient, rule-specific criteria for the requirements post*-effective, lossy, and $\bot$-bounded.

Complexity

As stated in Chapter 4, the complexity of the alogrithms highly depends on the concrete graph transformation system and the input. Proper forward procedures might be of considerable complexity. We encounter proper forward procedures when using the post*-effectiveness of finite-marked GTS$^{\text{bp}}$ with only node-bijective rules.

Approximation

The post*-effectiveness of finite-marked GTS$^{\text{bp}}$ with only node-bijective rules relies on the post*-effectiveness of finite-marked Petri nets. Similar to finite-marked Petri nets, we obtain approximation results for finite-marked GTS$^{\text{bp}}$s with only node-bijective rules in the case where Bad is negative and Safe is positive. Weak invertibility for node-bijective rules is achieved by inverting the rules.

Proposition 5.10. Finite-marked GTS$^{\text{bp}}$s with only node-bijective rules are weakly invertible.

Proof. Let $\langle S, \mathcal{R}, \text{INIT}\rangle$ be a finite-marked GTS$^{\text{bp}}$s with only node-bijective rules. For every rule $r = \langle L \rightharpoonup R\rangle \in \mathcal{R}$, we consider the inverse rule $r^{-1} = \langle R \rightharpoonup L\rangle$. Let $\mathcal{R}^{-1}$ be the set of inverse rules of $\mathcal{R}$, $\text{post}^*_{-1}(\text{INIT})$ the set of graphs reachable via $\mathcal{R}^{-1}$ from INIT, and $\text{pre}^*_{-1}(\text{INIT})$ the set of all predecessors of INIT via $\mathcal{R}^{-1}$. We can embed $\text{post}^*_{-1}(\text{INIT})$ in a node-bounded graph class S' (i.e., of bounded path length). It holds $\text{post}^*(\text{INIT}) = \text{pre}^*_{-1}(\text{INIT})$ and hence, $\uparrow\text{post}^*(\text{INIT}) = \text{pre}^*_{-1}(\text{INIT})$. By Lemma 5.1 and 2.3, we can compute the basis of $\text{pre}^*_{-1}(\text{INIT})$ for the inverted GTS. □

Example 5.14. The rules of the circular process protocol (Figure 5.5) except the `Clear`- and the `Leave`-rules are node-bijective.

As a consequence, we can perform under- and over-approximations (cp. Proposition 4.4). k_{ov} and k^{ℓ}_{un} are defined analogously to Definition 4.5.

Corollary 5.3 (approximation for node-bijective rules). For finite-marked GTS$^{\text{bp}}$s with only node-bijective rules, k_{ov} and k^{ℓ}_{un} are computable for every $\ell \geq 0$.

5.6 Related Concepts

We give an overview of results for graph transformation systems related to and used in our approach. First, we summarize related results of Bertrand et al. Afterwards, we compare the approach of König and Stückrath to our results.

Bertrand et al. [BDK$^+$12] study the (un)decidability of reachability and ideal reachability for graph transformation systems. This single paper has a number of decidability results for reachability problems in graph transformation. Some results are based on previous work. The underlying formalism is the SPO approach for labeled hypergraphs. They use, in parts, well-structuredness w.r.t. the subgraph and the minor order. A variety of rule-specific restrictions is investigated, e.g., context-free GTSs, only preserving rules, and containedness of contraction/deletion rules.

The reachability problem asks for a given GTS $\mathcal{R}$ and graphs G, H, whether $G \Rightarrow_{\mathcal{R}}^{*} H$. Bertrand et al. consider the *coverability problem* (in our context: ideal reachability problem) w.r.t. the subgraph order or the minor order, i.e., given a GTS $\mathcal{R}$ and graphs G, H, they ask whether there exists a graph $H' \succeq H$ with $G \Rightarrow_{\mathcal{R}}^{*} H'$. Here the symbol "$\preceq$" denotes one of both, the subgraph order or the minor order. We summarize selected results from [BDK+12] in the following table. For the ideal reachability problem, we only refer to their results concerning the subgraph order.

type of GTS	reachability	ideal reachability	reference [BDK+12]
context-free	+	+	Prop. 8 & 9
node-bijective	+	+	Prop. 10
preserving	+	–	Prop. 12 & 13
bounded path length	–	+	Prop. 11 & 16
edge-contracting	–	+	Prop. 14 & 17
minor rules	+	+	Prop. 14 & 17

+ = decidable, – = undecidable

For context-free and for node-bijective graph transformation systems, both problems are decidable. Interestingly, for preserving graph transformation systems, the reachability problem is decidable and the ideal reachability problem is undecidable. The opposite holds for graph transformation systems of bounded path length and for graph transformation systems containing all edge-contraction rules. For graph transformation systems containing all *minor* rules, i.e., all edge-contraction, node-, and edge-deletion rules, both problems are decidable. The investigated reachability problems are more or less related to the resilience problems which we investigate.

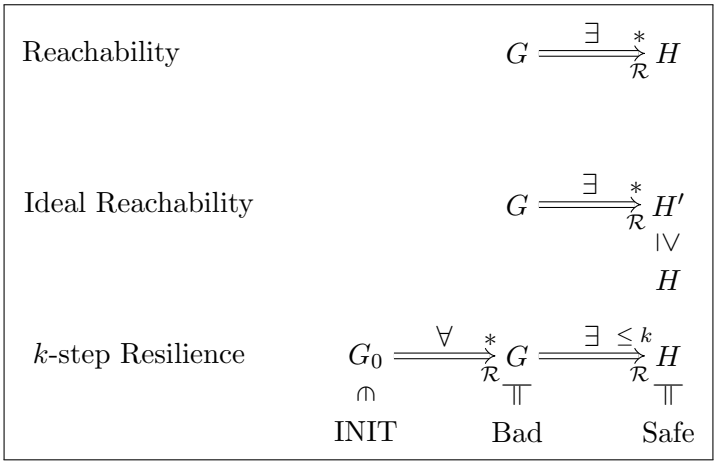


Figure 5.16: Reachability, ideal reachability, and resilience.

In Figure 5.16, the different notions are illustrated: In the resilience

problems, we consider the reachability of a graph H which satisfies the proper constraint Safe in a bounded number of steps. All graphs G which are reachable from $G_0 \in \text{INIT}$ and satisfy the proper constraint Bad have to be taken into account. For a graph H, let $\exists!H$ be the non-nested[19] constraint expressing that "H is a subgraph but no larger graph is". Let false be the negative constraint $\neg\exists\varnothing$. We can express (ideal) reachability as resilience in the following ways.

Fact 5.4. Let $\langle S, \mathcal{R}, \text{INIT}\rangle$ be a finite-marked GTS where S is the class of all graphs and INIT consists only of G_0.

(1) For every $k \geq 0$, the marked GTS is k-step resilient w.r.t. Bad $= \exists H!$ and Safe = false iff H is not reachable from G_0.

(2) For every $k \geq 0$, the marked GTS is k-step resilient w.r.t. Bad $= \exists H$ and Safe = false iff $\uparrow\{H\}$ is not reachable from G_0 (ideal reachability).

(3) $\exists k \geq 0$ s.t. the marked GTS is k-step resilient w.r.t. Bad $= \exists!G_0$ and Safe $= \exists!H$ iff H is reachable from G_0.

(4) $\exists k \geq 0$ s.t. the marked GTS is k-step resilient w.r.t. Bad $= \exists!G_0$ and Safe $= \exists H$ iff $\uparrow\{H\}$ is reachable from G_0 (ideal reachability).

Proof. (1) & (2). For Safe = false, i.e., $[\![\text{Safe}]\!] = \varnothing$, k-step resilience is equivalent to the non-reachability of $[\![\text{Bad}]\!]$:

$$\begin{aligned} & \text{post}^*(G_0) \cap [\![\text{Bad}]\!] \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]) \\ \iff \quad & \text{post}^*(G_0) \cap [\![\text{Bad}]\!] = \varnothing \end{aligned}$$

It holds $[\![\exists!H]\!] = \{H\}$ and $[\![\exists H]\!] = \uparrow\{H\}$.
(3) & (4). For Bad $= \exists!G_0$, i.e., $[\![\text{Bad}]\!] = \{G_0\}$, bounded resilience is equivalent to the reachability of $[\![\text{Safe}]\!]$:

$$\begin{aligned} & \exists k \geq 0 : \text{post}^*(G_0) \cap [\![\text{Bad}]\!] \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]) \\ \iff \quad & \exists k \geq 0 : G_0 \in \text{pre}^{\leq k}([\![\text{Safe}]\!]) \\ \iff \quad & G_0 \in \text{pre}^*([\![\text{Safe}]\!]) \end{aligned}$$

□

However, Safe = false is an unusual case.

The types of GTSs investigated in [BDK+12] show similarities to our rule-specific criteria. We use one of their results to show that bijectivity on the nodes is a sufficient criterion for post*-effectiveness.

We use the decidability of ideal reachability at the level of WSTSs as an integrant of our proofs in Chapter 4, see Figure 5.17. However, we do not use the decidability of ideal reachability for GTSs directly.

[19] $\exists!H$ is a "mixed" constraint, i.e., *not* proper.

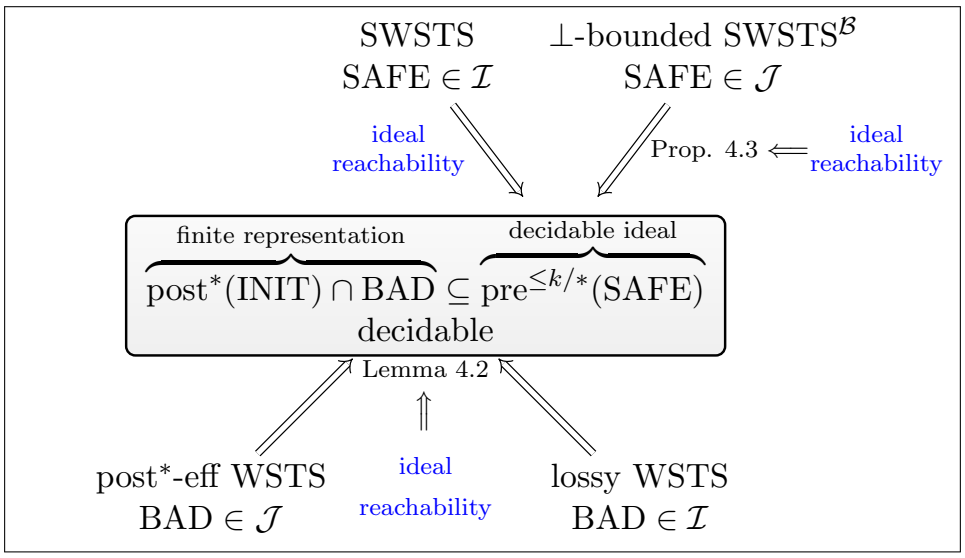


Figure 5.17: Decidability for finite-marked SWSTSs.

In contrast to [BDK+12], we stay in the framework of well-structured graph transformation systems.

In [BDK+12], a number of undecidability results are presented. E.g., ideal reachability is undecidable for preserving GTSs. As a consequence, the bounded resilience problem for finite-marked preserving[20] GTSs with ∩-based graph class is undecidable in the cases where Safe is positive.

Proposition 5.11 (undecidability). The bounded resilience problem is undecidable for preserving finite-marked GTS with ∩-based graph class in the cases where Safe is positive.

Proof. See Appendix B. □

Remark. The explicit resilience problem is decidable for preserving finite-marked GTS if Bad is negative and Safe is positive: We compute a finite representation of post*(INIT) ∩ ⟦Bad⟧$_S$ and iteratively check inclusion in pre$^{\leq k}$(⟦Safe⟧$_S$).

Joshi & König [JK08] introduced well-structuredness to the area of graph transformation. They considered graph transformation systems containing all minor rules. Following up on this, **König & Stückrath** [KS17] extensively studied the well-structuredness of graph transformation systems. They regard three types of well-quasi-orders (minor, subgraph, induced subgraph) based on results of **Ding** [Din92] and **Robertson & Seymour** [RS04].

[20] A GTS is *preserving* if each of its rules is injective on the domain.

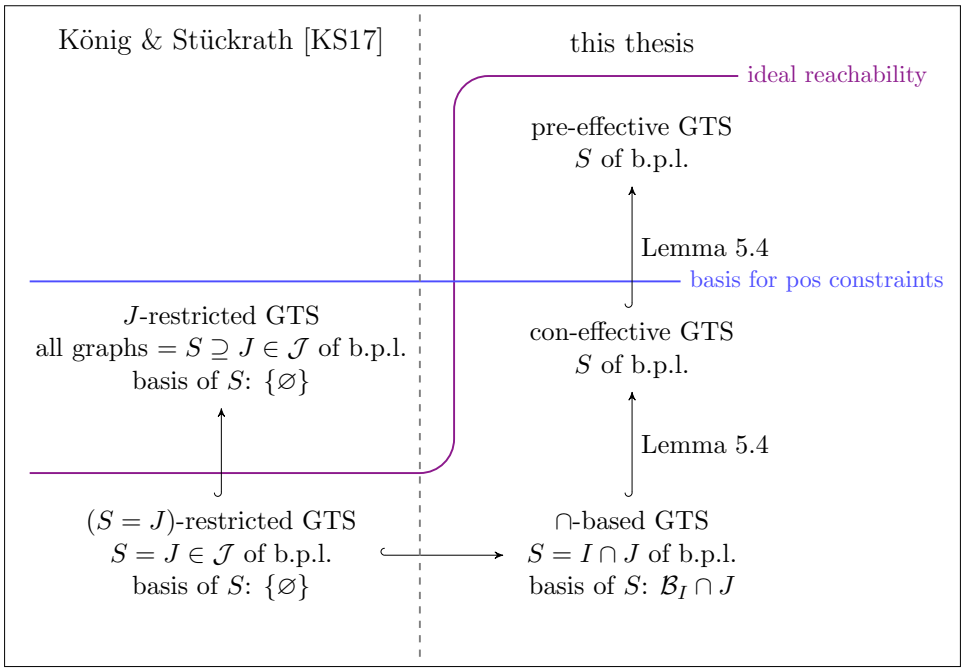
All graph transformation systems are strongly well-structured on graphs of bounded path length w.r.t. the subgraph order. This result enables us to apply our abstract results to graph transformation systems. They regard *J-restricted*[21] well-structured transition systems whose state sets have not to be a well-quasi-order but rather a subset J of the states S is a well-quasi-order. The subset J is a decidable anti-ideal. König & Stückrath develop a *backwards algorithm* based on [FS01] for J-restricted well-structured transition systems and graph transformation systems. However, decidability is achieved only for $S = J$. More detailed considerations can be found in Stückrath's thesis [Stü16].

The main contribution of König & Stückrath is the identification of criteria for well-structuredness of graph transformation systems w.r.t. several well-quasi-orders. By contrast, we provided decidability results for strongly well-structured transition systems and applied them to graph transformation systems of bounded path length. Thereby, we used the result of König & Stückrath for the subgraph order. While the backwards algorithm of König & Stückrath extends the decidability result for ideal reachability of Abdulla et al., we use it to obtain a decidability result for resilience.

Another difference is the considered graph class. For the backwards algorithm of König & Stückrath, the graph class is not relevant. For our result, we require properties which often depend on the graph class. In Figure 5.18, a comparison with our approach is given.

Except for pre-effective GTS, the basis of $[\![c]\!]_S$ is computable for every given positive constraint c. For J-restricted graph transformation systems, the graph class contains all graphs. However, decidability of ideal reachability is achieved only for $S = J$, i.e., the graph class is a decidable anti-ideal and of bounded path length. In [BDK$^+$12], Bertrand et al. show the undecidability of ideal reachability for preserving[22] graph transformation systems. We consider only graph classes of bounded path length in order to keep the decidability of ideal reachability. We prove our results for more arbitrary graph classes assumming additional effectiveness requirements. More specifically, we consider graph classes each of which is an intersection of ideal with a given basis and a decidable anti-ideal. This has the advantage that the effective procedures can be derived from the proof. In both cases, we can consider more arbitrary bases. This is crucial when one considers lossiness or $\bot$-boundedness. In the setting of [KS17], the marked GTS$^{\text{bp}}$ in the circular process protocol (Section 5.3) is not lossy: We cannot reach the empty graph.

[21] In [KS17], they speak of *Q-restricted* well-structured transition systems.
[22] In [BDK$^+$12], preserving rules are called *non-deleting*.



b.p.l. = bounded path length, pos = positive, $\mathcal{B}_I$: basis of I

Figure 5.18: Comparison of the approaches in [KS17] and in this thesis.

By contrast, we showed that the marked GTS$^{\text{bp}}$ is lossy if we choose the basis appropriately.

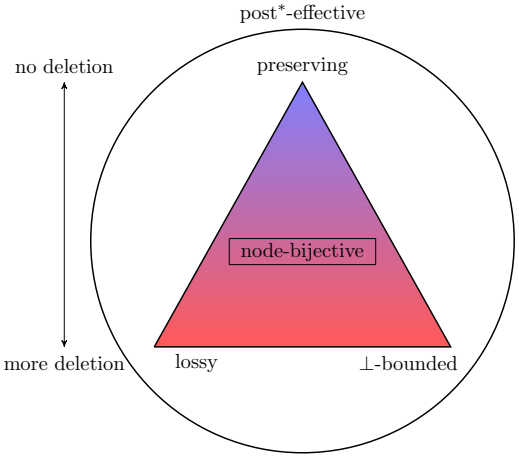
5.7 Summary

We translated our decidability results from Chapter 4 into the graph-transformational setting. The used well-quasi-order is the subgraph order which comes with the restriction of bounded path lengths. Instead of ideal-based sets, we input proper graph constraints. In every case, the graph class is of bounded path length and the marked GTS$^{\text{bp}}$ is post*-effective, lossy, or $\bot$-bounded, respectively. We considered $\cap$-based graph classes to ensure well-structuredness. More generally, the results hold for arbitrary graph classes of bounded path length (case: post*-effectiveness), con-effective graph classes of bounded path length (case: lossy) pre-effective marked GTS$^{\text{bp}}$s (case: $\bot$-bounded). Furthermore, we identified sufficient, rule-specific criteria for the requirements of post*-effectiveness, lossiness, and $\bot$-boundedness. In summary, we obtained the following results for marked GTS$^{\text{bp}}$.

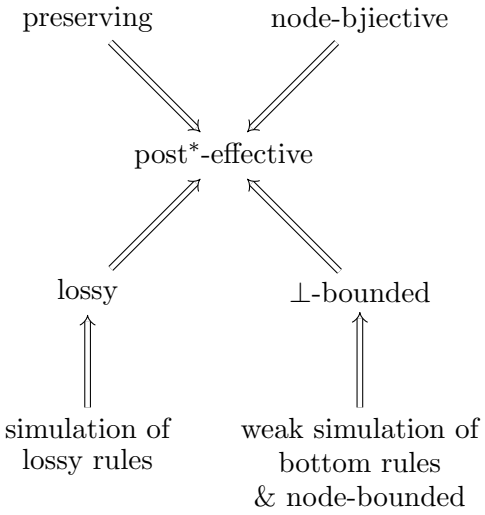
type of GTS	rule-specific criterion	graph class	Bad	Safe	bounded	explicit
post*-eff	preserving or node-bijective	bounded path length	negative	positive	+	+
lossy	sim lossy rules	c-eff & bnd path length	positive	positive	+	+
lossy & ⊥-bnd	sim lossy rules & w-sim bottom rules	c-eff & node-bnd	positive	negative	+	∘
pre-eff & ⊥-bnd	w-sim bottom rules	basis given & node-bnd	negative	negative	+	∘

eff = effective, bnd = bounded, sim = simulation of, w-sim = weak simulation of, + = decidable, ∘ = unknown

The requirements (and rule-specific criteria) cover a larger class of GTSs, as depicted in the diagrammatic representation below. Intuitively speaking, GTSs are classified according to their "degree of deleting". Preserving rules do not delete any items. On the other side of the "spectrum", lossiness and ⊥-boundedness are located. These kinds of GTSs exhibit rules which delete items.[23]



The diagram below illustrates these subclasses in more detail.



[23]We consider here only lossiness in the sense of (simulation of) lossy rules.

If all rules are preserving or all rules are node-bijective, post*-effectiveness is satisfied. If all lossy rules can be simulated by the original rules, lossiness is satisfied. If all bottom rules can be weakly simulated by the original rules and the number of nodes is bounded, $\bot$-boundedness is satisfied.

Preserving rules, e.g., as in Example 5.2 (starry sky), are often used for generating objects. Node-bijective rules, e.g., as in Example 3.1 (traffic network), are common in models with a rather "static" topology. In Section 5.3, we consider a circular process protocol as an example which satisfies lossiness, $\bot$-boundedness, and hence post*-effectiveness. In Section 5.4, we give examples of logistic systems each of which satisfies one requirement, i.e., post*-effectiveness, lossiness, or $\bot$-boundedness, respectively.

Takeaway (Chapter 5).

- The bounded resilience problem for GTSs asks: Given a GTS, a set INIT of graphs, graph constraints Bad, Safe, and is there a bound k s.t. starting from any INITial graph, whenever we reach a Bad graph, we can reach a Safe graph in $\leq k$ steps? This problem is decidable for GTSs of bounded path length satisfying additional requirements: post*-effective, lossy, $\bot$-bounded; INIT finite; Bad, Safe proper.
- The explicit resilience problem for GTSs is formulated as the bounded resilience problem, except that a bound $k \geq 0$ is given. It is decidable for finite-marked GTSs of bounded path length in the cases where INIT is finite and Safe is positive.
- Sufficient criteria for the requirements:

preserving or node-bijective	implies	post*-effective
simulation of lossy rules	implies	lossy
weak simulation of bottom rules & node-bounded	implies	$\bot$-bounded

---

Chapter 6

Conclusion

We recapitulate related concepts, summarize the results of this thesis, and give an outlook for further topics.

6.1 Related Concepts

We summarize related work, starting with concepts generally related to adverse conditions and resilience.

Correctness and *verification* of programs are addressed by **Apt et al.** [AOdB09]. In **Pennemann** [Pen09] and **Poskitt & Plump** [PP13], correctness of graph programs is considered. System correctness under *adverse conditions* is a topic of recent research, see **Olderog et al.** [OFTK21]. This concept addresses systems which interact with an environment. To the best of our knowledge, there is little research on adverse conditions in the context of graph transformation, see, e.g., **Flick** [Fli16] and **Peuser** [Peu18].

The concept of resilience is broadly used in different areas with varying definitions, see, e.g., **Trivedi et al.** [TKG09] and **Jackson & Ferris** [JF13]. Our notion of resilience captures recovery of a safety constraint in a limited number of steps (in a specified window). To the best of our knowledge, there is only little research on similar resilience problems, e.g., for timed automata, see **Akshay et al.** [AGH$^+$21].

The following concepts and results are the basis upon which we built our decidability theory.

Abdulla et al. [AČJT96] show the decidability of ideal reachability, eventuality properties and simulation in (labeled) strongly well-structured transition systems. We use the presented algorithm as an integrant of our decidability proof for well-structured transition systems. **Finkel & Schnoebelen** [FS01] show that the concept of well-structuredness is ubiquitous in computer science by providing a large class of example models (e.g.,

Petri nets and their extensions, communicating finite state machines, lossy systems, basic process algebras). Moreover, they give several decidability results for well-structured systems with varying notions of compatibility. They also generalize the algorithm of [AČJT96] to (not necessarily strongly) well-structured transition systems to show decidability of ideal reachability.

For modeling systems, we use single-pushout graph transformation as in **Löwe** [Löw91] (see also **Ehrig et al.** [EHK+97]).

Bertrand et al. [BDK+12] study the decidability of reachability and ideal reachability for graph transformation systems. This single paper has a number of decidability results for reachability problems in graph transformation. The underlying formalism is the SPO approach for labeled hypergraphs. They use, in parts, well-structuredness w.r.t. the subgraph and the minor order. A variety of rule-specific restrictions is investigated, e.g., preserving GTSs and containedness of contraction/deletion rules. We use one of their results to show that bijectivity on the nodes is a sufficient criterion for post*-effectiveness.

König & Stückrath [KS17] extensively study the well-structuredness of graph transformation systems. More detailed considerations can be found in Stückrath's thesis [Stü16]. They identify three types of well-qasi-orders (minor, subgraph, induced subgraph) on graphs based on results of **Ding** [Din92] and **Robertson & Seymour** [RS04]. The fact that the subgraph order is a well-qasi-order on graphs of bounded path length while the minor order allows all graphs comes with a trade-off: For obtaining well-structuredness w.r.t. the minor order, the graph transformation system must contain all edge contraction rules, i.e., it must be "minor-lossy". (Note that we consider lossiness of graph transformation systems in the context of the subgraph order.) On the other hand, all graph transformation systems are strongly well-structured on graphs of bounded path length w.r.t. the subgraph order. This result enables us to apply our results for well-structured transition systems to graph transformation systems. They also generalize the notion of well-structured transition systems by regarding J-restricted well-structured transition systems whose state sets need not to be a well-quasi-order but rather a subset J of the states is a well-quasi-order. König & Stückrath develop a backwards algorithm based on [FS01] for J-restricted well-structured transition systems obtaining a method to check coverability under additional assumptions. For strongly well-structured transition systems, this approach coincides with the ideal reachability algorithm [AČJT96].

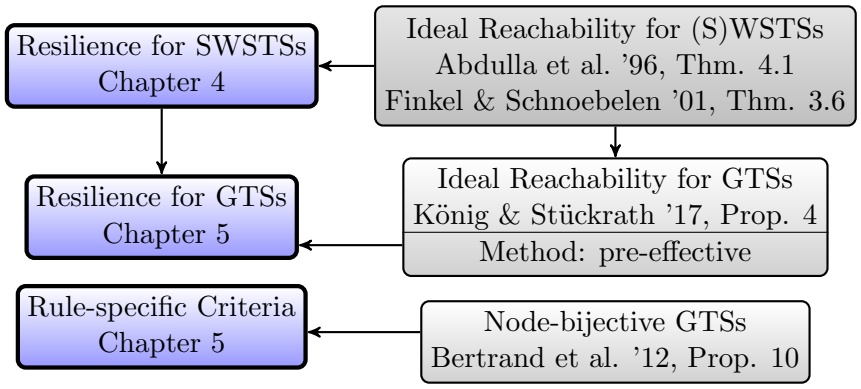


Figure 6.1: Our decidability results in the context of known results.

In Figure 6.1, the main decidability results (bold boxes) are placed in the context of known results. The arrows mean "used for" or "based on". We use the decidability of ideal reachability for well-structured transition systems [AČJT96, FS01] as an integrant for the decidability of resilience for well-structured transition systems. The method for pre-effectiveness of graph transformation systems w.r.t. the subgraph order [KS17] is used in our decidability results for graph transformation systems. We use result of [BDK$^+$12] to show that bijectivity on the nodes is a sufficient criterion for post*-effectiveness.

6.2 Summary

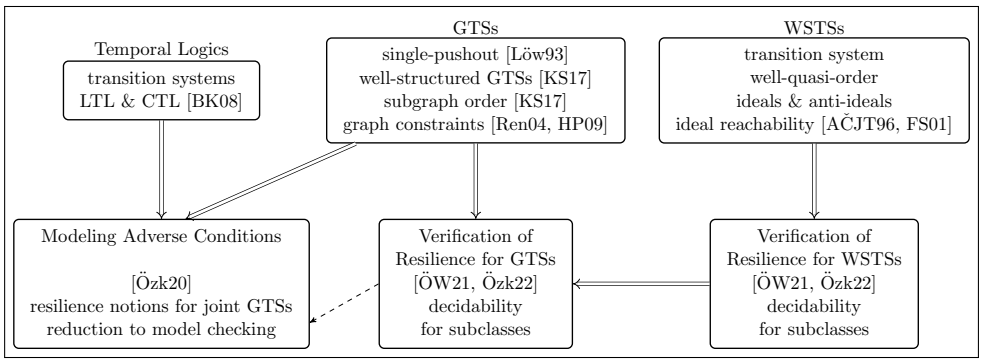
First, we considered resilience in the context of adverse conditions. We provided (1) a construction of joint graph transformation systems for modeling adverse conditions, (2) meaningful notions of resilience for this kind of graph transformation systems, and (3) showed how resilience can be expressed in temporal logics. A natural notion of resilience is k-step resilience which captures recovery of the safety constraint in $\leq k$ steps.

We considered k-step resilience in more detail and investigated decidability for graph transformation systems. We provided (1) decidability results for subclasses of well-structured transition systems and ideal-based subsets BAD and SAFE. By applying them, we obtained (2) decidability results for subclasses of graph transformation systems of bounded path length and proper graph constraints Bad and Safe. The requirements corresponding to these subclasses are effectiveness conditions or a kind of unreliability. We identified (3) sufficient, rule-specific criteria for decidability. Ultimately, we applied our results to joint graph transformation systems, i.e., to the setting of adverse conditions.

Our decidability results for graph transformation systems as well as for well-structured transition systems are – to the best of our knowledge – new.

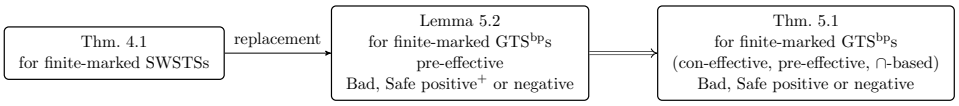
These results may pave the way for automatic verification of resilience, a topic of high relevance. Up to now, there is only little research on the decidability of related resilience problems.

The main concepts used in this thesis are the established theories of graph transformation systems, well-structured transition systems, and to a smaller extent also temporal logics.

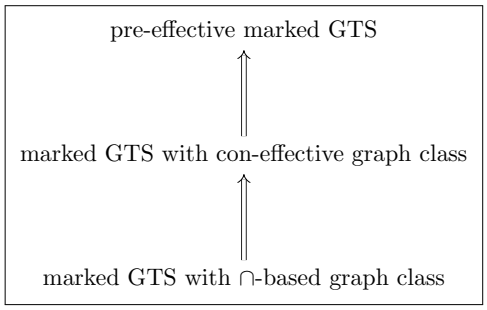


In more detail, this work comprises three main aspects: the modeling of adverse conditions, the verification of resilience for well-structured transition systems, and the verification of resilience for graph transformation systems. The modeling of adverse conditions is based on graph transformation systems, graph constraints, and temporal logics. The verification of resilience for well-structured transition systems is based on the theory of well-structuredness which uses the concept of ideals in well-quasi-orders. The verification of resilience for graph transformation systems is based on our results for well-structured transition systems and the theory of graph transformation systems. Our results for graph transformation systems can be used for the verification in the context of adverse conditions.

The main theorems for bounded resilience are derived from our decidability theorem for finite-marked SWSTSs (Theorem 4.1).



By replacing "marked SWSTSs" with "pre-effective GTSs of bounded path length", we derive a decidability result for pre-effective finite-marked GTSs of bounded path length. However, if Bad (Safe) is positive, we require a given basis of the set of graphs satisfying Bad (Safe). This result is improved by the decidability theorem which requires not always pre-effectiveness but also con-effectiveness ($\cap$-basedness) and proper constraints Bad and Safe. The relation between the effectiveness requirements on the graph class and the GTS is as follows: $\cap$-based implies con-effective implies pre-effective.



By these theorems, we also obtain decidability results for example classes: Reset Petri nets can be seen as an example for SWSTSs (and GTSs), and joint GTSs are GTSs. Moreover, we identify sufficient, rule-specific criteria for the requirements post*-effective, lossy, and $\perp$-bounded.

6.3 Further Topics

As further topics one may consider generalizations & extensions of this approach, alternative methods, complexity & approximations, an implementation, and undecidability.

Generalizations & Extensions, i.e.,

- more general graph constraints, e.g., nested graph constraints,
- more general graph transformation systems, e.g., rules with negative application conditions [HHT96], and
- other well-quasi-orders on graphs, e.g., the induced subgraph order considered by König & Stückrath [KS17].

A result on "mixed" constraints Bad and Safe expressing the existence and absence of specified subgraphs is desirable. To solve resilience problems for nested graph constraints, alternative methods might be useful. In [HST18], an automata-based approach for integration of graph constraints into graph grammars is presented. This could be exploited for automata-based methods. Furthermore, methods for handling graph classes of unbounded path length are desirable.
Using the induced subgraph order would allow us to use a restricted type of nested graph constraints and restricted negative application conditions. However, the graph class for the induced subgraph order is more restrictive. "Minor-lossy" graph transformation systems [KS17] are well-structured w.r.t. to minor order which is a well-quasi-order on the class of all graphs. However, our main objective is to solve the bounded resilience problem. For this, one requires at least "bounded" compatibility (we use strong compatibility, i.e., the bound is equal to 1).

Complexity & Approximations. An investigation on the complexity of the algorithms in Section 4.3 may achieve more clarity regarding the exact complexity bounds for subclasses of graph transformations systems, e.g., for node-bijective rules. In cases where the complexity is considerable, approximations are helpful for handling costly computations. Approximations of graph transformations by Petri nets, e.g., Baldan et al. [BCK08], may be taken into account.

Implementation. The algorithms presented in Section 4.3 can be used for an implementation based on UNCOVER (Stückrath [Stü15]) which provides an implementation of the ideal reachability algorithm w.r.t. subgraph order. An obstacle may be that the post*-effectiveness of Petri nets relies on the reachability algorithm for Petri nets. This might be resolved using approximations.

Undecidability. To point out the limits of this approach, undecidability results are favorable. The objective is to narrow down the decidability gap. Based on the results of Bertrand et al. [BDK$^+$12], one may show that, in some cases, the lack of a requirement or the presence of unbounded paths yields the undecidability of the bounded (or explicit) resilience problem for graph transformation systems. Also undecidability results for reset Petri nets (reachability, boundedness, deadlock-freeness, see, Dufourd et al. [DFS98] and Akshay et al. [ACD$^+$17]) seem to be good sources of undecidability for graph transformation systems.

Appendix A

Related Formalisms

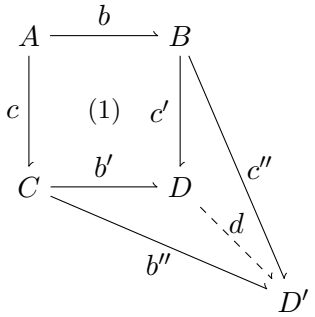
We give an overview of related formalisms. Formal notions of pushouts, graph constraints, and temporal logics are included.

Pushouts

The notion of a pushout is usually used in the context of category theory, see, e.g., Ehrig et al. [EPS73, EHK+97, EEPT06] or Löwe [Löw91]. We give a definition and a construction of pushouts on the levels of graphs and morphisms. Moreover, we state used properties of pushouts.

The definitions in this section follow Ehrig et al. [EHK+97] and the constructions follow Stückrath [Stü16]. For the proofs, see the corresponding reference.

Definition A.1 (pushout). A *pushout (PO)* of morphisms (partial functions) $b : A \rightharpoonup B$ and $c : A \rightharpoonup C$ is a diagram as (1) below with $c' \circ b = b' \circ c$ s.t. the *universal property* holds: for all morphisms (partial functions) $b'' : C \rightharpoonup D'$ and $c'' : B \rightharpoonup D'$ with $c'' \circ b = b'' \circ c$, there exists a unique morphism (partial function) $d : D \rightharpoonup D'$ s.t. $d \circ c' = c''$ and $d \circ b' = b''$. The graph (set) D is called *pushout object*.



Using the definition of a pushout, one can show that pushouts are unique.

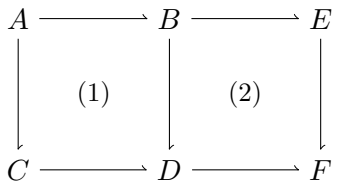
Lemma A.1 (uniqueness of pushouts [EEPT06, Fact 2.20]). Pushout objects are unique up to isomorphism.

Pushouts can be composed and decomposed.

Lemma A.2 (composition of pushouts [EEPT06, Fact 2.20]).

(i) If (1) and (2) are pushouts, then (1)+(2) is a pushout.

(ii) If (1) and (1)+(2) are pushouts, then (2) is a pushout.



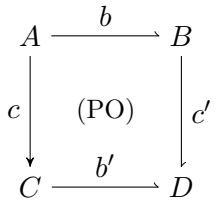
The following properties of pushouts are often used in proofs.

Lemma A.3 (properties of pushouts [EHK$^+$97, Lemma 8]). Let the diagram below be a pushout where $c : A \to C$ is total.

(i) If b is injective, then b' is injective.

(ii) If b is surjective, then b' is surjective.

(iii) If c is injective, then c' is total and injective.



We depict a set-theoretic construction of pushouts.

Construction A.1 (pushout of sets [Stü16, Prop. 3.24]). For sets A, B, C and (partial) functions $b : A \rightharpoonup B$, $c : A \hookrightarrow C$, the pushout is constructed as follows: Let $(B + C)/\sim$ be a quotient of the disjoint union[24] $B + C$ where $\sim$ is the smallest equivalence relation on $B + C$ s.t. $b(x) \sim c(x)$ for every $x \in A$. For $x \in B + C$, let $[x] = \{y \in B + C : x \sim y\}$ be the equivalence class of x. We say that an equivalence class is *valid* if it contains no element $c(x)$ for which $b(x)$ is undefined. The set D is consisting of all valid equivalence classes of $(A + B)/\sim$. Let $b' : C \rightharpoonup D$ be the partial function given by $b'(x) = [x]$ if $[x]$ is valid, otherwise $b'(x)$ is undefined. Let $c' : B \hookrightarrow D$ be the function given by $c'(x) = [x]$. The pushout object is the set D.

[24] W.l.o.g., we assume that B and C are disjoint, meaning that $B + C = B \cup C$.

Intuitively, the pushout object of $b : A \rightharpoonup B$ and $c : A \hookrightarrow C$ is the disjoint union of B and C where we leave out elements for which b is undefined and identify elements with the same preimage.

This construction can be carried over to graphs "componentwise".

Construction A.2 (pushout of graphs [Stü16, Prop. 3.24]). For graphs L, R, G and morphisms $p : L \rightharpoonup R$, $m : L \hookrightarrow G$, the pushout is constructed componentwise on nodes and edges: Let H be the graph given by

$$
\begin{aligned}
V_H &\subseteq (V_G + V_R)/\sim \\
E_H &\subseteq (E_G + E_R)/\sim
\end{aligned}
$$

$$
\mathrm{src}_H([e]) = \begin{cases} [\mathrm{src}_G(e)] & \text{if } e \in E_G \\ [\mathrm{src}_R(e)] & \text{if } e \in E_R \end{cases}
$$

$$
\mathrm{tgt}_H([e]) = \begin{cases} [\mathrm{tgt}_G(e)] & \text{if } e \in E_G \\ [\mathrm{tgt}_R(e)] & \text{if } e \in E_R \end{cases}
$$

$$
\mathrm{lab}^V_H([v]) = \begin{cases} \mathrm{lab}^V_G(v) & \text{if } v \in V_G \\ \mathrm{lab}^V_R(v) & \text{if } v \in V_R \end{cases}
$$

$$
\mathrm{lab}^E_H([e]) = \begin{cases} \mathrm{lab}^E_G(e) & \text{if } e \in E_G \\ \mathrm{lab}^E_R(e) & \text{if } e \in E_R \end{cases}
$$

where V_H consists of all valid equivalence classes of nodes and E_H consists of all valid equivalence classes of edges with incident nodes v, v' s.t. $[v], [v'] \in V_H$. Let $p' : G \rightharpoonup H$ be the morphism given by $p'(x) = [x]$ if $[x] \in V_H \cup E_H$, otherwise $p'(x)$ is undefined, for every item x of G. Let $m' : R \hookrightarrow H$ be the morphism given by $m'(x) = [x]$ for every item x of R. The pushout object is the graph H.

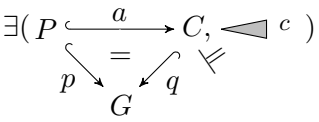
$$
\begin{array}{ccc}
L & \xrightarrow{\;p\;} & R \\
{\scriptstyle m}\downarrow & \text{(PO)} & \downarrow{\scriptstyle m'} \\
G & \xrightarrow{\;p'\;} & H
\end{array}
$$

By Lemma A.1, pushout objects are unique up to isomorphism. Therefore, the latter construction is "canonical". Moreover, it shows that pushout of graphs are effectively constructible.

Graph Constraints

Graph conditions are nested constructs, which can be represented as trees of morphisms equipped with quantifiers and Boolean connectives. Graph conditions and first-order graph formulas are expressively equivalent [HP09]. Graph constraints are special graph conditions.

Definition A.2 (graph conditions). The class of *(nested) graph conditions* over a graph P is defined inductively: (i) true is a graph condition over P, (ii) $\exists(a, c)$ is a graph condition over P where $a : P \hookrightarrow C$ is an injective morphism and c is a condition over C, (iii) for conditions c, c' over P, $\neg c$ and $c \wedge c'$ are conditions over P. Conditions over the empty graph $\varnothing$ are called *(nested) graph constraints*.
The semantics of graph conditions are defined inductively: (i) Any injective morphism $p\colon P \hookrightarrow G$ *satisfies* true.
(ii) An injective morphism p *satisfies* $\exists(a, c)$ with $a\colon P \hookrightarrow C$ if there exists an injective morphism $q\colon C \hookrightarrow G$ such that $q \circ a = p$ and q satisfies c.



(iii) An injective morphism p *satisfies* $\neg c$ if p does not satisfy c, and p *satisfies* $c \wedge c'$ if p satisfies both c and c'.
We write $p \models c$ if p satisfies the condition c (over P). A graph G *satisfies* a constraint c, $G \models c$, if the morphism $p\colon \varnothing \hookrightarrow G$ satisfies c.

Graph conditions may be written in a more compact form: $\exists a$ abbreviates $\exists(a, \text{true})$, false abbreviates $\neg$true. The expressions $c \vee c'$ and $c \Rightarrow c'$ are defined as usual. For an injective morphism $a : P \hookrightarrow C$ in a condition, we just depict the codomain C if the domain P can be unambiguously inferred. E.g., the constraint $\forall($ $\circ, \exists($ $\circlearrowleft$ $))$ expresses that every node has a loop.

Temporal Logics

Temporal formulas such as LTL and CTL formulas are well-known in logic, see, e.g., [CE82, Eme90, BK08]. We adapt the notions and consider so-called LTL and CTL graph constraints [Peu18], i.e., temporal formulas whose atoms equate to graph constraints.

First, we define LTL graph constraints and their semantics. The temporality is interpreted as the changes along a transformation sequence. Every direct transformation correlates to a time step. Besides the common propositional operators there are *temporal operators*, e.g., the operator $\mathbf{X}$ (*Ne**X**t*) describes the validity of a formula in the next step while $\mathbf{G}$ (***G**lobally*) describes the validity of a formula in every following step. The operator $\mathbf{U}$

(***U**ntil*) describes the validity of a first formula until a second formula is valid.

Definition A.3 (LTL graph constraints). The class LTL of *linear temporal logic (graph) constraints* is the smallest class of expressions, which (i) contains all graph constraints, (ii) is closed under the *propositional operators* $\neg, \wedge, \vee, \Rightarrow$, i.e., for all $\phi, \psi \in$ LTL, $\neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \Rightarrow \psi \in$ LTL, and (iii) is closed under the *temporal operators* $\mathbf{X}, \mathbf{G}, \mathbf{U}$, i.e., for all $\phi, \psi \in$ LTL, $\mathbf{X}\phi, \mathbf{G}\phi, \phi\mathbf{U}\psi \in$ LTL.
The semantics of LTL constraints is defined for infinite *paths*, i.e., infinite transformation sequences: Let P be an infinite path $G_0 \Rightarrow \ldots$. The *satisfaction* of LTL constraints in P, denoted by $\models$, is defined inductively:
(i) For a graph constraint c, $G_i \models c$ if G_i satisfies c as graph constraint.
(ii) The semantics for the propositional operators are as usual, e.g., for $\phi \in$ LTL, $G_i \models \neg\phi$ if $G_i \not\models \phi$.
(iii) For all $\phi, \psi \in$ LTL,
(a) $G_i \models \mathbf{X}\phi$ if $G_{i+1} \models \phi$,
(b) $G_i \models \mathbf{G}\phi$ if $G_j \models \phi$ for all $j \geq i$,
(c) $G_i \models \phi\mathbf{U}\psi$ if there is $k \geq i$ s.t. $G_k \models \psi$ and $G_j \models \phi$ for all $i \leq j < k$.
For a LTL constraint ϕ, P *satisfies* ϕ, in symbols $P \models \phi$, if $G_0 \models \phi$.
A marked GTS $\langle S, \mathcal{R}, \text{INIT}\rangle$ *satisfies* ϕ, in symbols $\langle S, \mathcal{R}, \text{INIT}\rangle \models \phi$, if $P \models \phi$ for all infinite paths $P = \langle G_0 \Rightarrow_{\mathcal{R}} \ldots\rangle$ with $G_0 \in$ INIT.

CTL graph constraints are, like LTL graph constraints, temporal formulas where the atoms equate to the graph constraints. By contrast, the temporality is here branching. Besides the common propositional operators, there are *path-quantified temporal operators* which are pairs of operators: the first one is either **A** (*for **A**ll following paths*) or **E** (*there **E**xists a following path*), the second one is a temporal operator. The operator **AG** means the valdity of a formula in all following sequences, and **EF** means that a graph can be reached where the formula is valid.

Definition A.4 (CTL graph constraints). The class CTL of *computation tree logic (graph)* constraints is the smallest class of expressions, which (i) contains all graph constraints, (ii) is closed under the *propositional operators* $\neg, \wedge, \vee, \Rightarrow$, i.e., for all $\phi, \psi \in$ CTL, $\neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \Rightarrow \psi \in$ CTL, and (iii) is closed under the *path-quantified temporal operators* $\mathbf{AX}, \mathbf{EX}, \mathbf{AG}, \mathbf{EG}, \mathbf{AU}, \mathbf{EU}$, i.e., for all $\phi, \psi \in$ CTL, $\mathbf{AX}\phi, \mathbf{EX}\phi, \mathbf{AG}\phi, \mathbf{EG}\phi, \phi\mathbf{AU}\psi, \phi\mathbf{EU}\psi \in$ CTL.
The *satisfaction* of CTL constraints in a marked GTS $\langle S, \mathcal{R}, \text{INIT}\rangle$ is defined inductively:
(i) For a graph constraint c and $G \in S$, $G \models c$ if G satisfies c as graph constraint.
(ii) The semantics for the propositional operators are as usual, e.g., for $\phi \in$ CTL and $G \in S$, $G \models \neg\phi$ if $G \not\models \phi$.

(iii) For $\phi, \psi \in \text{CTL}$, $G \in S$ and $\boldsymbol{O} \in \{\mathbf{X}, \mathbf{G}, \mathbf{U}\}$,

(a) $G \models \mathbf{A}\boldsymbol{O}\phi$ ($\models \phi\mathbf{A}\boldsymbol{O}\psi$) if for all infinite paths P via $\Rightarrow_{\mathcal{R}}$, which start in G, P satisfies ϕ ($\phi\boldsymbol{O}\psi$) in the LTL-sense, e.g., $G \models \mathbf{AG}\phi$ means: for every infinite path $G = G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \ldots$, for every $i \geq 0$, $G_i \models \phi$.

(b) $G \models \mathbf{E}\boldsymbol{O}\phi$ ($\models \phi\mathbf{E}\boldsymbol{O}\psi$) if there is an infinite path via $\Rightarrow_{\mathcal{R}}$, which starts in G and satisfies ϕ ($\phi\boldsymbol{O}\psi$) in the LTL-sense, e.g., $G \models \mathbf{EG}\phi$ means: there is an infinite path $G = G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \ldots$ s.t. for every $i \geq 0$, $G_i \models \phi$.

A marked GTS $\langle S, \mathcal{R}, \text{INIT}\rangle$ *satisfies* ϕ, in symbols $\langle S, \mathcal{R}, \text{INIT}\rangle \models \phi$ if $G_0 \models \phi$ for all $G_0 \in \text{INIT}$.

Reset Petri Nets

Petri nets, considered, e.g., by Reisig [Rei85], are a common model for discrete, distributed systems in computer science, often applied, e.g., in logistics or supply chains.

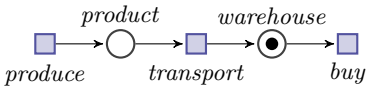
Definition A.5 (Petri net). A *Petri net* (also called *place/transition net*) is a tuple $\langle P, T, F\rangle$ with disjoint finite sets of *places* P and *transitions* T, and a *flow function* $F : (P\times T)\cup(T\times P) \to \mathbb{N}$. A *marking* is a function $M : P \to \mathbb{N}$ which indicates the number of tokens on each place. $F(x, y) = n > 0$ means that there is an *arc* of *weight* n from x to y describing the flow of tokens in the net. A transition $t \in T$ is *enabled* in a marking M if $\forall p \in P : F(p, t) \leq M(p)$. If t is enabled, then t can *fire* in M, leading to a new marking M' calculated by $\forall p \in P : M'(p) = M(p) - F(p, t) + F(t, p)$. This is denoted by $M[t\rangle M'$. Usually, a Petri net is equipped with a set of *initial markings* INIT (in the literature often a single marking). The tuple $\langle P, T, F, \text{INIT}\rangle$ is then called a *finite-marked* Petri net (in the literature often called a "marked Petri net").

A Petri net can be visually represented by a bipartite graph where places and transitions are the nodes, and the flow is indicated by edges labeled with $F(x, y)$ (if $F(x, y) = 0$, we do not draw an edge between x and y). Markings are represented by adding tokens as black dots in the corresponding places.

Example A.1. Consider a Petri net consisting of the two places *product* and *warehouse* and the three transitions *produce*, *transport*, and *buy*. The flow function is given by

$$\begin{aligned} & F(\mathit{produce}, \mathit{product}) = F(\mathit{product}, \mathit{transport}) \\ = \;& F(\mathit{transport}, \mathit{warehouse}) = F(\mathit{warehouse}, \mathit{buy}) = 1 \end{aligned}$$

and is equal to 0 on all other place/transition pairs. This Petri net (together with a marking) can be represented as below.



The marking M in the latter representation is given by $M(\mathit{product}) = 0$ and $M(\mathit{warehouse}) = 1$.

Every Petri net $\langle P, T, F\rangle$ induces a transition system with the states S given by $\mathcal{M}$, the set of all markings of $M : P \to \mathbb{N}$, and the transitions $\to$ given by $M \to M' \iff \exists t \in T : M[t\rangle M'$. Together with the well-quasi-order $\leq_{\mathrm{PN}}$, given by $\forall M, M' \in \mathcal{M} : M \leq_{\mathrm{PN}} M' :\iff \forall p \in P : M(p) \leq M'(p)$, this constitutes a SWSTS, see, e.g., Finkel & Schnoebelen [FS01, Thm. 6.1].

Fact A.1. Petri nets constitute post*-effective SWSTSs.

Proof. Reachability for Petri nets is decidable [May84] and recursively equivalent to submarking reachability [Hac85]. This corresponds to the anti-ideal reachability problem for Petri nets. Thus, by Proposition 4.1, finite-marked Petri nets are post*-effective. □

For Petri nets, reachability and equivalent problems are decidable, see, e.g., Esparza & Nielsen [EN94].

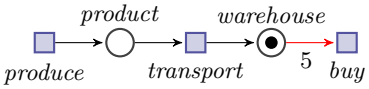
Reset Petri nets, considered, e.g., by Dufourd et al. [DFS98], are a generalization of Petri nets. A set of *reset arcs* $R \subseteq P \times T$ from places to transitions is added. When the corresponding transition fires, the number of tokens in the originating place is set to zero.

Definition A.6 (reset Petri net). A *reset Petri net* is a tuple $\langle P, T, R, F\rangle$ where $\langle P, T, F\rangle$ is a Petri net and is a $R \subseteq P \times T$ is a set of *reset arcs*. The firing in a reset Petri net is defined slightly different than in a Petri net: If $t \in T$ is enabled in a marking M, then the new marking M' is calculated by $\forall p \in P : \langle p, t\rangle \in R : M'(p) = F(t, p)$ and $\forall p \in P : \langle p, t\rangle \notin R : M'(p) = M(p) - F(p, t) + F(t, p)$.

Example A.2. Consider a reset Petri net consisting of the two places *product* and *warehouse* and the three transitions *produce*, *transport*, and *buy*. The flow function is given by

$$\begin{aligned} & F(\mathit{produce}, \mathit{product}) = F(\mathit{product}, \mathit{transport}) \\ = \;& F(\mathit{transport}, \mathit{warehouse}) = 1, \end{aligned}$$

$F(\mathit{warehouse}, \mathit{buy}) = 5$, and is equal to 0 on all other place/transition pairs. The only reset arc is the place/transition pair $\langle \mathit{warehouse}, \mathit{buy}\rangle$. This reset Petri net (together with a marking) can be represented as below.



The transition *buy* is enabled iff there are at least 5 tokens at the place *warehouse*. If it fires, the number of tokens at the place *warehouse* is set to zero.

The well-structuredness of Petri nets can be transferred to reset Petri nets, see, e.g., Dufourd et al. [DFS98, p. 108].

Fact A.2. Reset Petri nets constitute SWSTSs which are, in general, not post*-effective.

Proof. See proof of Proposition B.1. □

Appendix B

Proofs

We provide the proofs omitted in the chapters. These include a normal form lemma for positive constraints, reductions to temporal logics, undecidability results, a criterion for post*-effectiveness of marked joint graph transformation systems, and decidability results for $\mathcal{I}$-marked SWSTSs.

Normal Form

We used that for every positive constraint, we can effectively construct an equivalent, reduced $\vee$-normal form.

Lemma 2.1 ($\vee$-normal form). For every positive constraint, we can effectively construct an equivalent positive constraint of the form $\bigvee_{1 \leq i \leq n} \exists C_i$.

Proof (by structural induction over positive constraints). Let c be a positive constraint.
Hypothesis. An equivalent positive constraint c' of the form $\bigvee_{1 \leq i \leq n} \exists C_i$ is constructible.
Base case. For $c = \exists C$, the statement holds.
Induction step. (i) Let $c = c_1 \vee c_2$ and assume that the induction hypothesis holds for c_1 and c_2. Then, c is equivalent to $c_1' \vee c_2'$ where c_1' and c_2' are positive constraint of the form $\bigvee_{1 \leq i \leq n} \exists C_i$, which are equivalent to c_1 and c_2, respectively. Thus, the statement holds for c.
(ii) Let $c = c_1 \wedge c_2$ and assume that the induction hypothesis holds for c_1 and c_2. Then, c_1 and c_2 are equivalent to positive constraints $\bigvee_{C_1 \in \mathcal{B}_1} \exists C_1$ and $\bigvee_{C_2 \in \mathcal{B}_2} \exists C_2$, respectively. Thus, c is equivalent to

$$\left(\bigvee_{C_1 \in \mathcal{B}_1} \exists C_1 \right) \wedge \left(\bigvee_{C_2 \in \mathcal{B}_2} \exists C_2 \right).$$

We transform the latter positive constraint into an equivalent positive con-

straint in disjunctive normal form[25], i.e.,

$$\bigvee_{C_1 \in \mathcal{B}_1, C_2 \in \mathcal{B}_2} (\exists C_1 \wedge \exists C_2).$$

The next step is the elimination of $\wedge$. Let $\mathrm{Merge}(C_1, C_2)$ be the set of all graphs C' s.t. there exist total, injective morphisms $C_1 \hookrightarrow C'$, $C_1 \hookrightarrow C'$ which are jointly surjective, i.e., every item of C' has a preimage in C_1 or C_2. We show that for every graph H,

$$H \models \exists C_1 \wedge \exists C_2 \iff H \models \bigvee_{C' \in \mathrm{Merge}(C_1, C_2)} \exists C'.$$

"$\Rightarrow$": Let $g_1 : C_1 \hookrightarrow H$, $g_2 : C_2 \hookrightarrow H$ be total, injective morphisms. Let C' be the graph $g_1(C_1) \cup g_2(C_2)$. By definition, the morphisms $g_1 : C_1 \hookrightarrow C'$, $g_2 : C_2 \hookrightarrow C'$ are total, injective, and together jointly surjective. Moreover, there exists a total, injective morphism $C' \hookrightarrow H$.
"$\Leftarrow$": Let $h : C' \hookrightarrow H$ be a total, injective morphism and $g_1 : C_1 \hookrightarrow C'$, $g_2 : C_2 \hookrightarrow C'$ jointly surjective. The morphisms $h \circ g_1 : C_1 \hookrightarrow H$ and $h \circ g_2 : C_2 \hookrightarrow H$ are total and injective.
Thus, we can transform c into an equivalent positive constraint

$$c' = \bigvee_{C_1 \in \mathcal{B}_1, C_2 \in \mathcal{B}_2} \bigvee_{C' \in \mathrm{Merge}(C_1, C_2)} \exists C'.$$

We show that for all graphs C_1, C_2, the set $\mathrm{Merge}(C_1, C_2)$ of graphs is finite and computable. By joint surjectivity, the number of nodes (edges) of any graph in $\mathrm{Merge}(C_1, C_2)$ is $\leq |V_{C_1}| + |V_{C_2}|$ (edges: $\leq |E_{C_1}| + |E_{C_2}|$). We can test all such graphs for joint surjectivity in the following way: for two graphs $C_1 \in \mathcal{B}_1$, $C_2 \in \mathcal{B}_2$, and a graph C' to be tested, we compute all total, injective morphisms $C_1 \hookrightarrow C'$ and $C_2 \hookrightarrow C'$. We check whether every item of C' has a preimage in C_1 or C_2. The sets $\mathcal{B}_1$ and $\mathcal{B}_2$ are finite. Hence, we can effectively construct the latter positive constraint c'. □

Every positive constraint of the form $\bigvee_{1 \leq i \leq n} \exists C_i$ can be reduced by taking out all redundant graphs C_j for which exists C_i with $C_i \hookrightarrow C_j$.

Reduction to Temporal Logics

Theorem 3.1 (reduction to LTL). For every graph constraint Safe and every natural number $k \geq 0$, there exist LTL constraints $\phi(\mathrm{Safe})$ and $\phi_k(\mathrm{Safe})$ s.t. for every marked joint GTS without deadlocks:

(1) the marked joint GTS is k-step$_\forall$ resilient w.r.t. Safe iff the marked annotated joint GTS satisfies $\phi_k(\mathrm{Safe})$,

[25] See an introduction to logic, e.g., Mendelsohn [Men97, p. 30].

(2) the marked joint GTS is $\mathcal{E}$-step resilient w.r.t. Safe iff
the marked annotated joint GTS satisfies ϕ(Safe).

For better readability, we write $\Rightarrow_{\mathcal{S}'}$ instead of $\Rightarrow_{\mathcal{S}'_A}$ ($\Rightarrow_{\mathcal{E}'}$ instead of $\Rightarrow_{\mathcal{E}'_A}$).

Proof of Theorem 3.1 (1). "$\Rightarrow$": Let $G_0 \Rightarrow G_1 \Rightarrow \ldots$ be an infinite transformation sequence via $(\mathcal{SE})'$. Let $M = G_i$ with $i \geq 0$ and $M \models$ Env, i.e., M was obtained by the application of an environment rule. Consider the subsequence $G_0 \Rightarrow^*_{(\mathcal{SE})'} H \Rightarrow_{\mathcal{E}'} M \Rightarrow^k_{(\mathcal{SE})'} N$. By definition of k-step$_\forall$ resilience, there exists a subsequence $M \Rightarrow^{\leq k}_{(\mathcal{SE})'} N'$ with $N' \models$ Safe. That is, there exists $0 \leq j \leq k$ s.t. $M \models \mathbf{X}^j$Safe, i.e., $M \models \bigvee_{j=0}^k \mathbf{X}^j$Safe. Thus, $G_0 \models \mathbf{G}(\text{Env} \Rightarrow \bigvee_{j=0}^k \mathbf{X}^j\text{Safe})$.
"$\Leftarrow$": Let $G_0 \Rightarrow^*_{(\mathcal{SE})'} H \Rightarrow_{\mathcal{E}} M \Rightarrow^k_{(\mathcal{SE})'} N$ be a transformation sequence via $(\mathcal{SE})'$ obtained from the corresponding transformation sequence via $\mathcal{SE}$. By deadlock-freeness, the latter sequence can be completed to an infinite transformation sequence $G_0 \Rightarrow \ldots$. By assumption, $G_0 \Rightarrow \ldots \models \mathbf{G}(\text{Env} \Rightarrow \bigvee_{j=0}^k \mathbf{X}^j\text{Safe})$. We obtain $M \models \bigvee_{j=0}^k \mathbf{X}^j\text{Safe}$. Thus, there exists a subsequence $M \Rightarrow^{\leq k}_{(\mathcal{SE})'} N'$ with $N' \models$ Safe (and a corresponding subsequence via $\mathcal{SE}$). □

Proof of Theorem 3.1 (2). "$\Rightarrow$": Let $G_0 \Rightarrow G_1 \Rightarrow \ldots$ be an infinite transformation sequence via $(\mathcal{SE})'$. Let $M = G_i$ with $i \geq 0$ and $M \models \text{Env} \wedge \mathbf{X}\text{Sys}$, i.e., M was obtained by the application of an environment rule and a system rule is applied next, yielding a graph N. By definition of $\mathcal{E}$-step resilience, there exists a subsequence $M \Rightarrow^*_{\mathcal{S}'} N'$ with $N' \models$ Safe. That is, $M \models$ Safe or $N \models$ Sys**U**Safe, i.e., $M \models \text{Safe} \vee \mathbf{X}(\text{Sys}\mathbf{U}\text{Safe})$. Thus, $G_0 \models \mathbf{G}((\text{Env} \wedge \mathbf{X}\text{Sys}) \Rightarrow (\text{Safe} \vee \mathbf{X}(\text{Sys}\mathbf{U}\text{Safe})))$.
"$\Leftarrow$": Let $G_0 \Rightarrow^*_{(\mathcal{SE})'} H \Rightarrow_{\mathcal{E}'} M \Rightarrow_{\mathcal{S}'} N \Rightarrow \ldots$ be an infinite transformation sequence via $(\mathcal{SE})'$ obtained from the corresponding transformation sequence via $\mathcal{SE}$. By assumption, $G_0 \Rightarrow \ldots \models \mathbf{G}((\text{Env} \wedge \mathbf{X}\text{Sys}) \Rightarrow (\text{Safe} \vee \mathbf{X}(\text{Sys}\mathbf{U}\text{Safe})))$. We obtain $M \models \text{Safe} \vee \mathbf{X}(\text{Sys}\mathbf{U}\text{Safe})$. Thus, there exists a subsequence $M \Rightarrow^*_{\mathcal{S}'} N'$ with $N' \models$ Safe (and a corresponding subsequence via $\mathcal{S}_A$). □

Theorem 3.2 (reduction to CTL). For every graph constraint Safe and every natural number $k \geq 0$, there exists a CTL constraints φ_k(Safe) s.t. for every marked joint GTS without deadlocks:

the marked joint GTS is k-step$_\exists$ resilient w.r.t. Safe iff
the marked annotated joint GTS satisfies φ_k(Safe).

Proof. "$\Rightarrow$": Let $G_0 \in$ INIT and $G_0 \Rightarrow_{(\mathcal{SE})'} M$ a transformation sequence via $(\mathcal{SE})'$ with $M \models$ Env, i.e., M was obtained by the application of an environment rule. Consider the transformation sequence $G_0 \Rightarrow^*_{(\mathcal{SE})'} H \Rightarrow_{\mathcal{E}'} M$.

By definition of k-step$_\exists$ resilience, there exists a transformation sequence $M \Rightarrow^{\leq k}_{(\mathcal{SE})'} N'$ with $N' \models \text{Safe}$. That is, there exists $0 \leq j \leq k$ s.t. $M \models \mathbf{EX}^j\text{Safe}$, i.e., $M \models \bigvee_{j=0}^{k} \mathbf{EX}^j\text{Safe}$. Thus, $G_0 \models \mathbf{AG}(\text{Env} \Rightarrow \bigvee_{j=0}^{k} \mathbf{EX}^j\text{Safe})$.
"$\Leftarrow$": Let $G_0 \Rightarrow^*_{(\mathcal{SE})'} H \Rightarrow_{\mathcal{E}'} M$ be a transformation sequence via $(\mathcal{SE})'$ obtained from the corresponding transformation sequence via $\mathcal{SE}$. By assumption, $G_0 \models \mathbf{AG}(\text{Env} \Rightarrow \bigvee_{j=0}^{k} \mathbf{EX}^j\text{Safe})$. We obtain $M \models \bigvee_{j=0}^{k} \mathbf{EX}^j\text{Safe}$. Thus, there exists a transformation sequence $M \Rightarrow^{\leq k}_{(\mathcal{SE})'} N'$ with $N' \models \text{Safe}$ (and a corresponding transformation sequence via $\mathcal{SE}$). □

Undecidability

We discuss two undecidability results hinting at the limits of our approach.

In Bertrand et al. [BDK+12], a number of undecidability results are presented. E.g., ideal reachability is undecidable for preserving GTSs. As a consequence, undecidability of the bounded resilience problem for preserving GTSs can be derived.

Proposition 5.11 (undecidability). The bounded resilience problem is undecidable for preserving finite-marked GTS with $\cap$-based graph class in the cases where Safe is positive.

Proof. We consider the bounded resilience problem for preserving finite-marked GTSs, $\text{INIT} = \{G\}$, $\text{BAD} = \exists\varnothing$, $\text{Safe} = \exists H$, and S as the class of all graphs (which is $\cap$-based). We express ideal reachability as bounded resilience:

$$
\begin{aligned}
& \exists k \geq 0 : \ \text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S) && \\
\iff & \exists k \geq 0 : \ \text{post}^*(\text{INIT}) \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S) && ([\![\text{Bad}]\!]_S = S) \\
\iff & \exists k \geq 0 : \uparrow\text{post}^*(\text{INIT}) \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S) && (\text{pre}^{\leq k} \text{ ideal}) \\
\iff & \exists k \geq 0 : \uparrow\text{INIT} \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S) && (\text{preserving}) \\
\iff & \exists k \geq 0 : \ \text{INIT} \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S) && (\text{pre}^{\leq k} \text{ ideal}) \\
\iff & \text{INIT} \subseteq \text{pre}^*([\![\text{Safe}]\!]_S) && (\text{INIT finite}) \\
\iff & \text{INIT} \subseteq \text{pre}^*([\![\exists H]\!]_S) && (\text{Safe} = \exists H) \\
\iff & G \in \text{pre}^*([\![\exists H]\!]_S) && (\text{INIT} = \{G\}) \\
\iff & G \in \text{pre}^*(\uparrow\{H\}) && (S = \text{ all graphs})
\end{aligned}
$$

Thus, ideal reachability in this case is reducible to bounded resilience. By [BDK+12, Prop. 13], ideal reachability is undecidable for preserving GTSs. □

We consider a class of SWSTSs which is not post*-effective – at least not by a generic procedure. We use an undecidability result for reset Petri nets by Dufourd et al. [DFS98].

Proposition B.1. Finite-marked reset Petri are, in general, not post*-effective.

Proof. Assume that finite-marked reset Petri nets are post*-effective (by a generic procedure). By Proposition 4.1, the anti-ideal reachability problem for reset Petri nets is decidable. We consider the anti-ideal J of all markings where no transition can fire (all deadlocks). The basis of the complement of J is given by the set of the "smallest" markings where at least one transition can fire. The anti-ideal is J is not reachable from INIT iff the finite-marked reset Petri net is deadlock-free. Thus, we could decide the deadlock-freeness problem. However, the deadlock-freeness problem is undecidable for finite-marked reset Petri nets, see Akshay et al. [ACD+17, Table 1], reduction from Dufourd et al. [DFS98]. □

Rule-specific Criteria for Joint GTSs

A rule of an annotated joint GTS is neither preserving nor node-bijective. For an annotated joint GTS of $\mathcal{S}$ and $\mathcal{E}$, we consider the properties of the rules of $\mathcal{S}$ and $\mathcal{E}$.

Proposition 5.6. A finite-marked annotated joint GTS$^{\text{bp}}$ of $\mathcal{S}$ and $\mathcal{E}$ is post*-effective if all rules in $\mathcal{S} \cup \mathcal{E}$ are preserving or all rules in $\mathcal{S} \cup \mathcal{E}$ are node-bijective.

Proof. Let $\langle S, (\mathcal{SE})', \text{INIT} \rangle$ be a finite-marked annotated joint GTS$^{\text{bp}}$. If every rule in $\mathcal{S} \cup \mathcal{E}$ is preserving, we can compute the basis of $\uparrow \text{post}^*(\text{INIT})$ by computing the bases of $\uparrow \text{post}^{\leq k}(\text{INIT})$ until every reachable ideal $[\![\exists(\textcircled{q}\textcircled{\varsigma})]\!]_S$ is reached where q is a state of the control automaton and ς is an annotation symbol. By Lemma 2.3, it is decidable whether an ideal $[\![\exists(\textcircled{q}\textcircled{\varsigma})]\!]_S$ is reachable from INIT (choosing the $\cap$-based graph class $\downarrow \text{post}^*(\text{INIT})$).
Let every rule in $\mathcal{S} \cup \mathcal{E}$ be node-bijective. The annotated joint GTS can also be modeled by node-bijective rules encoding the current state of the control automaton and the current annotation symbol as labels of loops at extra nodes. By Theorem 5.3, the finite-marked annotated joint GTS$^{\text{bp}}$ $\langle S, \mathcal{SE}, \text{INIT} \rangle$ is post*-effective. □

Infinite Set of Initial States

We provide the proof of the following decidability result for $\mathcal{I}$-marked SWSTSs.

Theorem 4.3 (bounded resilience for $\mathcal{I}$-marked SWSTSs).
The bounded resilience problem is decidable for

(1) weakly invertible $\mathcal{I}$-marked SWSTSs if BAD $\in \mathcal{J}$, SAFE $\in \mathcal{I}$,

(2) lossy, weakly $\cap$-effective $\mathcal{I}$-marked SWSTSs if BAD, SAFE $\in \mathcal{I}$,

(3) lossy, weakly $\cap$-effectve, $\bot$-bounded $\mathcal{I}$-marked SWSTS$^{\mathcal{B}}$s if BAD $\in \mathcal{I}$, SAFE $\in \mathcal{J}$,

(4) weakly invertible, $\bot$-bounded $\mathcal{I}$-marked SWSTS$^{\mathcal{B}}$s if BAD, SAFE $\in \mathcal{J}$.

Although not used for the proof, the next proposition shows how the requirements for INIT $\in \mathcal{I}$ are related.

Proposition B.2. Lossy ($\bot$-bounded) and weakly $\cap$-effective $\mathcal{I}$-marked WSTS$^{\mathcal{B}}$s are weakly invertible.

Proof. Let $\langle S, \leq, \rightarrow, \text{INIT}\rangle$ be a lossy ($\bot$-bounded) and weakly $\cap$-effective $\mathcal{I}$-marked WSTS$^{\mathcal{B}}$. To compute a basis of $\uparrow\text{post}^*(\text{INIT})$ for an ideal INIT with a given basis, we look at the reachable elements of a basis of the set S of all states. Such a basis element is reachable iff its upward-closure is reachable:

Let $\mathcal{B}$ be a basis of S and INIT $\in \mathcal{I}$. In both cases, a basis of $\uparrow\text{post}^*(\text{INIT})$ is given by the set $\mathcal{B}_{\text{post}}(\text{INIT}) = \{b \in \mathcal{B} : \uparrow\{b\} \text{ is reachable from INIT}\}$. We show that $\uparrow\mathcal{B}_{\text{post}} = \uparrow\text{post}^*(\text{INIT})$:
"$\supseteq$": Let s be any element reachable from INIT. By definition of a basis, there exists a basis element $b \leq s$. It follows that $\uparrow\{b\}$ is reachable from INIT. We showed $\text{post}^*(\text{INIT}) \subseteq \uparrow\mathcal{B}_{\text{post}}(\text{INIT})$ and thus, $\uparrow\text{post}^*(\text{INIT}) \subseteq \uparrow\mathcal{B}_{\text{post}}(\text{INIT})$.
"$\subseteq$": By definition of lossiness and $\bot$-boundedness, if $\uparrow\{b\}$ is reachable from INIT, also b is reachable from INIT. We showed $\mathcal{B}_{\text{post}}(\text{INIT}) \subseteq \text{post}^*(\text{INIT})$ and hence, $\uparrow\mathcal{B}_{\text{post}}(\text{INIT}) \subseteq \uparrow\text{post}^*(\text{INIT})$.
The set $\mathcal{B}_{\text{post}}(\text{INIT})$ can be computed in the following way: By Lemma 2.3, a basis of $\text{pre}^*(\uparrow\{b\})$ is computable. By definition of weak $\cap$-effectiveness, we can decide whether $\text{INIT} \cap \text{pre}^*(\uparrow\{b\})$ is non-empty. The latter is equivalent to the statement that $\uparrow\{b\}$ is reachable from INIT. □

The following lemma states that the inclusion of the intersection $\text{post}^*(\text{INIT}) \cap \text{BAD}$ in an decidable ideal is decidable if we consider weakly invertible in the case BAD $\in \mathcal{J}$ or lossy, weakly $\cap$-effective $\mathcal{I}$-marked WSTSs in the case BAD $\in \mathcal{I}$.

Lemma B.1 (checking inclusion for $\mathcal{I}$-marked WSTSs).
Let $\langle S, \leq, \rightarrow, \mathrm{INIT}\rangle$ be a $\mathcal{I}$-marked WSTS, $\mathrm{BAD} \subseteq S$, and $I \subseteq S$ be a decidable ideal. It is decidable whether $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I$ provided that the $\mathcal{I}$-marked WSTS is

(a) weakly invertible and $\mathrm{BAD} \in \mathcal{J}$,

(b) lossy, weakly $\cap$-effective, and $\mathrm{BAD} \in \mathcal{I}$.

Proof. The idea is to compute a finite representation of the intersections $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD}$ for checking inclusion in the decidable ideal I. To this aim, we use Lemma 4.1:

(a) We consider weakly invertible $\mathcal{I}$-marked WSTSs and $\mathrm{BAD} \in \mathcal{J}$. It holds

$$\begin{aligned} & \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \\ \iff \quad & {\uparrow}\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \text{(Lemma 4.1)} \\ \iff \quad & B_{\mathrm{post}}(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \text{(Lemma 4.1)} \end{aligned}$$

where the set $B_{\mathrm{post}}(\mathrm{INIT})$ is a basis of ${\uparrow}\mathrm{post}^*(\mathrm{INIT})$, i.e., ${\uparrow}B_{\mathrm{post}}(\mathrm{INIT}) = {\uparrow}\mathrm{post}^*(\mathrm{INIT})$. By weak invertibility, $B_{\mathrm{post}}(\mathrm{INIT})$ is computable. By Fact 2.6, $B_{\mathrm{post}}(\mathrm{INIT})$ is finite. The last inclusion is algorithmically checkable. We take out all elements of $B_{\mathrm{post}}(\mathrm{INIT})$ which are not in the decidable anti-ideal BAD and then check inclusion of the remaining elements in the decidable ideal I.

(b) We consider lossy, weakly $\cap$-effective $\mathcal{I}$-marked WSTSs and $\mathrm{BAD} \in \mathcal{I}$. We use the same idea as in the previous case, but we change the roles of $\mathrm{post}^*(\mathrm{INIT})$ and BAD. It holds

$$\begin{aligned} & \mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \\ \iff \quad & {\downarrow}\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq I & \text{(Def. lossy)} \\ \iff \quad & {\downarrow}\mathrm{post}^*(\mathrm{INIT}) \cap \mathcal{B}_{\mathrm{BAD}} \subseteq I & \text{(Lemma 4.1)} \end{aligned}$$

where $\mathcal{B}_{\mathrm{BAD}}$ is a basis of BAD, i.e., ${\uparrow}\,\mathcal{B}_{\mathrm{BAD}} = \mathrm{BAD}$. We show that ${\downarrow}\,\mathrm{post}^*(\mathrm{INIT})$ is a decidable anti-ideal. It holds $s \in {\downarrow}\,\mathrm{post}^*(\mathrm{INIT})$ iff $\mathrm{INIT} \cap \mathrm{pre}^*({\uparrow}\{s\}) \neq \varnothing$ for any $s \in S$. The latter is decidable by Lemma 2.3 and the definition of weak $\cap$-effectiveness. Hence, the last inclusion is decidable. We take out all elements of $\mathcal{B}_{\mathrm{BAD}}$ which are not in ${\downarrow}\mathrm{post}^*(\mathrm{INIT})$ and then check inclusion of the remaining elements in the decidable ideal I. □

Proof of Theorem 4.3. Cases (1) & (2). By Fact 2.8, $\mathrm{pre}^{\leq k}(\mathrm{SAFE})$ is an ideal for every $k \geq 0$ since $\mathrm{SAFE} \in \mathcal{I}$ and, by Definition 2.12, it is decidable. By Lemma B.1, we can decide whether $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^{\leq k}(\mathrm{SAFE})$ for (1) weakly invertible $\mathcal{I}$-marked SWSTSs and (2) lossy, weakly $\cap$-effective $\mathcal{I}$-marked SWSTSs, respectively. By Lemma 2.2, the

sequence SAFE $\subseteq \mathrm{pre}^{\leq 1}(\mathrm{SAFE}) \subseteq \mathrm{pre}^{\leq 2}(\mathrm{SAFE}) \subseteq \ldots$ becomes stationary, i.e., there is a minimal $k_0 \geq 0$ s.t. $\mathrm{pre}^{\leq k_0}(\mathrm{SAFE}) = \mathrm{pre}^*(\mathrm{SAFE})$. By Lemma 2.3, we can also determine this k_0. Thus, we can determine the minimal number $k = k_{\min}$ s.t. $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^{\leq k}(\mathrm{SAFE})$ (if it exists) and also whether it exists. Hence, we can decide the bounded resilience problem. To sum up, the bounded resilience problem is decidable for (1) weakly invertible $\mathcal{I}$-marked SWSTSs and (2) lossy, weakly $\cap$-effective $\mathcal{I}$-marked SWSTSs, respectively.

Cases (3) & (4). By Lemma 4.3, for $\bot$-bounded SWSTSs, there exists a $k \geq 0$ s.t. $\mathrm{pre}^*(\mathrm{SAFE}) = \mathrm{pre}^{\leq k}(\mathrm{SAFE})$. Hence, checking bounded resilience is equivalent to testing inclusion in $\mathrm{pre}^*(\mathrm{SAFE})$. By Proposition 4.3, for $\bot$-bounded SWSTS$^{\mathcal{B}}$s, $\mathrm{pre}^*(\mathrm{SAFE})$ is a decidable ideal since SAFE $\in \mathcal{J}$. By Lemma B.1, we obtain that checking $\mathrm{post}^*(\mathrm{INIT}) \cap \mathrm{BAD} \subseteq \mathrm{pre}^*(\mathrm{SAFE})$ is decidable for (3) lossy, weakly $\cap$-effective, $\bot$-bounded $\mathcal{I}$-marked SWSTS$^{\mathcal{B}}$s and (4) weakly invertible, $\bot$-bounded finite-marked SWSTS$^{\mathcal{B}}$s, respectively. Hence, we can decide the bounded resilience problem for (3) lossy, weakly $\cap$-effective, $\bot$-bounded SWSTS$^{\mathcal{B}}$s and (4) weakly invertible, $\bot$-bounded SWSTS$^{\mathcal{B}}$s, respectively. □

Appendix C

Computations

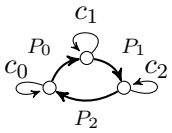
We give more detailed computations for the studied examples. All computations were implemented in basic **Java** code and run on a standard device. Additionally, the examples were verified by hand.

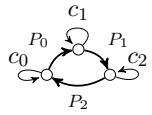
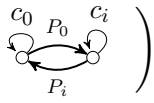
Circular Process Protocol

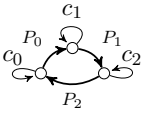
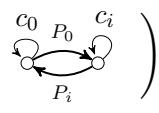
We consider the circular process protocol. The graph class S is depicted in Section 5.3. We write $\mathcal{B}_{\text{post}}$ for the basis of $\uparrow\text{post}^*(\text{INIT})$, $\mathcal{F}_{\text{INIT,Bad}}$ for the finite representation of $\text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S$, $\mathcal{B}_{\text{Safe}}$ for the basis of $[\![\text{Safe}]\!]_S$, and $\mathcal{B}^{\leq k}_{\text{Safe}}$ for the basis of $\text{pre}^{\leq k}([\![\text{Safe}]\!]_S)$ where $k \geq 1$.

Instance 1

GTS: rules in Figure 5.5

INIT: 

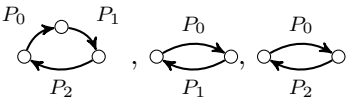
Bad: $\neg\text{AllEnabled} = \neg\exists$  $\wedge \bigwedge_{i=1,2} \neg\exists$ 

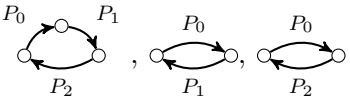
Safe: $\text{AllEnabled} = \exists$  $\vee \bigvee_{i=1,2} \exists$ 

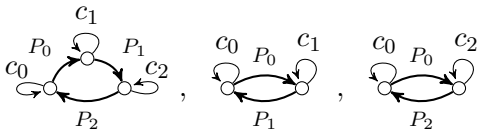
In the first step, we compute the basis $\mathcal{B}_{\text{post}}$ which consists of all basis graphs of S. These graphs are contained in $[\![\text{Bad}]\!]_S$. Hence, the finite representation $\mathcal{F}_{\text{INIT,Bad}}$ consists of all basis graphs of S. We iteratively check whether $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq k}_{\text{Safe}}$. If this is the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{Safe}}$ is

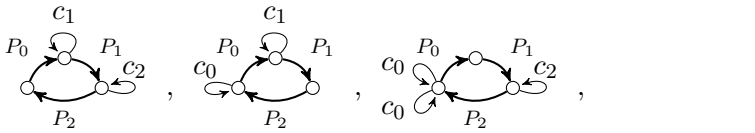
equal to $\mathcal{B}^{\leq k}_{\text{Safe}}$ (up to isomorphism). If this is the case, we terminate (there is no such k), otherwise, we continue.

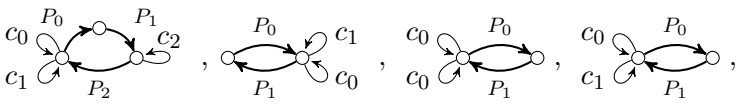
Computation

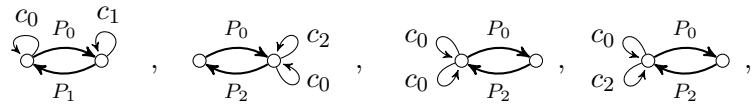
$\mathcal{B}_{\text{post}}$: 

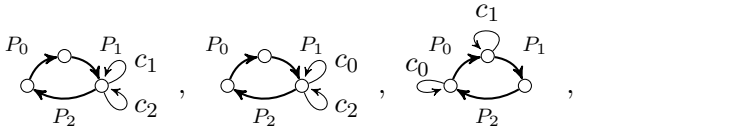
$\mathcal{F}_{\text{INIT,Bad}}$: 

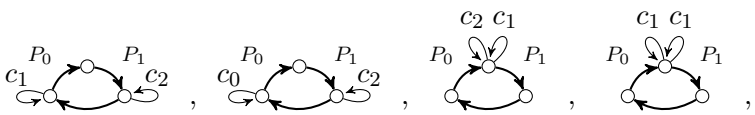
$\mathcal{B}_{\text{Safe}}$: 

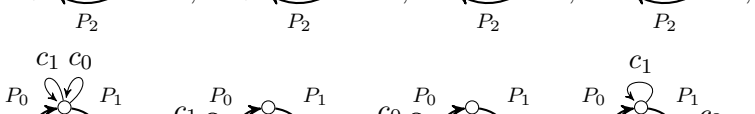
$\mathcal{B}^{\leq 1}_{\text{Safe}}$: 

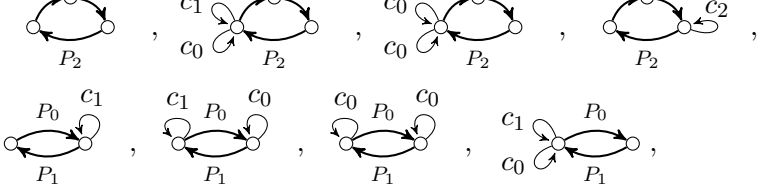


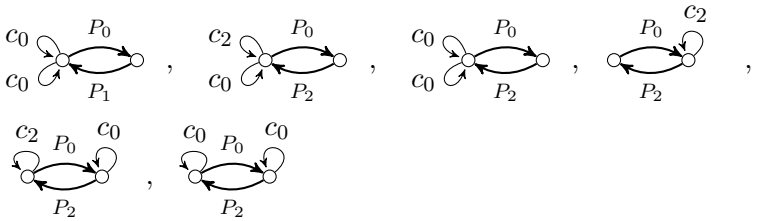


$\mathcal{B}^{\leq 2}_{\text{Safe}}$: 

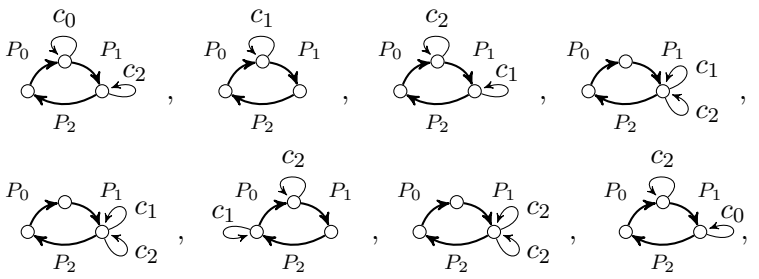




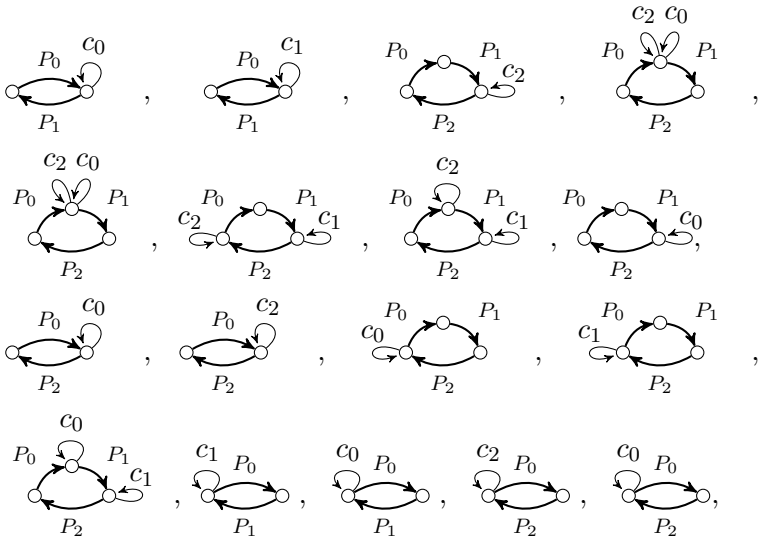


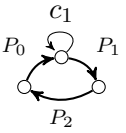


$\mathcal{B}^{\leq 3}_{\text{Safe}}$:

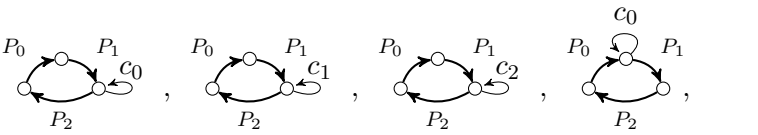


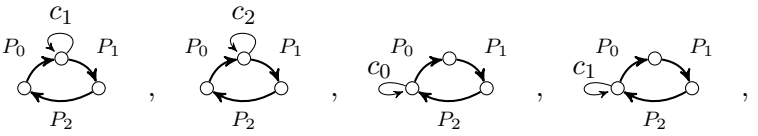
$\mathcal{B}^{\leq 4}_{\text{Safe}}$:

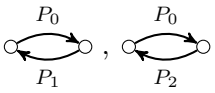




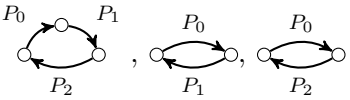
$\mathcal{B}^{\leq 5}_{\text{Safe}}$:







$\mathcal{B}^{\leq 6}_{\text{Safe}}$:



$\Rightarrow$ $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 6}_{\text{Safe}}$

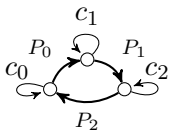
We find that $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 6}_{\text{Safe}}$. Thus, $k_{\min} = 6$.

Conclusion
$\text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq 6}([\![\text{Safe}]\!]_S)$ minimal
$k_{\min}$: 6

Instance 2

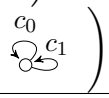
GTS: rules in Figure 5.5

INIT:



Bad: $\text{Command}(c_2) = \exists$

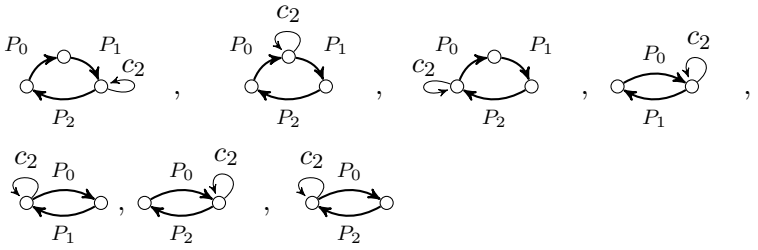
Safe: $\text{Collection}(c_0, c_1) = \exists$



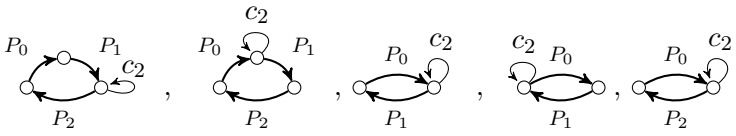
In the first step, we compute the basis $\mathcal{B}_{\text{Bad}}$. The finite representation $\mathcal{F}_{\text{INIT,Bad}}$ consists of all graphs B in $\mathcal{B}_{\text{Bad}}$ s.t. $B \in\downarrow \text{post}^*(\text{INIT})$. We iteratively check whether $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq k}_{\text{Safe}}$. If this is the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{Safe}}$ is equal to $\mathcal{B}^{\leq k}_{\text{Safe}}$ (up to isomorphism). If this is the case, we terminate (there is no such k), otherwise, we continue.

Computation

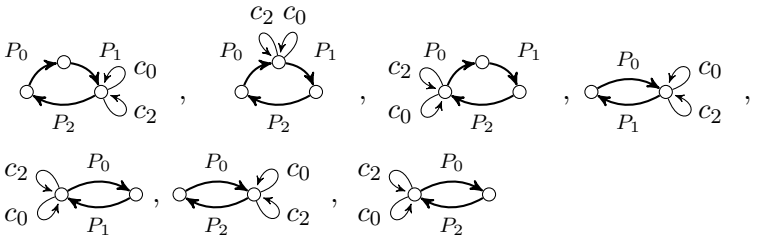
$\mathcal{B}_{\text{Bad}}$:



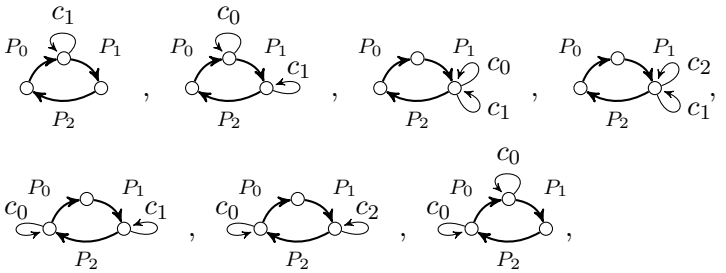
$\mathcal{F}_{\text{INIT,Bad}}$:



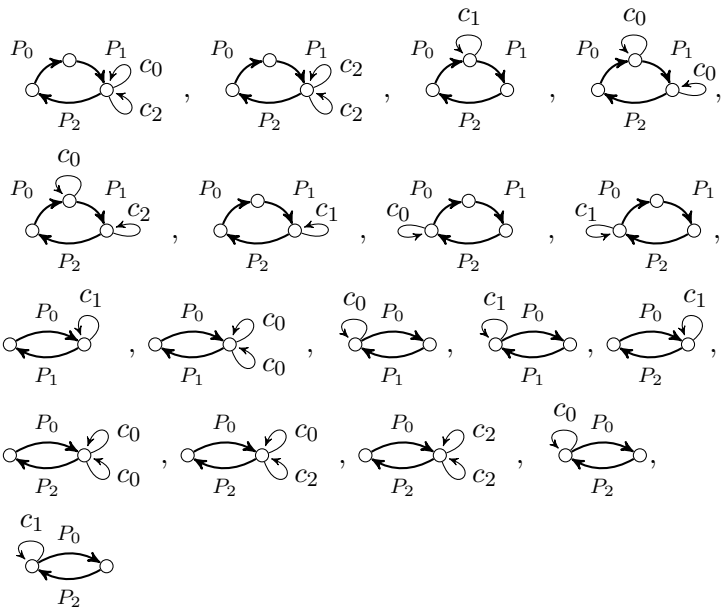
$\mathcal{B}_{\text{Safe}}$:



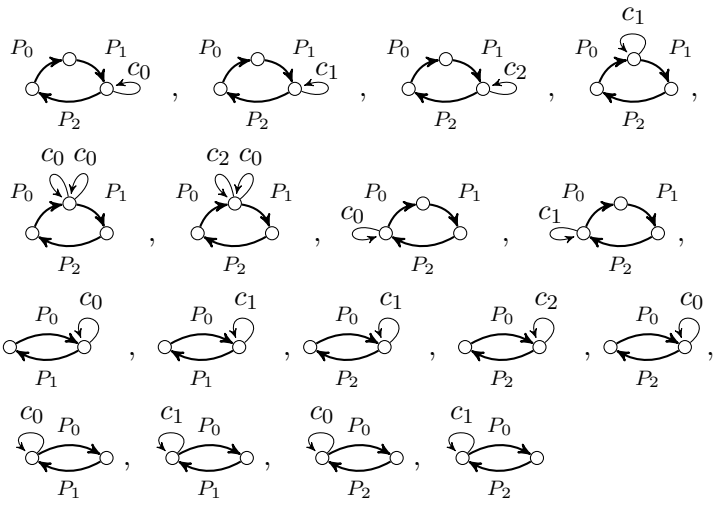
$\mathcal{B}^{\leq 1}_{\text{Safe}}$:



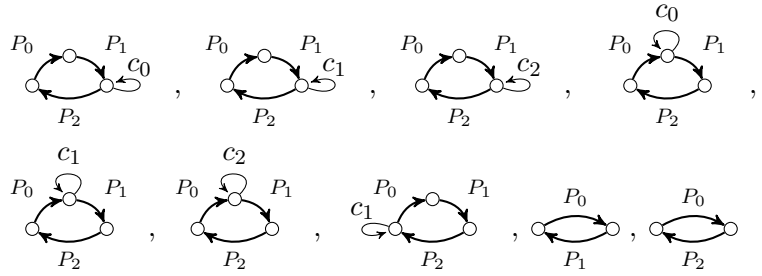
$\mathcal{B}^{\leq 2}_{\text{Safe}}$:



$\mathcal{B}^{\leq 3}_{\text{Safe}}$:



$\mathcal{B}^{\leq 4}_{\text{Safe}}$:



$\Rightarrow$ $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 4}_{\text{Safe}}$

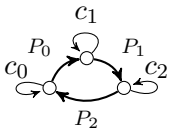
We find that $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 4}_{\text{Safe}}$. Thus, $k_{\min} = 4$.

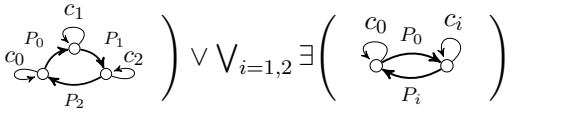
Conclusion
$\text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq 4}([\![\text{Safe}]\!]_S)$ minimal
$k_{\min}$: 4

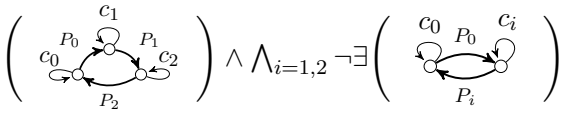
We consider the third instance to illustrate the decidability algorithm for the case that Bad is negative.

Instance 3

GTS: rules in Figure 5.5

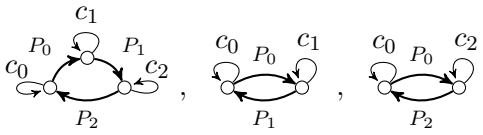
INIT: 

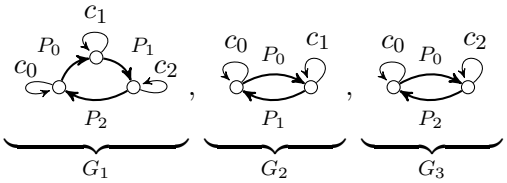
Bad: AllEnabled=$\exists$ 

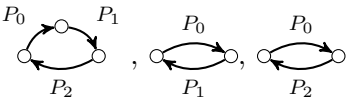
Safe: $\neg$AllEnabled=$\neg\exists$ 

In the first step, we compute the basis $\mathcal{B}_{\text{Bad}}$. The finite representation $\mathcal{F}_{\text{INIT,Bad}}$ consists of all graphs B in $\mathcal{B}_{\text{Bad}}$ s.t. $B \in \downarrow \text{post}^*(\text{INIT})$. For a graph G, let $\mathcal{B}_{\text{post}}(G)$ be the basis of $\uparrow \text{post}^*(G)$. For every $G \in \mathcal{F}_{\text{INIT,Bad}}$, we check whether $\mathcal{B}_{\text{post}}(G) \cap [\![\text{Safe}]\!]_S \neq \varnothing$. If this is the case for every $G \in \mathcal{F}_{\text{INIT,Bad}}$, there exists a bound k, otherwise it does not exist.

Computation

$\mathcal{B}_{\text{Bad}}$: 

$\mathcal{F}_{\text{INIT,Bad}}$: 

$\mathcal{B}_{\text{post}}(G_1)$: 

$\Rightarrow$ $\mathcal{B}_{\text{post}}(G_1) \cap [\![\text{Safe}]\!]_S \neq \varnothing$

$\mathcal{B}_{\text{post}}(G_2)$: 

$\Rightarrow$ $\mathcal{B}_{\text{post}}(G_2) \cap [\![\text{Safe}]\!]_S \neq \varnothing$

$\mathcal{B}_{\text{post}}(G_3)$: 

$\Rightarrow$ $\mathcal{B}_{\text{post}}(G_3) \cap [\![\text{Safe}]\!]_S \neq \varnothing$

We conclude that there exists a bound k.

Conclusion

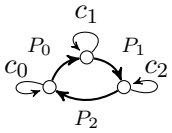
$\exists k \geq 0:$ $\quad \text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S)$

Note that we can infer that $k_{\min} = 1$: By applying a `Clear`-rule to a graph in which all processes are enabled, we can reach a graph in which at least one process is not enabled in one step. Thus, $k_{\min} \leq 1$. Since ¬AllEnabled is the negation of AllEnabled, $k_{\min} \neq 0$.

For the fourth instance, we use that the negative constraint ¬3Processes is equivalent to the positive constraint 2Processes.

Instance 4

GTS: rules in Figure 5.5

INIT: 

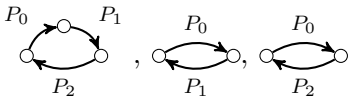
Bad: NoCommand $= \bigwedge_{i=0,1,2} \neg\exists$ 

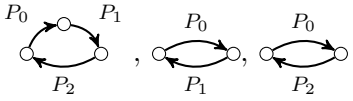
Safe: 2Processes $= \exists$ 

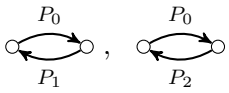
$\vee \exists$ 

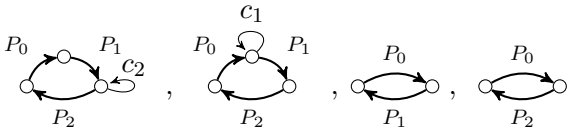
In the first step, we compute the basis $\mathcal{B}_{\text{post}}$ which consists of all basis graphs of S. These graphs are contained in $[\![\text{Bad}]\!]_S$. Hence, the finite representation $\mathcal{F}_{\text{INIT,Bad}}$ consists of all basis graphs of S. We iteratively check whether $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq k}_{\text{Safe}}$. If this is the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{Safe}}$ is equal to $\mathcal{B}^{\leq k}_{\text{Safe}}$ (up to isomorphism). If this is the case, we terminate (there is no such k), otherwise, we continue.

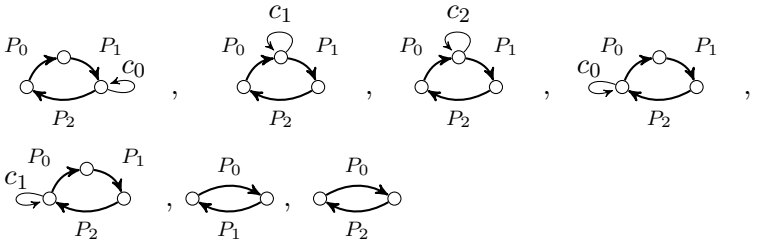
Computation

$\mathcal{B}_{\text{post}}$: 

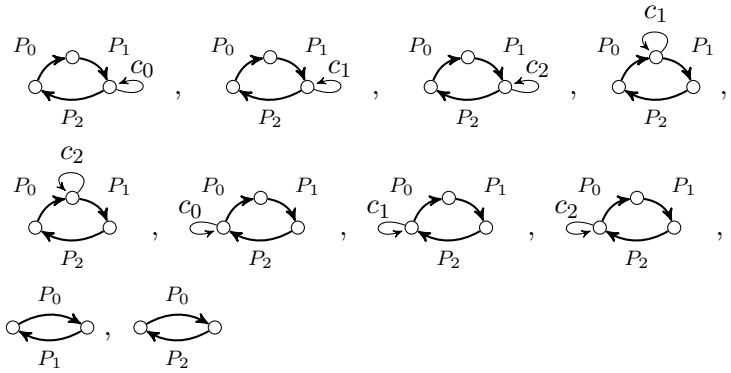
$\mathcal{F}_{\text{INIT,Bad}}$: 

$\mathcal{B}_{\text{Safe}}$: 

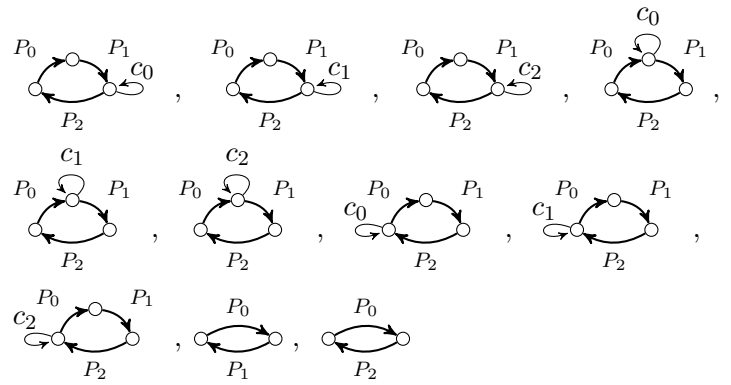
$\mathcal{B}^{\leq 1}_{\text{Safe}}$: 

$\mathcal{B}^{\leq 2}_{\text{Safe}}$: 

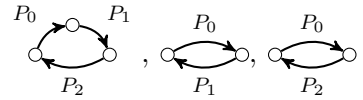
$\mathcal{B}^{\leq 3}_{\text{Safe}}$:



$\mathcal{B}^{\leq 4}_{\text{Safe}}$:



$\mathcal{B}^{\leq 5}_{\text{Safe}}$:



$\Rightarrow$ $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 5}_{\text{Safe}}$

We find that $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 5}_{\text{Safe}}$. Thus, $k_{\min} = 5$.

Conclusion

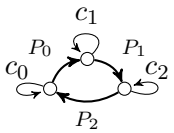
$\text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq 5}([\![\text{Safe}]\!]_S)$ minimal

$k_{\min}$: 5

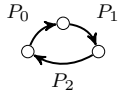
Instance 5

GTS: rules in Figure 5.5

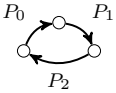
INIT:



Bad: $\neg 3\text{Processes} = \neg\exists$

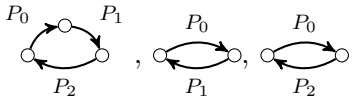


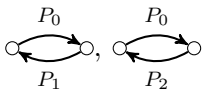
Safe: $3\text{Processes} = \exists$

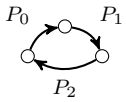


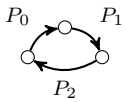
In the first step, we compute the basis $\mathcal{B}_{\text{post}}$ which consists of all basis graphs of S. The finite representation $\mathcal{F}_{\text{INIT,Bad}}$ consists of the two basis graphs with two processes. We iteratively check whether $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq k}_{\text{Safe}}$. If this is the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{Safe}}$ is equal to $\mathcal{B}^{\leq k}_{\text{Safe}}$ (up to isomorphism). If this is the case, we terminate (there is no such k), otherwise, we continue.

Computation

$\mathcal{B}_{\text{post}}$: 

$\mathcal{F}_{\text{INIT,Bad}}$: 

$\mathcal{B}_{\text{Safe}}$: 

$\mathcal{B}^{\leq 1}_{\text{Safe}}$: 

$\Rightarrow$ $\mathcal{B}^{\leq 1}_{\text{Safe}} = \mathcal{B}_{\text{Safe}}$

We conclude that there does not exist a bound k.

Conclusion

$\nexists k \geq 0$: $\text{post}^*(\text{INIT}) \cap [\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq k}([\![\text{Safe}]\!]_S)$

Supply Chain

We consider the supply chain as Petri net in Example 4.3. The markings are represented by tuples. We write $\mathcal{B}_{\text{post}}$ for the basis of $\uparrow \text{post}^*(\text{INIT})$, $\mathcal{F}_{\text{INIT,BAD}}$ for the finite representation of $\text{post}^*(\text{INIT}) \cap \text{BAD}$, $\mathcal{B}_{\text{SAFE}}$ for the basis of SAFE, and $\mathcal{B}^{\leq k}_{\text{SAFE}}$ for the basis of $\text{pre}^{\leq k}(\text{SAFE})$ where $k \geq 1$.

Given

WSTS:	Petri net in Figure 4.4
INIT:	$\langle 0, 1, 1, 1 \rangle$
BAD:	M marking : $M(\mathit{warehouse}) = 0$ or $M(\mathit{store}_i) = 0$ for some i
SAFE:	M marking : $M(\mathit{warehouse}), M(\mathit{store}_1), M(\mathit{store}_2) \geq 1$

In the first step, we compute the basis $\mathcal{B}_{\text{post}}$ which consists only of the zero-marking. Hence, the finite respresentation $\mathcal{F}_{\text{INIT,BAD}}$ consists also of the zero-marking. We iteratively check whether $\mathcal{F}_{\text{INIT,BAD}} \subseteq\uparrow \mathcal{B}^{\leq k}_{\text{SAFE}}$. If this is

the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{SAFE}}$ is equal to $\mathcal{B}^{\leq k}_{\text{SAFE}}$. If this is the case, we terminate (there is no such k), otherwise, we continue.

Computation	
$\mathcal{B}_{\text{post}}$:	$\langle 0,0,0,0\rangle$
$\mathcal{F}_{\text{INIT,BAD}}$:	$\langle 0,0,0,0\rangle$
$\mathcal{B}_{\text{SAFE}}$:	$\langle 0,1,1,1\rangle$
$\mathcal{B}^{\leq 1}_{\text{SAFE}}$:	$\langle 0,1,1,1\rangle, \langle 0,2,0,1\rangle, \langle 0,2,1,0\rangle, \langle 1,0,1,1\rangle$
$\mathcal{B}^{\leq 2}_{\text{SAFE}}$:	$\langle 0,0,1,1\rangle, \langle 0,2,0,1\rangle, \langle 0,2,1,0\rangle, \langle 0,3,0,0\rangle$
$\mathcal{B}^{\leq 3}_{\text{SAFE}}$:	$\langle 0,0,1,1\rangle, \langle 0,2,0,1\rangle, \langle 0,2,1,0\rangle, \langle 0,3,0,0\rangle, \langle 1,1,0,1\rangle, \langle 1,1,1,0\rangle, \langle 1,2,0,0\rangle$
$\mathcal{B}^{\leq 4}_{\text{SAFE}}$:	$\langle 0,0,1,1\rangle, \langle 0,1,0,1\rangle, \langle 0,1,1,0\rangle, \langle 0,2,0,0\rangle, \langle 2,0,0,1\rangle, \langle 2,0,1,0\rangle, \langle 2,1,0,0\rangle$
$\mathcal{B}^{\leq 5}_{\text{SAFE}}$:	$\langle 0,0,1,1\rangle, \langle 0,1,0,1\rangle, \langle 0,1,1,0\rangle, \langle 0,2,0,0\rangle, \langle 1,0,0,1\rangle, \langle 1,0,1,0\rangle, \langle 1,1,0,0\rangle, \langle 3,0,0,0\rangle$
$\mathcal{B}^{\leq 6}_{\text{SAFE}}$:	$\langle 0,0,0,1\rangle, \langle 0,0,1,0\rangle, \langle 0,1,0,0\rangle, \langle 2,0,0,0\rangle$
$\mathcal{B}^{\leq 7}_{\text{SAFE}}$:	$\langle 0,0,0,1\rangle, \langle 0,0,1,0\rangle, \langle 0,1,0,0\rangle, \langle 1,0,0,0\rangle$
$\mathcal{B}^{\leq 8}_{\text{SAFE}}$:	$\langle 0,0,0,0\rangle$

We find that $\mathcal{F}_{\text{INIT,Bad}} \subseteq \uparrow \mathcal{B}^{\leq 8}_{\text{Safe}}$. Thus, $k_{\min} = 8$.

Conclusion
$\text{post}^*(\text{INIT}) \cap \text{BAD} \subseteq \text{pre}^{\leq 8}(\text{SAFE})$ minimal
$k_{\min}$: 8

Supply Chain II

We consider the supply chain II as joint GTS in Example 5.5. However, for the graphs, we use the representations as markings with an additional state of the control automaton. The control states are numbered along the paths in the automaton in Figure 5.4 (upper path: q_1, q_2, q_3, lower path: q_5, q_6, q_7). Since the initial graph satisfies Safe, we assume that an environment was applied iff the current state of the control automaton is q_0. We write $\mathcal{B}_{\text{post}}$ for the basis of $\uparrow \text{post}^*(\text{INIT})$, BAD for $[\![\text{Bad}]\!]_S$, $\mathcal{F}_{\text{INIT,BAD}}$ for the finite representation of $\text{post}^*(\text{INIT}) \cap \text{BAD}$, $\mathcal{B}_{\text{SAFE}}$ for the basis of $\text{SAFE} = [\![\text{Safe}]\!]_S$, and $\mathcal{B}^{\leq k}_{\text{SAFE}}$ for the basis of $\text{pre}^{\leq k}(\text{SAFE})$ where $k \geq 1$.

Given	
GTS:	joint GTS constructed from Figure 5.4
INIT:	$\langle 0,1,1,1,q_0\rangle$
BAD:	$\langle M, q_0\rangle : M$ marking
SAFE:	$\langle M, q\rangle : M$ marking, $M(\mathit{warehouse}), M(\mathit{store}_1), M(\mathit{store}_2) \geq 1$

In this case, we directly compute the finite representation $\mathcal{F}_{\text{INIT,BAD}}$ using that this example constitutes a Petri net.[26] We iteratively check whether $\mathcal{F}_{\text{INIT,BAD}} \subseteq \uparrow \mathcal{B}^{\leq k}_{\text{SAFE}}$. If this is the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{SAFE}}$ is equal to $\mathcal{B}^{\leq k}_{\text{SAFE}}$. If this is the case, we terminate (there is no such k), otherwise, we continue.

Computation

$\mathcal{F}_{\text{INIT,BAD}}$: $\langle 0,1,1,1,q_0\rangle, \langle 0,5,0,0,q_0\rangle, \langle 0,0,3,0,q_0\rangle, \langle 0,0,0,3,q_0\rangle,$
$\langle 0,1,2,0,q_0\rangle, \langle 0,1,0,2,q_0\rangle, \langle 0,0,2,1,q_0\rangle, \langle 0,0,1,2,q_0\rangle,$
$\langle 0,3,1,0,q_0\rangle, \langle 0,3,0,1,q_0\rangle$

$\mathcal{B}_{\text{SAFE}}$: $\langle 0,1,1,1,q_i\rangle$ for $0 \leq i \leq 7$

$\mathcal{B}^{\leq 1}_{\text{SAFE}}$: $\langle 0,1,1,1,q_0\rangle, \langle 0,2,0,1,q_0\rangle, \langle 0,2,1,0,q_0\rangle, \langle 0,1,1,1,q_1\rangle,$
$\langle 1,0,1,1,q_1\rangle, \langle 0,1,1,1,q_2\rangle, \langle 0,2,0,1,q_2\rangle, \langle 0,2,1,0,q_2\rangle,$
$\langle 0,1,1,1,q_3\rangle, \langle 1,0,1,1,q_3\rangle, \langle 0,1,1,1,q_4\rangle, \langle 0,1,1,1,q_5\rangle,$
$\langle 0,2,0,1,q_5\rangle, \langle 0,2,1,0,q_5\rangle, \langle 0,1,1,1,q_6\rangle, \langle 1,0,1,1,q_6\rangle,$
$\langle 0,1,1,1,q_7\rangle$

$\mathcal{B}^{\leq 2}_{\text{SAFE}}$: $\langle 0,0,1,1,q_0\rangle, \langle 0,2,0,1,q_0\rangle, \langle 0,2,1,0,q_0\rangle, \langle 0,3,0,0,q_0\rangle,$
$\langle 0,1,1,1,q_1\rangle, \langle 1,0,1,1,q_1\rangle, \langle 1,0,1,1,q_1\rangle, \langle 1,1,0,1,q_1\rangle,$
$\langle 1,1,1,0,q_1\rangle, \langle 0,0,1,1,q_2\rangle, \langle 0,2,1,0,q_2\rangle, \langle 0,2,0,1,q_2\rangle,$
$\langle 0,1,1,1,q_3\rangle, \langle 1,0,1,1,q_3\rangle, \langle 0,1,1,1,q_4\rangle, \langle 0,2,0,2,q_4\rangle,$
$\langle 0,2,2,0,q_4\rangle, \langle 0,3,0,1,q_4\rangle, \langle 0,3,1,0,q_4\rangle, \langle 0,1,1,1,q_5\rangle,$
$\langle 0,2,0,1,q_5\rangle, \langle 0,2,1,0,q_5\rangle, \langle 0,1,1,1,q_6\rangle, \langle 1,0,1,1,q_6\rangle,$
$\langle 0,1,1,1,q_7\rangle, \langle 0,2,0,2,q_7\rangle, \langle 0,2,2,0,q_7\rangle$

$\mathcal{B}^{\leq 3}_{\text{SAFE}}$: $\langle 0,0,1,1,q_0\rangle, \langle 0,1,0,1,q_0\rangle, \langle 0,1,1,0,q_0\rangle, \langle 0,3,0,0,q_0\rangle,$
$\langle 1,0,1,1,q_1\rangle, \langle 1,1,0,1,q_1\rangle, \langle 0,1,1,1,q_1\rangle, \langle 1,1,1,0,q_1\rangle,$
$\langle 0,0,1,1,q_2\rangle, \langle 0,2,0,1,q_2\rangle, \langle 0,2,1,0,q_2\rangle, \langle 1,2,0,1,q_3\rangle,$
$\langle 1,2,1,0,q_3\rangle, \langle 1,1,2,0,q_3\rangle, \langle 1,1,0,2,q_3\rangle, \langle 1,0,1,1,q_3\rangle,$
$\langle 0,1,1,1,q_3\rangle, \langle 0,1,1,1,q_4\rangle, \langle 0,3,0,1,q_4\rangle, \langle 0,4,0,0,q_4\rangle,$
$\langle 0,3,1,0,q_4\rangle, \langle 0,0,2,1,q_4\rangle, \langle 0,2,2,0,q_4\rangle, \langle 0,0,1,2,q_4\rangle,$
$\langle 0,2,0,2,q_4\rangle, \langle 0,0,1,1,q_5\rangle, \langle 0,2,0,1,q_5\rangle, \langle 0,2,1,0,q_5\rangle,$
$\langle 1,1,2,0,q_6\rangle, \langle 1,1,0,2,q_6\rangle, \langle 1,0,1,1,q_6\rangle, \langle 0,1,1,1,q_6\rangle,$
$\langle 0,0,2,1,q_7\rangle, \langle 0,3,1,0,q_7\rangle, \langle 0,2,2,0,q_7\rangle, \langle 0,3,0,1,q_7\rangle,$
$\langle 0,0,1,2,q_7\rangle, \langle 0,2,0,2,q_7\rangle, \langle 0,1,1,1,q_7\rangle$

$\mathcal{B}^{\leq 4}_{\text{SAFE}}$: $\langle 0,1,0,1,q_0\rangle, \langle 0,1,1,0,q_0\rangle, \langle 0,0,1,1,q_0\rangle, \langle 0,3,0,0,q_0\rangle,$
$\langle 1,1,1,0,q_1\rangle, \langle 1,0,1,1,q_1\rangle, \langle 1,1,0,1,q_1\rangle, \langle 0,1,1,1,q_1\rangle,$
$\langle 0,2,0,1,q_2\rangle, \langle 0,4,0,0,q_2\rangle, \langle 0,2,1,0,q_2\rangle, \langle 0,1,2,0,q_2\rangle,$
$\langle 0,1,0,2,q_2\rangle, \langle 0,0,1,1,q_2\rangle, \langle 1,0,1,1,q_3\rangle, \langle 1,2,0,1,q_3\rangle,$
$\langle 1,3,0,0,q_3\rangle, \langle 1,2,1,0,q_3\rangle, \langle 1,1,2,0,q_3\rangle, \langle 1,1,0,2,q_3\rangle,$
$\langle 0,1,1,1,q_3\rangle, \langle 0,1,1,1,q_4\rangle, \langle 0,2,0,1,q_4\rangle, \langle 0,2,1,0,q_4\rangle,$
$\langle 0,4,0,0,q_4\rangle, \langle 0,0,2,1,q_4\rangle, \langle 0,1,2,0,q_4\rangle, \langle 0,0,1,2,q_4\rangle,$

[26] We computed the basis of $\uparrow \text{post}^{\leq 27}(\text{INIT})$ and proved that it contains all basis elements of $\uparrow \text{post}^{*}(\text{INIT})$ with the control state q_0.

	$\langle 0,1,0,2,q_4\rangle, \langle 0,2,1,0,q_5\rangle, \langle 0,4,0,0,q_5\rangle, \langle 0,2,0,1,q_5\rangle,$
	$\langle 0,1,2,0,q_5\rangle, \langle 0,1,0,2,q_5\rangle, \langle 0,0,1,1,q_5\rangle, \langle 1,2,1,0,q_6\rangle,$
	$\langle 1,1,2,0,q_6\rangle, \langle 1,1,0,2,q_6\rangle, \langle 1,2,0,1,q_6\rangle, \langle 1,0,1,1,q_6\rangle,$
	$\langle 0,1,1,1,q_6\rangle, \langle 0,0,2,1,q_7\rangle, \langle 0,1,1,1,q_7\rangle, \langle 0,1,2,0,q_7\rangle,$
	$\langle 0,3,1,0,q_7\rangle, \langle 0,3,0,1,q_7\rangle, \langle 0,0,1,2,q_7\rangle, \langle 0,1,0,2,q_7\rangle$
$\mathcal{B}^{\leq 5}_{\text{SAFE}}$:	$\langle 0,1,0,1,q_0\rangle, \langle 0,1,1,0,q_0\rangle, \langle 0,0,1,1,q_0\rangle, \langle 0,3,0,0,q_0\rangle,$
	$\langle 1,1,0,1,q_1\rangle, \langle 1,1,1,0,q_1\rangle, \langle 1,0,2,0,q_1\rangle, \langle 1,0,0,2,q_1\rangle,$
	$\langle 0,1,1,1,q_1\rangle, \langle 1,3,0,0,q_1\rangle, \langle 1,0,1,1,q_1\rangle, \langle 0,0,1,1,q_2\rangle,$
	$\langle 0,2,0,1,q_2\rangle, \langle 0,3,0,0,q_2\rangle, \langle 0,2,1,0,q_2\rangle, \langle 0,1,2,0,q_2\rangle,$
	$\langle 0,1,0,2,q_2\rangle, \langle 1,0,1,1,q_3\rangle, \langle 1,1,0,1,q_3\rangle, \langle 1,1,1,0,q_3\rangle,$
	$\langle 1,3,0,0,q_3\rangle, \langle 1,0,2,0,q_3\rangle, \langle 1,0,0,2,q_3\rangle, \langle 0,1,1,1,q_3\rangle,$
	$\langle 0,2,0,1,q_4\rangle, \langle 0,2,1,0,q_4\rangle, \langle 0,1,1,1,q_4\rangle, \langle 0,4,0,0,q_4\rangle,$
	$\langle 0,1,2,0,q_4\rangle, \langle 0,0,2,1,q_4\rangle, \langle 0,1,0,2,q_4\rangle, \langle 0,0,1,2,q_4\rangle,$
	$\langle 0,4,0,0,q_5\rangle, \langle 0,2,1,0,q_5\rangle, \langle 0,1,2,0,q_5\rangle, \langle 0,1,0,2,q_5\rangle,$
	$\langle 0,2,0,1,q_5\rangle, \langle 0,0,1,1,q_5\rangle, \langle 1,0,1,1,q_6\rangle, \langle 1,0,2,0,q_6\rangle,$
	$\langle 1,2,1,0,q_6\rangle, \langle 1,0,0,2,q_6\rangle, \langle 1,2,0,1,q_6\rangle, \langle 0,1,1,1,q_6\rangle,$
	$\langle 0,0,2,1,q_7\rangle, \langle 0,3,1,0,q_7\rangle, \langle 0,1,1,1,q_7\rangle, \langle 0,1,2,0,q_7\rangle,$
	$\langle 0,1,0,2,q_7\rangle, \langle 0,0,1,2,q_7\rangle, \langle 0,3,0,1,q_7\rangle$
$\mathcal{B}^{\leq 6}_{\text{SAFE}}$:	$\langle 0,1,0,1,q_0\rangle, \langle 0,1,1,0,q_0\rangle, \langle 0,0,2,0,q_0\rangle, \langle 0,0,0,2,q_0\rangle,$
	$\langle 0,3,0,0,q_0\rangle, \langle 0,0,1,1,q_0\rangle, \langle 1,1,0,1,q_1\rangle, \langle 1,2,0,0,q_1\rangle,$
	$\langle 1,1,1,0,q_1\rangle, \langle 1,0,2,0,q_1\rangle, \langle 1,0,0,2,q_1\rangle, \langle 1,0,1,1,q_1\rangle,$
	$\langle 0,1,1,1,q_1\rangle, \langle 0,0,1,1,q_2\rangle, \langle 0,1,0,1,q_2\rangle, \langle 0,1,1,0,q_2\rangle,$
	$\langle 0,3,0,0,q_2\rangle, \langle 0,0,2,0,q_2\rangle, \langle 0,0,0,2,q_2\rangle, \langle 1,1,0,1,q_3\rangle,$
	$\langle 1,1,1,0,q_3\rangle, \langle 1,0,1,1,q_3\rangle, \langle 1,3,0,0,q_3\rangle, \langle 1,0,2,0,q_3\rangle,$
	$\langle 1,0,0,2,q_3\rangle, \langle 0,1,1,1,q_3\rangle, \langle 0,2,0,1,q_4\rangle, \langle 0,2,1,0,q_4\rangle,$
	$\langle 0,1,1,1,q_4\rangle, \langle 0,4,0,0,q_4\rangle, \langle 0,1,2,0,q_4\rangle, \langle 0,0,2,1,q_4\rangle,$
	$\langle 0,1,0,2,q_4\rangle, \langle 0,0,1,2,q_4\rangle, \langle 0,0,1,1,q_5\rangle, \langle 0,0,2,0,q_5\rangle,$
	$\langle 0,2,1,0,q_5\rangle, \langle 0,0,0,2,q_5\rangle, \langle 0,2,0,1,q_5\rangle, \langle 0,4,0,0,q_5\rangle,$
	$\langle 1,0,1,1,q_6\rangle, \langle 1,0,2,0,q_6\rangle, \langle 1,2,1,0,q_6\rangle, \langle 1,0,0,2,q_6\rangle,$
	$\langle 1,2,0,1,q_6\rangle, \langle 0,1,1,1,q_6\rangle\langle 0,1,0,2,q_7\rangle, \langle 0,1,1,1,q_7\rangle,$
	$\langle 0,1,2,0,q_7\rangle, \langle 0,0,2,1,q_7\rangle, \langle 0,3,1,0,q_7\rangle, \langle 0,0,1,2,q_7\rangle,$
	$\langle 0,3,0,1,q_7\rangle$
$\Rightarrow$	$\mathcal{F}_{\text{INIT,BAD}} \subseteq\uparrow \mathcal{B}^{\leq 6}_{\text{SAFE}}$.

We find that $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 13}_{\text{SAFE}}$. Thus, $k_{\min} = 6$.

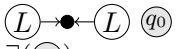
Conclusion
post*(INIT)$\cap$BAD $\subseteq$ pre$^{\leq 6}$(SAFE) minimal
$k_{\min}$: 6

Path Game

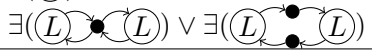
We consider the path game as joint GTS. The graph class S is depicted in Example 5.4. We use the same notation as in the computations for the circular process protocol. We assume that Bad is equivalent to the existence of a q_0-labeled node. (The following computation of the basis $\mathcal{B}_{\text{post}}$ implies that including the initial graph has no impact on the result.)

Given

GTS: joint GTS constructed from Example 5.4

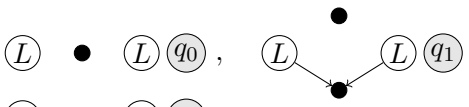
INIT: 

Bad: 

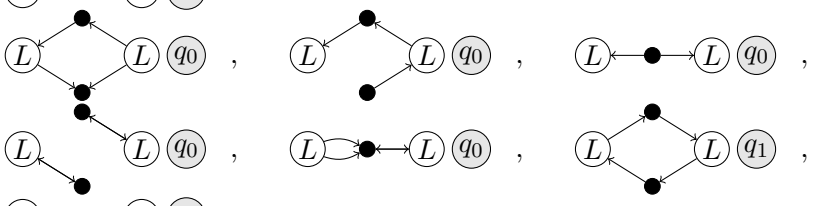
Safe: 

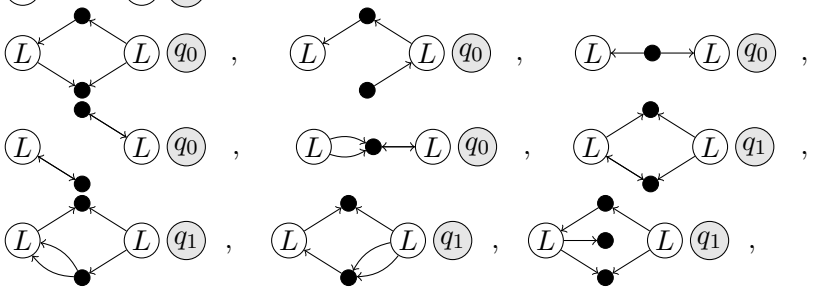
In the first step, we compute the basis $\mathcal{B}_{\text{post}}$ using the fact that we can reach a graph without edges when the system is only applying `Rev`-rules. The finite representation consists of the latter graph with the control state q_0. We iteratively check whether $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq k}_{\text{Safe}}$. If this is the case, we terminate (the current k is $k_{\min}$). Otherwise, we check whether the next basis $\mathcal{B}^{\leq k+1}_{\text{Safe}}$ is equal to $\mathcal{B}^{\leq k}_{\text{Safe}}$ (up to isomorphism). If this is the case, we terminate (there is no such k), otherwise, we continue.

Computation

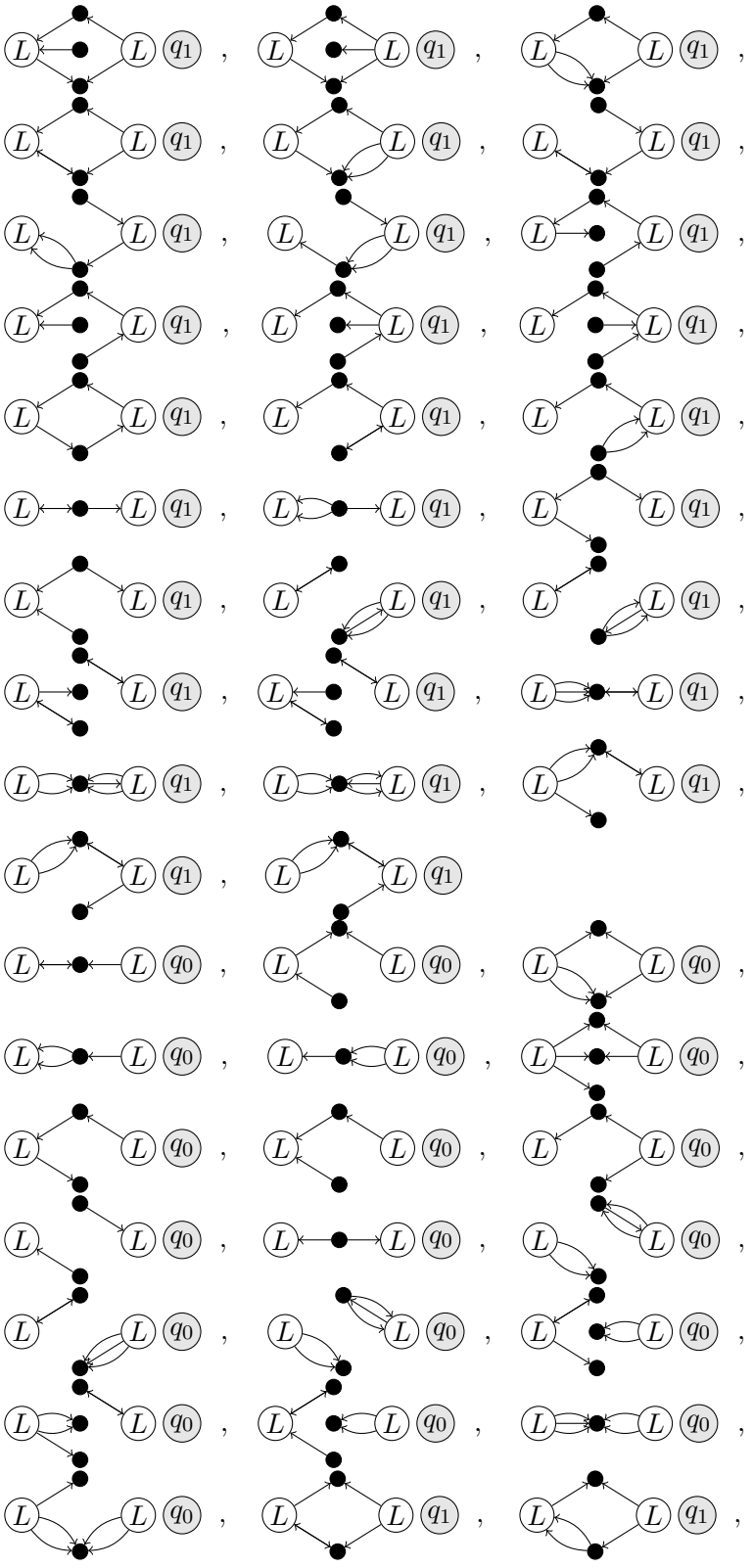
$\mathcal{B}_{\text{post}}$: 

$\mathcal{F}_{\text{INIT,Bad}}$: 

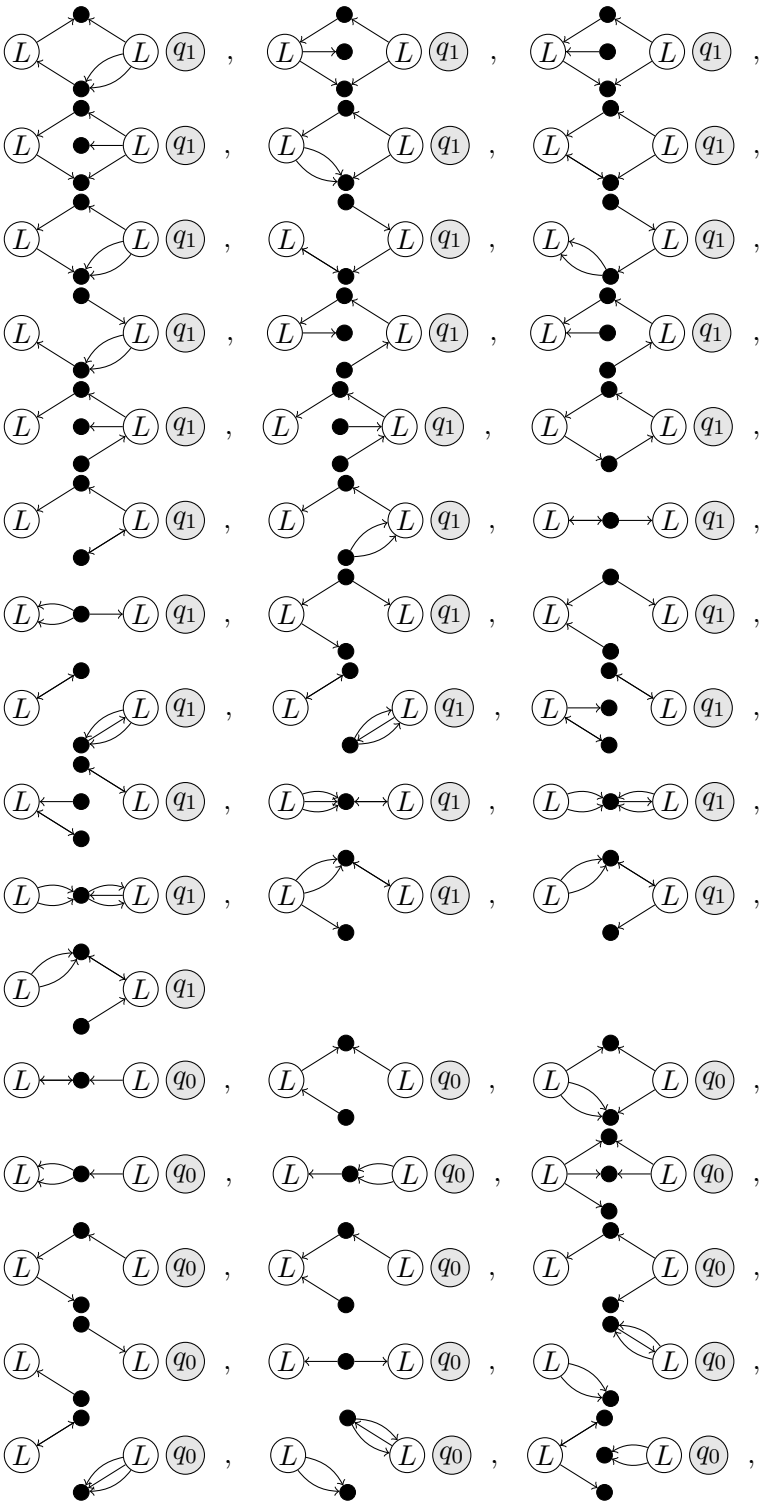
$\mathcal{B}^{\leq 1}_{\text{Safe}}$: 

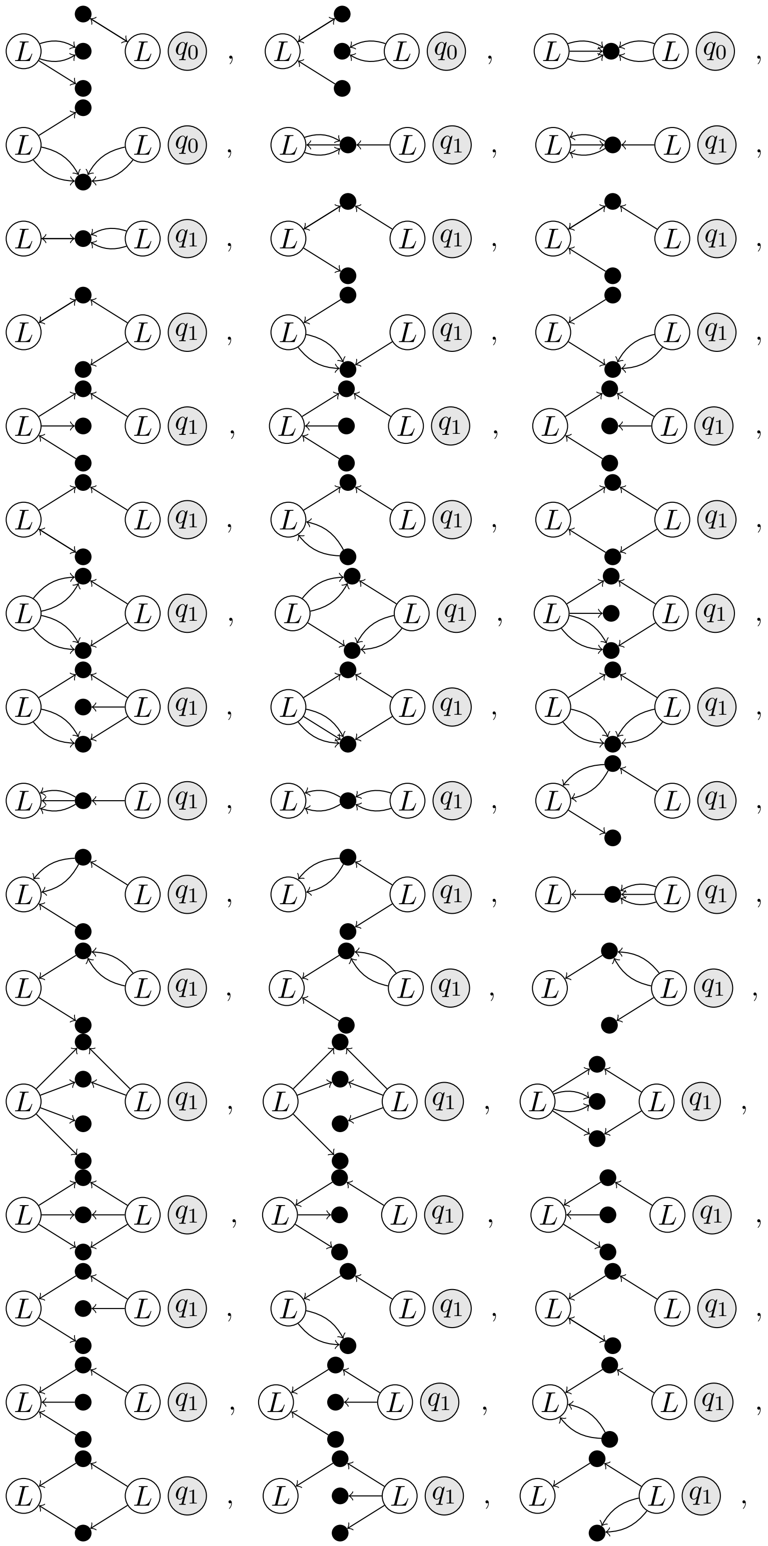
$\mathcal{B}^{\leq 2}_{\text{Safe}}$: 

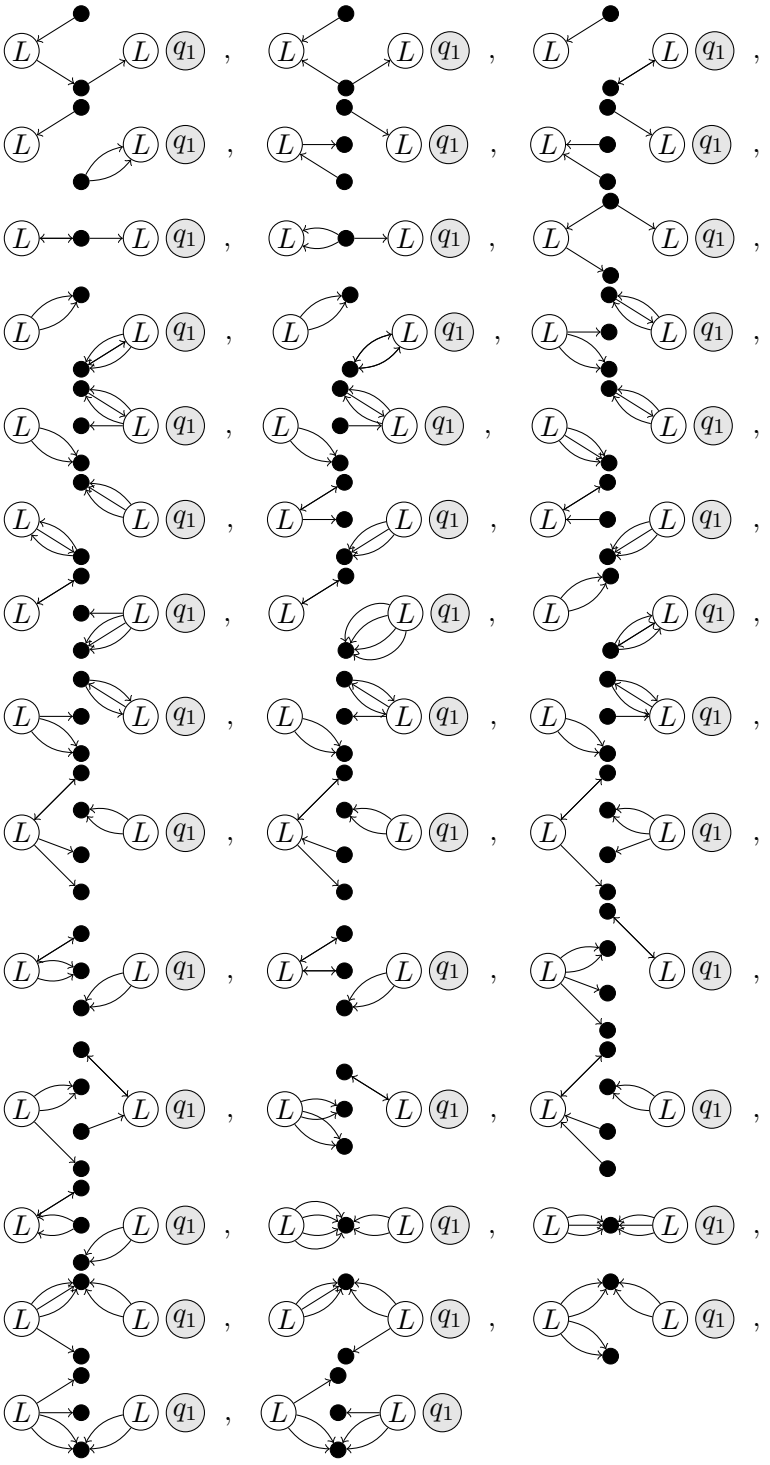
$\mathcal{B}^{\leq 3}_{\text{Safe}}$:



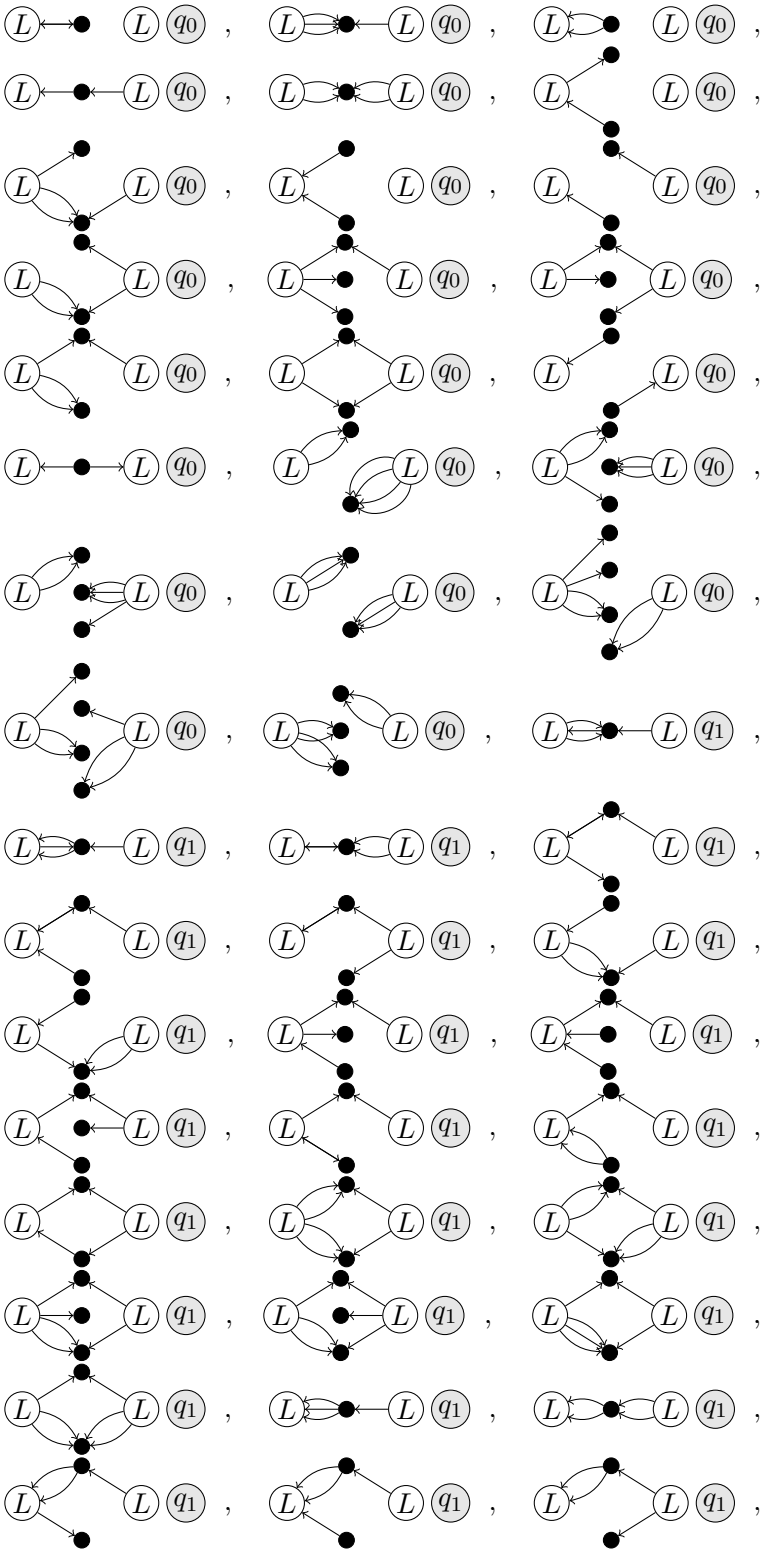
$\mathcal{B}^{\leq 4}_{\text{Safe}}$:

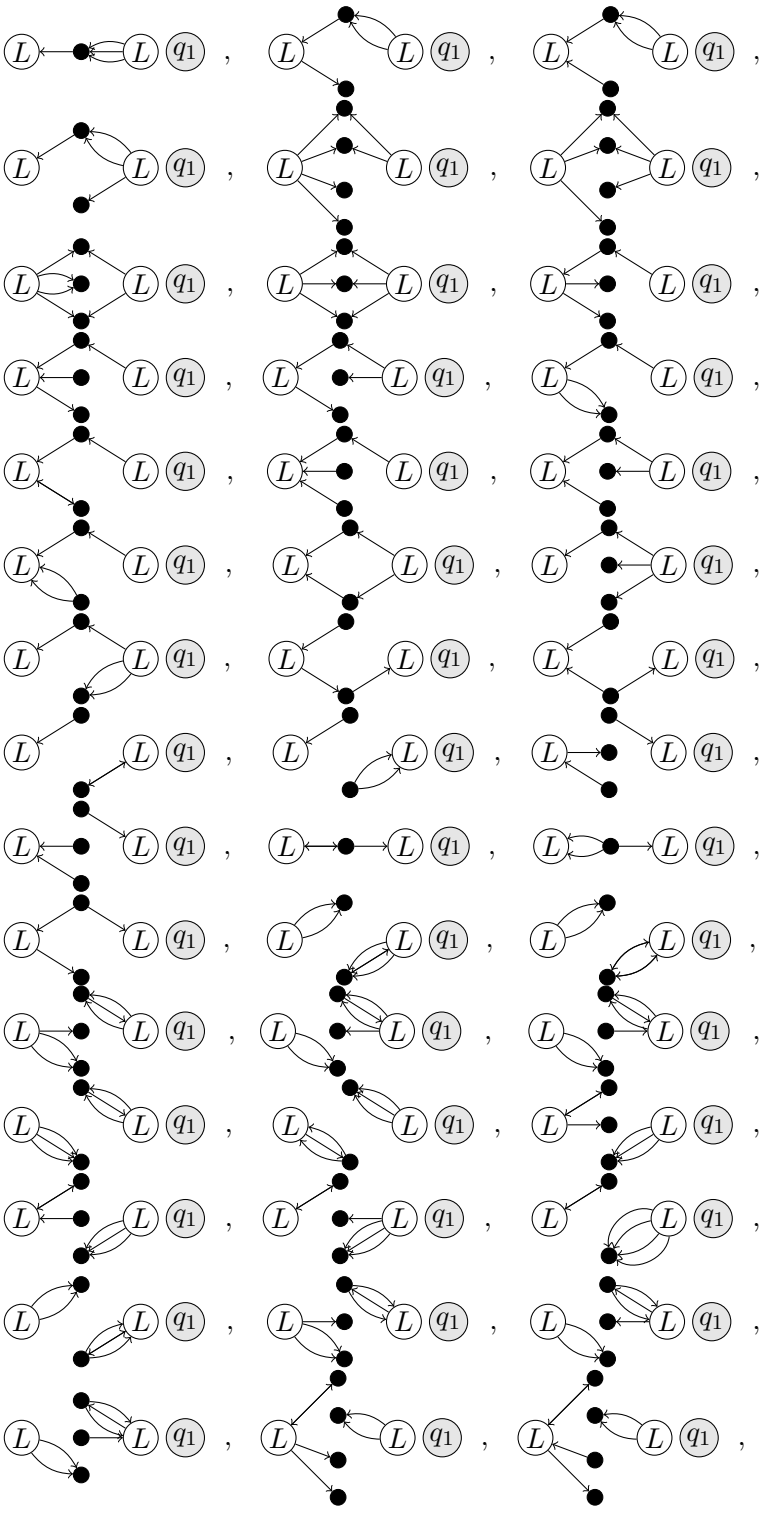




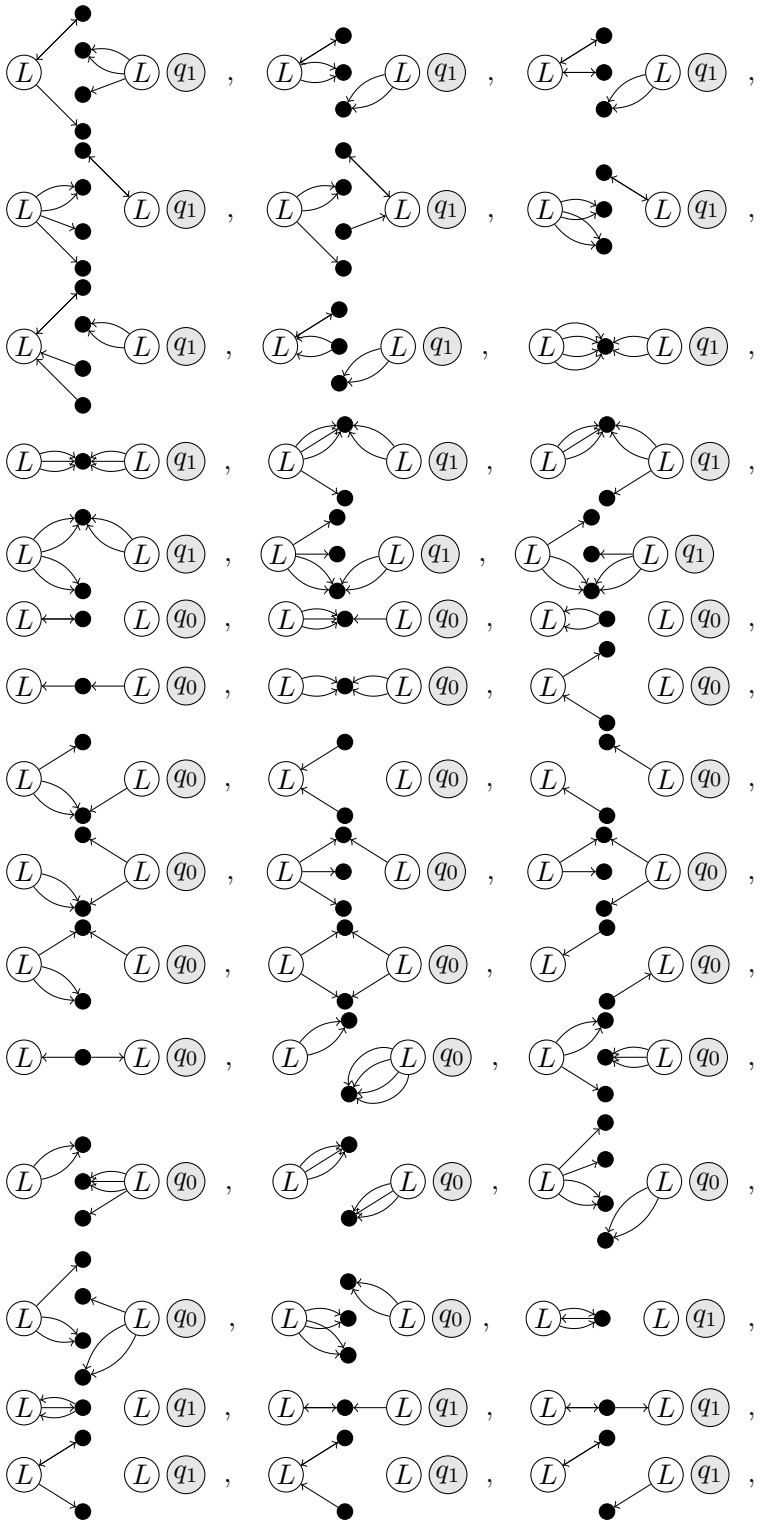

L
L
q_1

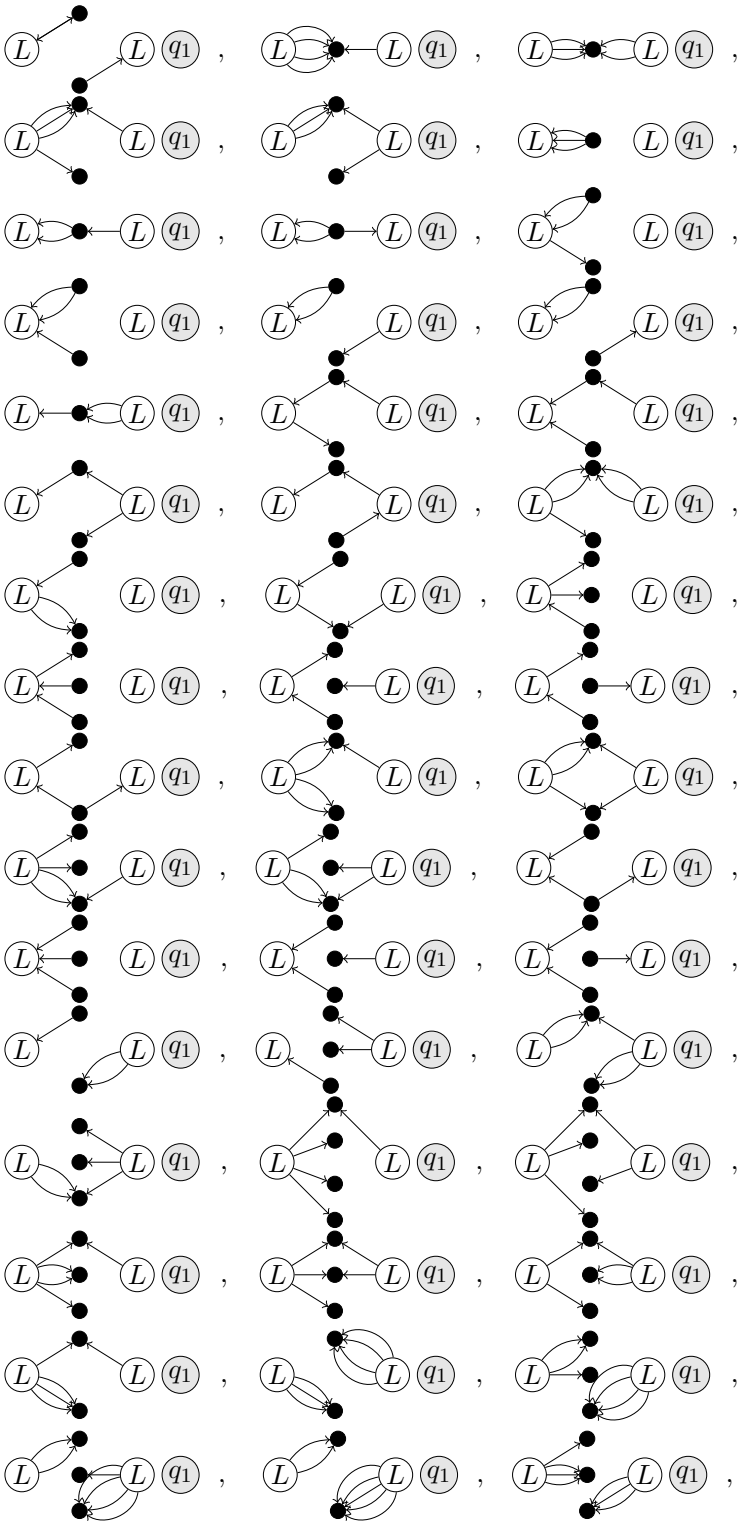
$\mathcal{B}^{\leq 5}_{\text{Safe}}$:



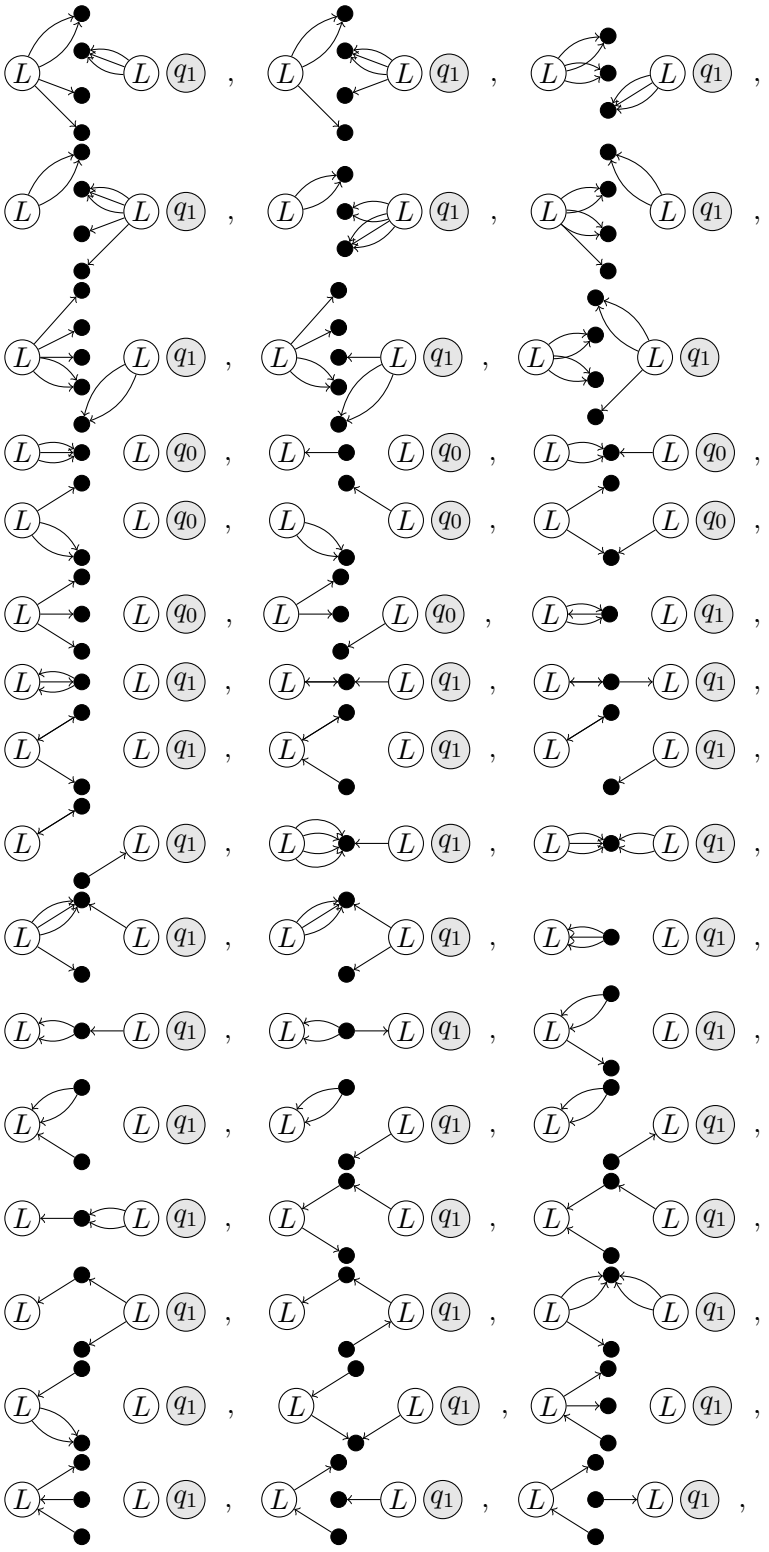

L
L
q_1

$\mathcal{B}^{\leq 6}_{\text{Safe}}$:

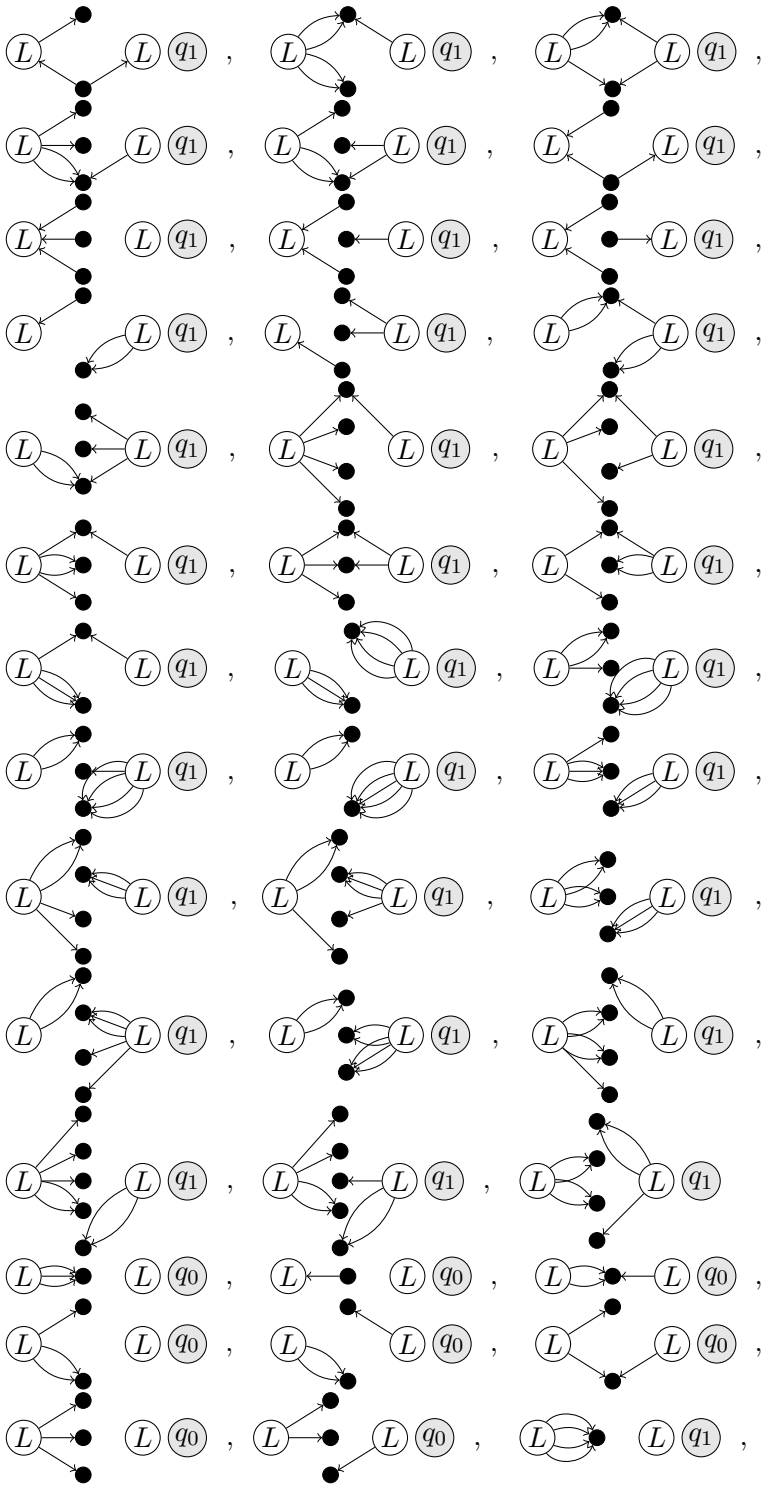


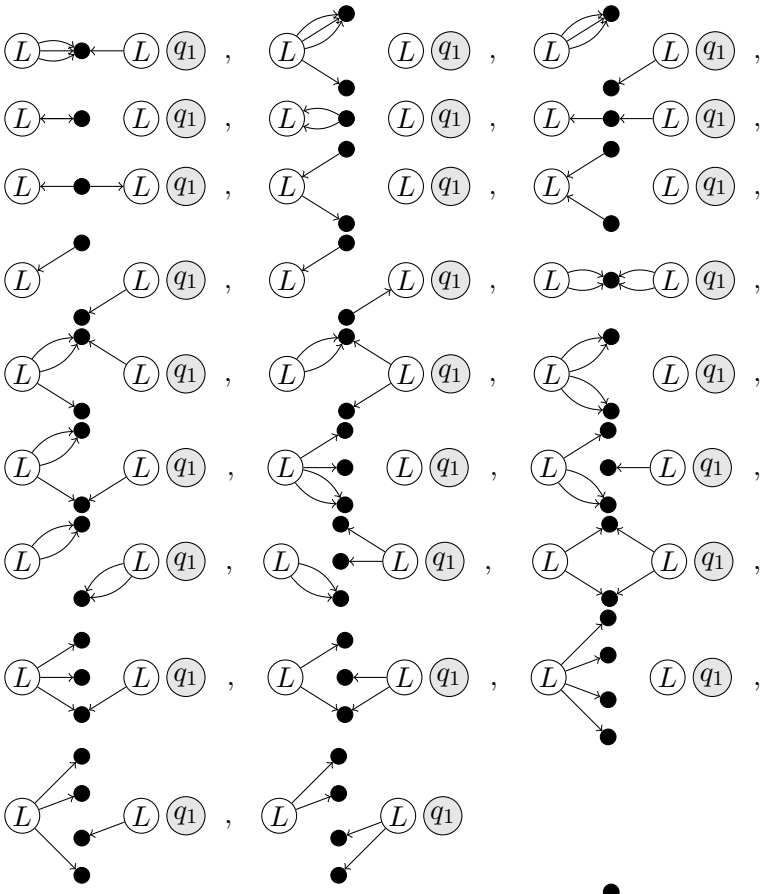

L
L
q_1

$\mathcal{B}^{\leq 7}_{\text{Safe}}$:

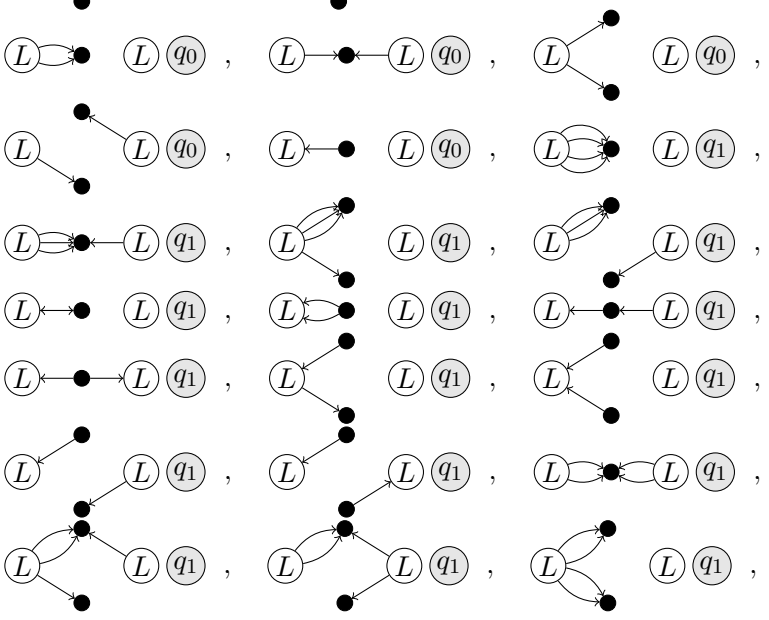


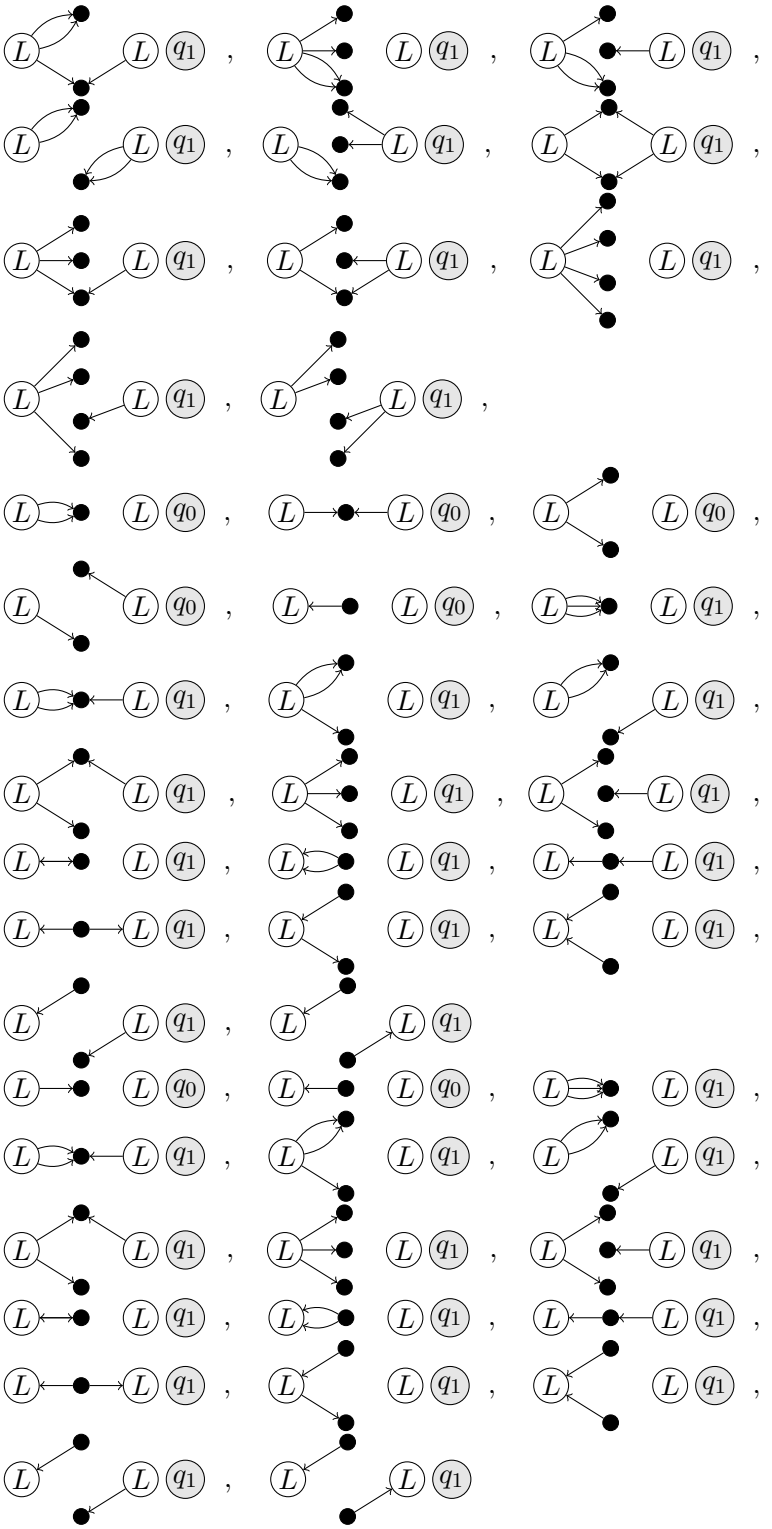
$\mathcal{B}^{\leq 8}_{\text{Safe}}$:



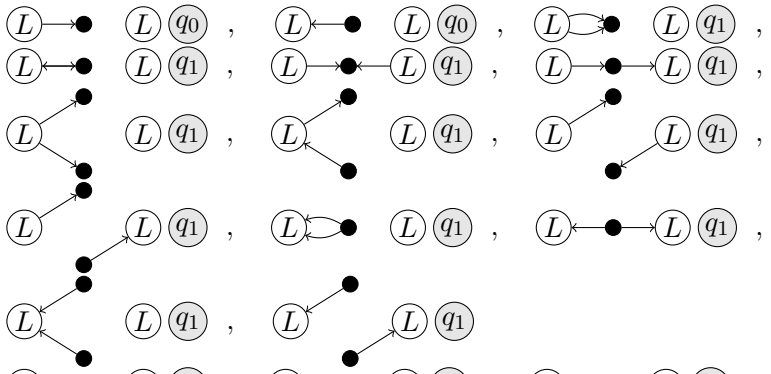


$\mathcal{B}^{\leq 9}_{\text{Safe}}$:

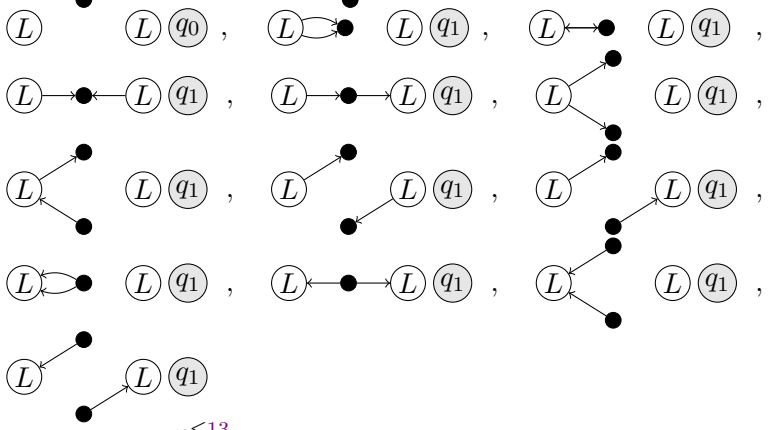



$\mathcal{B}^{\leq 10}_{\text{Safe}}$:
$\mathcal{B}^{\leq 11}_{\text{Safe}}$:

$\mathcal{B}^{\leq 12}_{\text{Safe}}$:



$\mathcal{B}^{\leq 13}_{\text{Safe}}$:



$\Rightarrow$ $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 13}_{\text{Safe}}$

We find that $\mathcal{F}_{\text{INIT,Bad}} \subseteq\uparrow \mathcal{B}^{\leq 13}_{\text{Safe}}$. Thus, $k_{\min} = 13$.

Conclusion
post*(INIT)$\cap[\![\text{Bad}]\!]_S \subseteq \text{pre}^{\leq 13}([\![\text{Safe}]\!]_S)$ minimal
$k_{\min}$: 13

Bibliography

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced concepts and tools for in-place EMF model transformations. In *Proc. 13th Int. Conference on Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135, 2010.

[ACD+17] S. Akshay, Supratik Chakraborty, Ankush Das, Vishal Jagannath, and Sai Sandeep. On Petri nets with hierarchical special arcs. In *Proc. 28th Int. Conference on Concurrency Theory*, volume 85 of *LIPIcs*, pages 40:1–40:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[AČJT96] Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proc. 11th Annual Symposium on Logic in Computer Science*, pages 313–321. IEEE, 1996.

[AGH+21] S. Akshay, Blaise Genest, Loïc Hélouët, Shankara Narayanan Krishna, and Sparsa Roychowdhury. Resilience of timed systems. In *Proc. 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 213 of *LIPIcs*, pages 33:1–33:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[AOdB09] Krzysztof R. Apt, Ernst-Rüdiger Olderog, and Frank S. de Boer. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 2009.

[BCGM10] Paolo Baldan, Andrea Corradini, Fabio Gadducci, and Ugo Montanari. From Petri nets to graph transformation systems. *Electron. Commun. EASST*, 26, 2010.

[BCK08] Paolo Baldan, Andrea Corradini, and Barbara König. A framework for the verification of infinite-state graph transformation systems. *Inf. Comput.*, 206(7):869–907, 2008.

[BCM05] Paolo Baldan, Andrea Corradini, and Ugo Montanari. Relating SPO and DPO graph rewriting with Petri nets having read, inhibitor and reset arcs. *Electron. Notes Theor. Comput. Sci.*, 127(2):5–28, 2005.

[BDK+12] Nathalie Bertrand, Giorgio Delzanno, Barbara König, Arnaud Sangnier, and Jan Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *Proc. 23rd Int. Conference on Rewriting Techniques and Applications*, volume 15 of *LIPIcs*, pages 101–116. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

[BFG20] Michael Blondin, Alain Finkel, and Jean Goubault-Larrecq. Forward analysis for wsts, part III: Karp-Miller trees. *Log. Methods Comput. Sci.*, 16(2), 2020.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BM99] Ahmed Bouajjani and Richard Mayr. Model checking lossy vector addition systems. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333, 1999.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, 1982.

[CLL+21] Wojciech Czerwiński, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. *J. ACM*, 68(1):7:1–7:28, 2021.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 193–242. Elsevier, 1990.

[Cou97] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 313–400. World Scientific, 1997.

[DFS98] Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th Int. Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, 1998.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[Din92] Guoli Ding. Subgraphs and well-quasi-ordering. *J. Graph Theory*, 16(5):489–502, 1992.

[Dol00] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[EHK+97] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. Algebraic approaches to graph transformation - part II: single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 247–312. World Scientific, 1997.

[Eme90] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier, 1990.

[EN94] Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets. *BRICS Report Series*, 1(8), 1994.

[EPS73] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *Proc. 14th Annual Symposium on Switching and Automata Theory*, pages 167–180. IEEE Computer Society, 1973.

[Fli16] Nils Erik Flick. *Proving correctness of graph programs relative to recursively nested conditions*. PhD thesis, University of Oldenburg, 2016.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[GMSS19] Holger Giese, Maria Maximova, Lucas Sakizloglou, and Sven Schneider. Metric temporal graph logic over typed attributed graphs. In *Proc. 22nd Int. Conference on Fundamental Approaches to Software Engineering*, volume 11424 of *Lecture Notes in Computer Science*, pages 282–298, 2019.

[Hab92] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer, 1992.

[Hac85] Michel Hack. *Decidability questions for Petri nets*. PhD thesis, M.I.T., 1985.

[HHT96] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundam. Informaticae*, 26(3/4):287–313, 1996.

[HKK09] Karsten Hölscher, Hans-Jörg Kreowski, and Sabine Kuske. Autonomous units to model interacting sequential and parallel processes. *Fundam. Informaticae*, 92(3):233–257, 2009.

[HP09] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Math. Struct. Comput. Sci.*, 19(2):245–296, 2009.

[HST18] Annegret Habel, Christian Sandmann, and Tilman Teusch. Integration of graph constraints into graph grammars. In *Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig*, volume 10800 of *Lecture Notes in Computer Science*, pages 19–36, 2018.

[JF13] Scott Jackson and Timothy L. J. Ferris. Resilience principles for engineered systems. *Syst. Eng.*, 16:152–164, 2013.

[JK08] Salil Joshi and Barbara König. Applying the graph minor theorem to the verification of graph transformation systems. In *Proc. 20th Int. Conference on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 214–226. Springer, 2008.

[KK08] Barbara König and Vitali Kozioura. AUGUR 2 - A new version of a tool for the analysis of graph transformation systems. *Electron. Notes Theor. Comput. Sci.*, 211:201–210, 2008.

[KM69] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[KR06] Harmen Kastenberg and Arend Rensink. Model checking dynamic states in GROOVE. In *Proc. 13th Int. Conference on Model Checking Software*, volume 3925 of *Lecture Notes in Computer Science*, pages 299–305, 2006.

[Kre80] Hans-Jörg Kreowski. A comparison between petri-nets and graph grammars. In *Proc. Int. Workshop on Graph-theoretic Concepts*

in Computer Science, volume 100 of *Lecture Notes in Computer Science*, pages 306–317. Springer, 1980.

[KS17] Barbara König and Jan Stückrath. Well-structured graph transformation systems. *Inf. Comput.*, 252:71–94, 2017.

[Löw91] Michael Löwe. *Extended algebraic graph transformation.* PhD thesis, Technical University of Berlin, Germany, 1991.

[Löw93] Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theor.Comput. Sci.*, 109:181–224, 1993.

[May84] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13, 1984.

[Men97] E. Mendelsohn. *Introduction to mathematical logic.* Chapman & Hall, 1997.

[Mil85] E. C. Milner. *Basic WQO- and BQO-Theory.* Springer Netherlands, 1985.

[MTF13] Nils Müllner, Oliver E. Theel, and Martin Fränzle. Combining decomposition and reduction for state space analysis of a self-stabilizing system. *J. Comput. Syst. Sci.*, 79(7):1113–1125, 2013.

[OFTK21] Ernst-Rüdiger Olderog, Martin Fränzle, Oliver E. Theel, and Paul Kröger. System correctness under adverse conditions. *it Inf. Technol.*, 63(5-6):249–251, 2021.

[ÖW21] Okan Özkan and Nick Würdemann. Resilience of well-structured graph transformation systems. In *Proc. 12th Int. Workshop on Graph Computational Models*, volume 350 of *Electronic Proceedings in Theoretical Computer Science*, pages 69–88, 2021.

[Özk20] Okan Özkan. Modeling adverse conditions in the framework of graph transformation systems. In *Proc. 11th Int. Workshop on Graph Computational Models*, volume 330 of *Electronic Proceedings in Theoretical Computer Science*, pages 35–54, 2020.

[Özk21] Okan Özkan. Infinite-state graph transformation systems under adverse conditions. *it Inf. Technol.*, 63(5-6):311–320, 2021.

[Özk22] Okan Özkan. Decidability of resilience for well-structured graph transformation systems. In *Proc. 15th Int. Conference on Graph Transformation*, volume 13349 of *Lecture Notes in Computer Science*, pages 38–57, 2022.

[Par92] Francesco Parisi-Presicce. Single vs. double pushout derivations of graphs. In *Proc. 18th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 657 of *Lecture Notes in Computer Science*, pages 248–262, 1992.

[Pen09] Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, University of Oldenburg, 2009.

[Peu18] Christoph Peuser. From hyperedge replacement grammars to decidable hyperedge replacement games. In *Proc. 9th Int. Workshop on Graph Computational Models at Software Technologies: Applications and Foundations*, volume 11176 of *Lecture Notes in Computer Science*, pages 463–478, 2018.

[PP13] Christopher M. Poskitt and Detlef Plump. Verifying total correctness of graph programs. *Electron. Commun. EASST*, 61, 2013.

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[Ren04] Arend Rensink. Representing first-order logic using graphs. In *Proc. Int. Conference on Graph Transformation*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335, 2004.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[San22] Christian Sandmann. *A Theory on Graph Generation and Graph Repair with Application to Meta-Modeling*. PhD thesis, University of Oldenburg, 2022.

[Sch18] Maike Schwammberger. An abstract model for proving safety of autonomous urban traffic. *Theor. Comput. Sci.*, 744:143–169, 2018.

[SS13] Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *Proc. 24th Int. Conference on Concurrency Theory*, pages 5–24, 2013.

[Ste15] Dominik Steenken. *Verification of infinite-state graph transformation systems via abstraction*. PhD thesis, University of Paderborn, 2015.

[Stü15] Jan Stückrath. UNCOVER: Using coverability analysis for verifying graph transformation systems. In *Proc. 8th Int. Conference on Graph Transformation*, volume 9151 of *Lecture Notes in Computer Science*, pages 266–274, 2015.

[Stü16] Jan Stückrath. *Verification of Well-Structured Graph Transformation Systems*. PhD thesis, University of Duisburg-Essen, 2016.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–191. Elsevier and MIT Press, 1990.

[TKG09] Kishor S. Trivedi, Dong Seong Kim, and Rahul Ghosh. Resilience in computer systems and networks. In *Proc. Int. Conference on Computer-Aided Design*, pages 74–77. IEEE/ACM, 2009.

[WKK19] Nils Worzyk, Hendrik Kahlen, and Oliver Kramer. Physical adversarial attacks by projecting perturbations. In *Proc. 28th Int. Conference on Artificial Neural Networks and Machine Learning*, volume 11729 of *Lecture Notes in Computer Science*, pages 649–659, 2019.

List of Symbols

Graphs & States	
G, H, C	graph
s	state
S	state or graph set
B	basis graph
b	basis element
$\mathcal{B}$	basis
q	state (automaton)
Q	state set (automaton)
Morphisms	
$\rightharpoonup$	partial morphism
$\to$	total morphism
$\hookrightarrow$	total, injective morphism
Quasi-orders & Closures	
$\leq$	subgraph order or a (well-)quasi-order
$\uparrow A$	upward-closure of a set A
$\downarrow A$	downward-closure of a set A
$\mathcal{I}$	set of ideals with a given basis
$\mathcal{J}$	set of decidable anti-ideals
Rules	
$r = \langle p : L \rightharpoonup R \rangle$	rule (given by the morphism $p : L \rightharpoonup R$)
$\mathcal{R}$	finite set of rules
$\mathcal{R}_{\text{loss}}(S)$ / $\mathcal{R}_{\perp}(S)$	lossy/bottom rules
$\mathcal{S}$	system rules
$\mathcal{E}$	environment rules
$\mathcal{SE}$	joint GTS
Constraints	
c	graph constraint
$[\![c]\!]$	all graphs satisfying c
$[\![c]\!]_S$	all graphs in S satisfying c
Transformations & Transitions	
$\Rightarrow$	direct transformation
$\Rightarrow_{\mathcal{R}}$	direct transformation via a rule $r \in \mathcal{R}$
$\to$	direct transition
δ	direct transition (automaton)
$\Rightarrow^*$, $\Rightarrow_{\mathcal{R}}^*$, $\to^*$	transformation/transition of length ≥ 0
$\Rightarrow^+$, $\Rightarrow_{\mathcal{R}}^+$, $\to^+$	transformation/transition of length ≥ 1
$\Rightarrow^k$, $\Rightarrow_{\mathcal{R}}^k$, $\to^k$	transformation/transition of length k
$\Rightarrow^{\leq k}$, $\Rightarrow_{\mathcal{R}}^{\leq k}$, $\to^{\leq k}$	transformation/transition of length $\leq k$
Pre- & Postsets	
$\text{pre}(A)$, $\text{pre}_S(A)$	all direct predecessors of A (in S)
$\text{pre}^*(A)$, $\text{pre}^{\leq k}(A)$	all predecessors (of $\leq k$ steps) of A
$\text{post}(A)$	all from A directly reachable elements
$\text{post}^*(A)$, $\text{post}^{\leq k}(A)$	all from A (in $\leq k$ steps) reachable elements
Simulations	
$\sqsubseteq$	simulation
$\sqsubseteq^{\bullet}$	weak simulation

Abbreviations

$\bot$	bottom
$\cap$	intersection
CTL	computation tree logic
DPO	double-pushout approach
GTS	graph transformation system
LTL	linear temporal logic
SPO	single-pushout approach
SWSTS	strongly well-structured transition system
WSTS	well-structured transition system
wqo	well-quasi-order