Electronic Communications of the EASST
Volume 39 (2011)

Graph Computation Models
Selected Revised Papers from the
Third International Workshop on
Graph Computation Models (GCM 2010)

Incremental update of constraint-compliant policy rules

Paolo Bottoni, Andrew Fish, Francesco Parisi Presicce

18 pages

Guest Editors: Rachid Echahed, Annegret Habel, Mohamed Mosbah
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Incremental update of constraint-compliant policy rules

Paolo Bottoni[1], **Andrew Fish**[2], **Francesco Parisi Presicce**[1]

Department of Computer Science, Sapienza University of Rome, Italy[1], School of Computing, Engineering and Mathematics, University of Brighton, UK[2]

Abstract: Organizations typically define policies to describe (positive or negative) requirements about strategic objectives. Examples are policies relative to the security of information systems in general or to the control of access to an organization's resources. Often, the form used to specify policies is in terms of general constraints (what and why) to be enforced via the use of rules (how and when). The consistency of the rule system (transforming valid states into valid states) can be compromised and rules can violate some constraints when constraints are updated due to changing requirements. Here, we explore a number of issues related to constraint update, in particular proposing a systematic way to update rules as a consequence of modifications of constraints, by identifying which components of the rule have to be updated. Moreover, we show the construction of sets of rules, directly derived from a positive constraint, to guarantee constraint preservation and constraint enforcement.

Keywords: Constraints, incremental construction of rules, rule update.

1 Introduction

Graph constraints and graph transformations provide a formal model for defining and managing strategic policies adopted by organisations with respect to security or access control issues. Policies can be specified with constraints, while rules of a graph transformation system are used for consistent management of these policies. In both cases, the specifications are given within the framework of typed attributed graphs, with special attributes defining the relevant properties, while rules exploit application conditions to enforce consistency. By consistency, we mean that rules can only cause transformations from valid states to valid states. As a consequence, the class of graphs generated, starting from a valid graph, by using the rule is a subset of the class of graphs satisfying the constraints.

Usually, policies and rules are not fixed, but can evolve independently, following the introduction, revision or deletion of constraints and rules. In any case, at each moment the current set of constraints must be satisfied by the current collection of rules. Two cases can be considered in which the consistency of the system can be compromised: introduction of new rules – which must be adapted to the current constraints – and definition of new constraints, again requiring rule adaptation. The identification of the modifications needed to adapt rules to constraints has been the subject of several studies [KMP05, HP09]. However, these usually require the complete analysis of each rule after each modification, without the possibility to adopt an incremental approach, taking into account the relations between the previous sets of constraints and rules.

In this paper, we explore this problem offering two major contributions. First, we define a procedure which constructs a new collection of rules from the specification of a positive constraint, where each rule is guaranteed to preserve the constraint. Second, we show how these rules can be incrementally updated when the constraint from which they were created is modified. The paper is complemented by two further contributions. First, we produce a set of rules to check and repair a model with respect to the introduction of a new constraint. Second, we present a technique for checking if a rule, not generated by the procedure, has to be updated when a constraint is modified, and for computing the relevant updates.

Paper organisation. After presenting related work in Section 2, we provide background notions on graph constraints and graph transformations in Section 3. Section 4 introduces the running example for the paper, while Section 5 recalls the notion of security policy and presents a procedure for the construction of a collection of rules realising a policy. Section 6 discusses modifications to rules derived from the procedure and Section 7 draws conclusions and points to future work.

2 Related work

In [KMP05], a security policy framework is defined by giving a type graph, a set of (named) graph transformation rules, and two sets of simple positive and negative constraints, expressed as morphisms from a premise to a conclusion. A framework was considered to be (positive/negative) coherent if all the graphs derivable from the rules and consistent with the type graph satisfied all the (positive/negative) constraints. A number of constructions were given for repairing possible violations, by modifying rules with application conditions, in situations where different policy frameworks were merged. Based on this approach, [KP06] defines a UML language to specify access control policies and exploits graph transformations to give a semantics and a verification method for such specifications. This line of research has also led to defining basic rules for controlling access in workflows [WWP08]. Rules allow the addition/removal of tasks and roles, plus the execution of tasks, and application conditions are defined based on existing constraints.

In [BFP10], we considered the construction of transformation units ensuring coherence with a simple form of nested constraints, admitting a single alternation of universal and existential quantifiers. The units were defined starting with the simple addition of an element, and producing repair rules to restore violated constraints. The procedures presented in this paper can also be exploited in transformation units, but each rule is guaranteed not to violate any constraint.

Habel and Pennemann have extensively treated the problem of making rules compatible with nested constraints by adding positive and negative application conditions. In [HP09] they unify theories about application conditions [EEHP06] and nested graph conditions [Ren04], lifting them to high-level transformations. An existing rule is transformed to make it constraint preserving or constraint guaranteeing, but no direct construction of rules from constraints is given.

Orejas is investigating a new approach, called *Symbolic Attributed Graphs*, relating attributed graph constraints with attribute evaluation (for a summary see [OL10]). For our restricted form of attributed evaluation, the presented techniques should be applicable with the same results.

Some works have studied the problem of constructing instances of metamodels, specialised by additional constraints. In [JKW08] constraints are given in a logical language, while [EKTW06] uses OCL and tests constraints after rule derivation from the metamodel. Using the terminology of [JKW08], ours is a soundness-preserving, rather than completeness-preserving, construction.

While [KMP05] studied the modifications required by introducing new constraints, or merging constraint systems, the consequences of modifying (weakening or strengthening) existing constraints were not analysed, nor does this problem seem to have been treated by other authors.

3 Background

We adopt the framework of attributed typed graphs. A graph $G = (V,E,s,t)$ consists of a set of *nodes* $V = V(G)$, a set of *edges* $E = E(G)$, and *source* and *target* functions, $s,t : E \to V$. For a set S, its cardinality is denoted by $| S |$ and we use the notations $| G |_V = | V(G) |$ and $| G |_E = | E(G) |$.

In a *type graph* $TG = (V_T, E_T, s^T, t^T)$, V_T and E_T are sets of node and edge types, while the functions $s^T : E_T \to V_T$ and $t^T : E_T \to V_T$ define source and target node types for each edge type.

For G a typed graph on TG, there is a graph morphism $type\colon G \to TG$, with $type_V : V \to V_T$ and $type_E\colon E \to E_T$ s.t. $type_V(s(e)) = s^T(type_E(e))$ and $type_V(t(e)) = t^T(type_E(e))$.

To introduce attributes, we follow [EEHP06], but we consider attributes only on nodes. Intuitively, we partition V into V_G and V_D (D for *domain*), the sets of *graph* and *value* nodes, respectively, while E is partitioned into E_G and E_A (A for *attribute*). Graph edges E_G are equivalent to those for non-attributed graphs, while an *attribute edge* in the set E_A defines the assignment of a value to an attribute of a node. Moreover, we have $s = s_V \cup s_A$, with $s_V\colon E_G \to V_G$ and $s_A\colon E_A \to V_G$; and $t = t_V \cup t_A$, with $t_V : E_G \to V_G$ and $t_A : E_A \to V_D$. In a similar way, TG has distinct sets V_T^G and V_T^D of graph and value node types respectively, as well as distinct sets E_T^G and E_T^A for graph and attribute edge types. Given $t \in V_T^G$, all nodes of type t are associated with the same subset $A(t) \subset E_T^A$ of edge types, corresponding to the set of attribute names for t. Formally: $\forall t \in V_T^G[\exists! A(t) \subset E_T^A[\forall n \in V_G[type_V(n) = t \implies \{type_E(e) \mid e \in E_A \wedge s_A(e) = n\} = A(t)]]]$[1]. Values in V_D are taken over the disjoint union of the set of sorts in a *data signature DSIG*. In what follows, we call the *structural part* of G, its projection to elements typed on V_T^G and E_T^G.

We use graph transformations according to the Double PushOut (DPO) approach [EEPT06]. A DPO rule has the form $p : L \xleftarrow{l} K \xrightarrow{r} R$ where L, K and R are called left-hand side, interface and right-hand side graphs, respectively, K contains the elements preserved by the rule application, and the morphisms $l\colon K \to L$ and $r\colon K \to R$ model the embedding of K in L and R. The left of Figure 1 shows a DPO direct derivation diagram, modeling the application of a rule[2] $p : L \xleftarrow{l} K \xrightarrow{r} R$ to a host graph G to produce a target graph H. First, the pushout complement D is evaluated. This is the unique graph for which morphisms $K \to D \to G$ exist s.t. square (1) is a pushout (i.e. G is the union of L and D through their common elements in K). In particular, D contains the elements of G which are not in the image of the elements in $L \setminus K$. The pushout (2) is then computed, adding to D new elements to form H, viz. the elements in $R \setminus K$. We denote the application of the rule p on a match $m\colon L \to G$ by $G \Rightarrow_p^m H$ and write $G \Rightarrow_p H$ to denote that H can be derived from G by applying p with respect to some match m for L in G.

DPO rules for typed graphs are lifted to attributed typed graphs by considering algebras on some signature including the sorts for V_D. Since morphisms can only identify values in V_D present in L and R, the modification of the value of an attribute a for a given node n from v_1 to v_2 is represented by removing an edge e_1 of type a from n to v_1 (i.e. e appears in L but not in K), and creating an edge e_2 of the same type a from n to v_2 (i.e. e_2 appears in R but not in K).

[1] In the graphs of this paper, some attributes may not be present for some node, indicating “don’t care” situations.
[2] Where no ambiguity arises, we will omit explicit mention of morphisms l and r.

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"'] \arrow[dr, phantom, "(2)"] & R \arrow[d, "m^*"] \\
G & D \arrow[l, "l^*"] \arrow[r, "r^*"'] & H
\end{tikzcd}
\qquad
\begin{tikzcd}
Y_{ij} \arrow[drrr, bend right, "o_{ij}"', "="] & X_i \arrow[l, "y_{ij}"] \arrow[drr, "n_i"', "="] & L \arrow[l, "x_i"'] \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"'] \arrow[dr, phantom, "(2)"] & R \arrow[d, "m^*"] \\
 & & G & D \arrow[l, "l^*"] \arrow[r, "r^*"'] & H
\end{tikzcd}$$

Figure 1: DPO Direct Derivation Diagram for simple rules (left) and with AC (right).

In this paper we consider only rules which are only increasing ($L = K$) or only decreasing ($K = R$) in their structural parts. However, from Proposition 1, rules of this form can be combined to express any DPO rule. To this end, the construction of an E-concurrent production (see Figure 2), denoted by $*_E$, is required, for an appropriate choice of the E-dependency relation [EEPT06].

Proposition 1 *Any rule $p : L \overset{l_p}{\leftarrow} K \overset{r_p}{\rightarrow} R$ is equivalent to the E-concurrent production of two rules, one increasing and one decreasing.*

Proof. With reference to Figure 2, and with all unnamed morphisms identities, we have:

1. if $p_1 : L \overset{l_p}{\leftarrow} K \overset{i_K}{\rightarrow} K$ and $p_2 : K \overset{i_K}{\leftarrow} K \overset{r_p}{\rightarrow} R$, then p_1 is only decreasing, p_2 is only increasing, and $p = p_1 *_K p_2$. (In this case the role of E is performed by K.)

2. if $L \overset{r_{q_1}}{\rightarrow} Q \overset{l_{q_2}}{\leftarrow} R$ is the pushout of $L \overset{l_p}{\leftarrow} K \overset{r_p}{\rightarrow} R$ and K is also its pullback (as in our case since $L \overset{l_p}{\leftarrow} K$ is injective), then $q_1 : L \overset{i_L}{\leftarrow} L \overset{r_{q_1}}{\rightarrow} Q$ is only increasing, $q_2 : Q \overset{l_{q_2}}{\leftarrow} R \overset{i_R}{\rightarrow} R$ is only decreasing, and $p = q_1 *_Q q_2$. (In this case the role of E is performed by Q.)

Case 1 proves the $\Rightarrow$ direction and case 2 proves the $\Leftarrow$ direction. $\square$

$$\begin{tikzcd}
 & & & K \arrow[dl] \arrow[dr] & & & \\
L \arrow[d] & K \arrow[l, "l_p"'] \arrow[r] \arrow[d] & K \arrow[dr] & & K \arrow[dl] & K \arrow[l] \arrow[r, "r_p"] \arrow[d] & R \arrow[d] \\
L & K \arrow[l, "l_p"'] \arrow[rr] & & K & & K \arrow[ll] \arrow[r, "r_p"] & R \\
 & & & K \arrow[ull] \arrow[urr] & & &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & & Q \arrow[dl] \arrow[dr] & & & \\
L \arrow[d] & L \arrow[l] \arrow[r, "r_{q_1}"] \arrow[d] & Q \arrow[dr] & & Q \arrow[dl] & R \arrow[l, "l_{q_2}"'] \arrow[r] \arrow[d] & R \arrow[d] \\
L & L \arrow[l] \arrow[rr, "r_{q_1}"] & & Q & & R \arrow[ll, "l_{q_2}"'] \arrow[r] & R \\
 & & & K \arrow[ull, "l_p"] \arrow[urr, "r_p"'] & & &
\end{tikzcd}$$

Figure 2: Decomposing (left) and constructing (right) a rule with increasing and decreasing parts.

An *atomic constraint* is a morphism[3] of typed attributed graphs $c : X \rightarrow Y$. A graph G *satisfies* c, noted $G \vDash c$, if for each *match morphism* $m\colon X \rightarrow G$ there exists a morphism $y\colon Y \rightarrow G$ s.t. $y \circ c = m$. If $c\colon X \rightarrow Y$ is an atomic constraint, $\neg c$ is an atomic constraint, and $G \vDash \neg c$ *iff* $G \nvDash c$. An atomic constraint of the form $\neg i_X\colon X \rightarrow X$, where i_X is the identity, is called a *negative atomic constraint*, and is represented by simply showing X. We call $M(c) = \{G \mid G \vDash c\}$ the set of *models* for c, and assume that a constraint system C is consistent, i.e. $\bigcap_{c_i \in C} M(c_i) \neq \emptyset$.

The right of Figure 1 shows that an atomic constraint can be associated with a rule in the form of an *application condition* AC, of the form $\{x_i\colon L \rightarrow X_i, \{y_{ij}\colon X_i \rightarrow Y_{ij}\}_{j \in J_i}\}_{i \in I}$, for a match

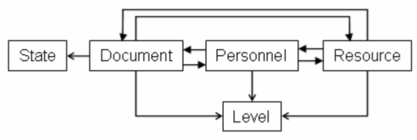
[3] Except typing morphisms, all graph morphisms in the paper are inclusions.

$m: L \rightarrow G$ of the LHS of a rule, where I and J_i are index sets for each $i \in I$. An AC is satisfied by m if, for each $n_i: X_i \rightarrow G$ s.t. $n_i \circ x_i = m$, there exists some $o_{ij}: Y_{ij} \rightarrow G$ s.t. $o_{ij} \circ y_{ij} = n_i$. A *negative application condition* (NAC) derives from a negative application constraint: for the rule to be applicable, X_i must not be present.

4 A scenario for policy update

We consider policies on type graphs which are instances of the metamodel MM in Figure 3 (left). Here, `Document`, `Personnel` and `Resource` are meta-types for graph node types, defining the structural elements involved in a policy, while `State` and `Level` are metatypes for attribute types. In this case, the *structural* nodes are those whose type is an instance of `Document`, `Personnel` or `Resource`.

Figure 3 (right) presents the specific type graph for the paper, where type *Doc* is an instance of `Document`, *Mgr*, representing managers, is an instance of `Personnel`, and the *Printer* type is an instance of `Resource`. The attribute types denote a document authorisation state and different types of access levels associated with structural elements. A concrete representation is adopted, where rectangular boxes represent instances of structural nodes in V_G and ovals represent attribute values. Attribute edges are distinguished from graph edges by the arrow end.



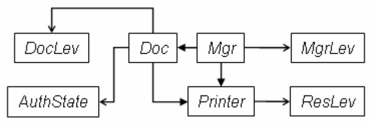


Figure 3: Metamodel for security policies (left) and type graph for the scenario (right).

A simple scenario illustrates the type of problems considered, introducing the notation exploited in the paper, Consider an organisation in which the policy for document authentication requires that, for a document D to be authenticated, it must have been signed off by two managers, one of level $Lv1$ and one of level $Lv2$, $Lv2$ being higher than $Lv1$ in the organisation hierarchy. Moreover, a general policy requires that documents signed off by $Lv2$ managers must have also been signed off by an $Lv1$ manager. Figure 4 illustrates the two resulting constraints.

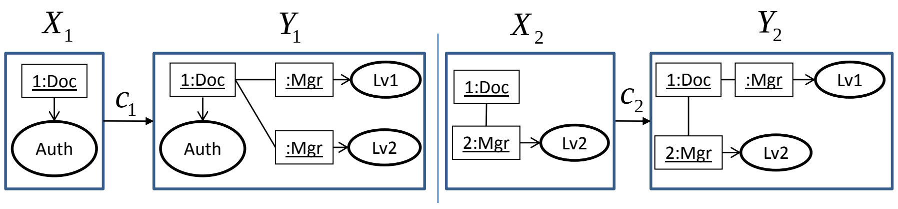


Figure 4: The two original constraints for document authentication.

The organisation adopts a collection of rules for authentication which is presented in Figure 5.

First an $Lv1$ manager will provide a reference for the document, and then an $Lv2$ manager will authorise it. The rules are equipped with NACs to avoid that the same manager reviews the same document twice. Note that a rule which requires a simultaneous authorisation by both managers would also be consistent with the constraints in Figure 4.

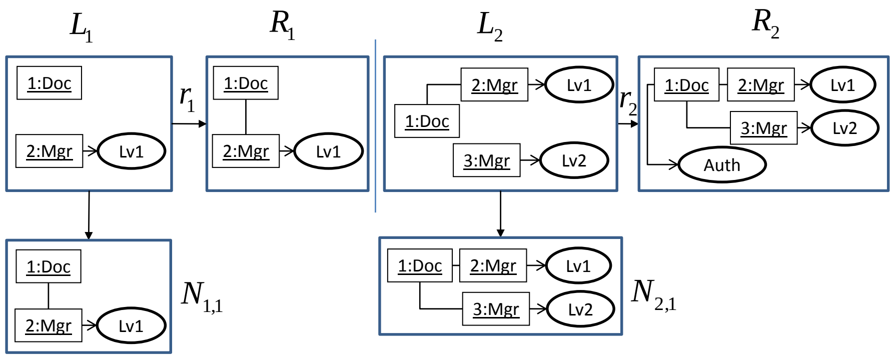


Figure 5: The two original rules for document authentication.

Realising that the current implementation of the policy does not prevent work on the same document from being duplicated, the organisation decides to forbid a document from being authorised twice, or referenced by two or more managers of the same level. Such conditions could be added as NACs to the rules, but a decision is made for these to be defined as negative constraints – shown in Figure 6 as forbidden graphs – $\neg i : X_3 \rightarrow X_3$, $\neg i : X_4 \rightarrow X_4$, and $\neg i : X_5 \rightarrow X_5$, respectively, becoming part of the general policy for document management.

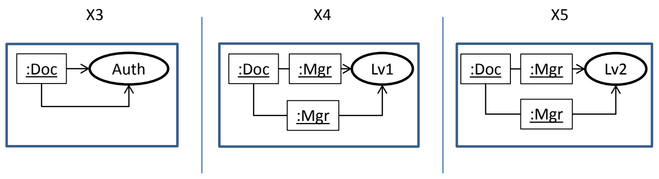


Figure 6: Negative atomic constraints for the authentication policy.

Rules r_1 and r_2 of Figure 5 are no longer consistent with the current configuration of constraints and must be amended by adding NACs to them. In particular, rule r_1 must be upgraded with the NAC $N_{1,2}$ requiring that no other manager of level $Lv1$ has already pre-approved the document, while rule r_2 is upgraded using $N_{2,2}$ to check that no authorisation already exists, besides requiring that no other $Lv2$ manager has approved the document. Figure 7 shows the new versions of the two rules. In a second moment, a general revision of the authorisation policies is issued, now requiring that two $Lv1$ managers must approve a document, while maintaining the request for $Lv2$ authentication, represented by the constraint $c_6 : X_6 \rightarrow Y_6$ in Figure 8, which subsumes $c_1 : X_1 \rightarrow Y_1$. Moreover, the constraint defined by the forbidden graph X_4 must be dropped to ensure the overall satisfiability of the system of constraints.

This modification has impact on both rules r_1 and r_2 in Figure 7. Indeed, condition $N_{1,2}$ for

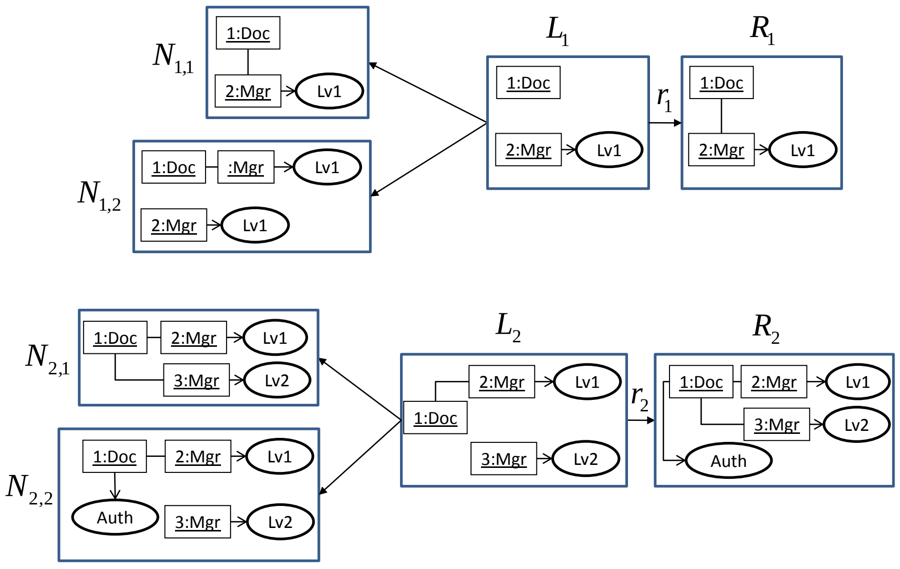


Figure 7: The modified rules with NACs.

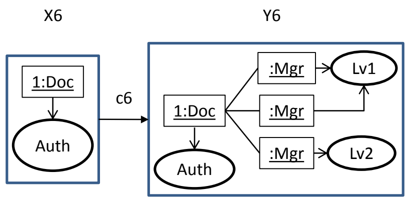


Figure 8: Revising authorisation constraints.

r_1, which was derived from the forbidden graph X_4, now dropped, would prevent the satisfaction of constraint $c_6 : X_6 \rightarrow Y_6$. Moreover, the right-hand side of rule r_2 would not be consistent with c_6, as it might lead to approval of documents without the signature of two $Lv1$ managers. While the solution to the first problem is simple (remove $N_{1,2}$), the second problem can be addressed in different modes: by requiring concurrent approval by the additional $Lv1$ manager and the $Lv2$ one, by sequentialising the approvals of the manager according to the level (i.e. simultaneous approval of both $Lv1$ managers and subsequent $Lv2$ approval) or by considering a sequential process, where the second $Lv1$ and the $Lv2$ approval can occur in any order. Note that since $c_2 : X_2 \rightarrow Y_2$ still holds, it remains impossible to have a collection of rules which leads to $Lv2$ approval without prior or concurrent $Lv1$ approval.

5 Constraint-compliant policy rules

We define the construction of sets of rules for enforcing consistency with policies, where rules are only increasing or decreasing in the structural part and may use only equality or inequality as operations on attributes.

We first propose a procedure for deriving a collection of increasing (in the structural part) rules consistent with a positive constraint, based on the initial pushout construction in the category of Attributed Typed Graphs [EEPT06]. Given the square in Figure 9 (left), this is the *initial pushout* over $c : X \to Y$, *iff* for every other pushout formed with $X \leftarrow Z \to T \to Y$, there exist morphisms $X' \to Z$ and $Q \to T$ such that all the squares in Figure 9 (right) commute. In the adopted framework, the *boundary object* X' is computed as the discrete graph formed by the structural nodes in X which are attached to edges in $E(Y) \setminus E(c(X))$, and the *context* of c is the attributed typed graph Q which is the pushout complement along X' for $X' \xrightarrow{x} X \xrightarrow{c} Y$.

$$\begin{tikzcd}
X' \arrow[r, "x"] \arrow[d, "q'"] & X \arrow[d, "c"] \\
Q \arrow[r, "y"] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
X' \arrow[r] \arrow[rr, bend left, "x"] \arrow[d, "q'"] & Z \arrow[r] \arrow[d] & X \arrow[d, "c"] \\
Q \arrow[r] \arrow[rr, bend right, "y"'] & T \arrow[r] & Y
\end{tikzcd}$$

Figure 9: (left) Boundary and context for a constraint. (right) Initiality of pushout.

We work on a restricted form of positive constraints $c : X \to Y$, for which we can derive a set of constraint-preserving rules. In particular, we define *policy* constraints on single *secured* nodes, describing the conditions under which such a node can be in certain states, as in the examples from Section 4. In particular, structural edges are in $E(Y) \setminus E(X)$ only if both their source and target are in $V_G(Y) \setminus c(V_G(X))$, or if one of them is the secured node.

5.1 The procedure for incremental satisfaction

Given a policy constraint $c : X \to Y$, `ConstructRules(c)` generates a collection $P(c)$ of increasing rules preserving c. Each rule constructs a subgraph of Y, in such a way that until an occurrence of Y is built no occurrence of X is produced in the host graph. To this end, it first constructs the rules for producing graphs intermediate between the boundary object X' and the context Q in the construction of the initial pushout over c, and then produces rules allowing the construction of intermediate graphs between Q and Y. After this, NACs are added to prevent the premature formation of Y and to force the application of the most specialised rule in case of conflicts. Finally, the rules are completed to include attribute assignment.

```
function ConstructRules(c:Constraint):SetOfRules =
    $[\mathscr{Q},\mathscr{H}]$ := FromBoundaryToContext($c$);
    $P_Q(c)$ := FromMorphismsToRules($\mathscr{Q},\mathscr{H}$);
    $P'(c)$ := CompleteToConclusion($P_Q(c)$,$Y$);
    $P''(c)$ := CompleteWithNacs($P'(c)$)
    $P(c)$ := UpdateAttributes($P''(c)$)
```

`FromBoundaryToContext(c:Constraint):GraphSet×MorphSet` constructs the uniquely determined (up to isomorphism) sets $\mathscr{Q} = \{Q_0, \ldots, Q_k\}$ and $\mathscr{H} = \{h_i^j : Q_i \to Q_j\}$ of all and only the graphs and associated morphisms s.t. the following are jointly satisfied:

- $Q_0 = X'$ and $Q_k = Q$;
- for each $i = 1, \dots, k$, there exist inclusions[4]; $h^i_{X'} \colon X' \to Q_i$ and $h^Q_i \colon Q_i \to Q$;
- for each $i = 1, \dots, k$, for $j > i$ s.t. $h^j_i : Q_i \to Q_j$ exists, $h^j_{X'} = h^j_i \circ h^i_{X'}$ and $h^Q_i = h^Q_j \circ h^j_i$;
- Q is the colimit of the diagram obtained from graphs in $\mathscr{Q}$ and morphisms in $\mathscr{H}$;
- given $h^j_i \colon Q_i \to Q_j$ we have that: $e_a \in E_A(Q_j) \setminus E_A(Q_i) \implies s_A(e_a) \in V_G(Q_j) \setminus V_G(Q_i)$[5].

`FromMorphismsToRules(`$\mathscr{Q}$`:GraphSet,`$\mathscr{H}$`:MorphSet):SetOfRules` produces the intermediate rule set $P_Q(c) = \{p(h^j_i) : Q_i \overset{l}{\leftarrow} Q_i \overset{r}{\rightarrow} Q_j \,|\, r = h^j_i : Q_i \to Q_j \in \mathscr{H}\}$, under the conditions:

1. given $Q_i, Q_j, Q_l \in \mathscr{Q}$ and $h^j_i, h^l_i, h^l_j \in \mathscr{H}$, we have $h^l_i = h^l_j \circ h^j_i \implies p(h^l_i) \notin P_Q(c)$;
2. either $Q_j = Q_k$ or $E_G(Q_j) \neq E_G(Q)$, i.e. the structural part of the right hand-side is equal to the structural part of the context Q only in rules which add some structural edge.

`CompleteToConclusion(int:SetOfRules,conc:Graph):SetOfRules` builds the set $P'(c)$ by generating the set IQY of all the graphs Z s.t. inclusions $k_1 \colon Q \to Z$ and $k_2 \colon Z \to Y$ exist, but no inclusion $k_3 \colon X \to Z$ exists. The set $HIQY$ of morphisms $h^Z_Q \colon Q \to Z$, $h^Y_Z \colon Z \to Y$, s.t. Y is the colimit of the diagram formed with nodes in $IQY \cup \{Q\}$ and morphisms in $HIQY \cup \{y \colon Q \to Y\}$, is concurrently built as before. In particular, as for each $e \in E(Y) \setminus E(c(X))$ we have $e \in E(Q)$, graphs in $IQY \setminus \{Y\}$ can only differ from Q by: (i) structural nodes in $V(X)$ not connected with the secured node in Y; (ii) structural edges attached to such nodes; and (iii) attribute edges already present in X (hence neither in X' nor in Q).

The structural parts of rules in a new set $P_Y(c)$ is built through the same construction as in `FromMorphismsToRules` for cases (i) and (ii). For case (iii), for any attribute edge $e_a \in E_A(W)$ s.t. $W \in IQY \wedge \nexists e'_a \in Q$, with $h^W_Q(e'_a) = e_a$, we produce a primed version Q'_i for each graph $Q_i \in \mathscr{Q}$ s.t. $\exists v \in V(Q_i)$, with $h^W_Q(h^Q_i(v)) = s(e_a) \in V(W)$, i.e. we have $e'_a \in E(Q'_i)$ with $s(e'_a) = v$ and $t(e'_a) = t(e_a)$. As a consequence, Q' is still the colimit of the diagram for the primed versions of the graphs in $\mathscr{Q}$. We then correspondingly update the rules with the attribute parts for all the rules derived from morphisms involving Q_i. The function then returns the union of the modified versions of $P_Y(c)$ and $P_Q(c)$, forming the set $P'(c)$.

The rules in $P'(c)$ are underspecified. Given morphisms h^j_i and h^i_l, both rules derived from them in $P'(c)$ are applicable on any match for h^j_i. `CompleteWithNacs(rls:SetOfRules):SetOfRules` constructs a NAC forcing the application of the more specific rule. Given $p_i : Q_l \overset{l}{\leftarrow} Q_l \overset{r}{\rightarrow} Q_i \in P'(c)$, we add a NAC $Q_i \overset{n_i}{\leftarrow} Q_l$ for each h^i_l. Moreover, NACs are added to prevent the formation of the premise X in rules with $p : L \overset{l}{\leftarrow} L \overset{r}{\rightarrow} R \neq Y$. In particular, we define the graph W s.t.

$$\begin{tikzcd} Y \arrow[rrr, bend left] & X' \arrow[l] \arrow[r] & L \arrow[r] & W \end{tikzcd}$$

is a pushout and add the NAC $W \overset{n}{\leftarrow} L$ to p.

Finally, `UpdateAttributes(rls:SetOfRules):SetOfRules` takes care of attribute update for the secured node, producing the final set $P(c)$. This is needed for the case where the

[4] In order to emphasize $X' = Q_0$ and $Q = Q_k$ we abuse notation, writing h^Q_i for $h^k_i : Q_i \to Q$ and $h^i_{X'}$ for $h^i_0 : X' \to Q_i$.

[5] As value nodes are assumed to always exist, we leave them implicit when not associated with structural elements.

constraint concerns the assignment of a value $\overline{d}$ for an attribute of the secured node $\overline{v}$, i.e. $\exists e_a \in E_A(X)$ with $s(e_a) = \overline{v}$ and $t(e_a) = \overline{d}$. Any rule $p: Z \overset{=}{\leftarrow} Z \overset{r}{\rightarrow} Y$ is such that $e_a \in E_A(Y) \setminus E_A(r(Z))$, but $\overline{v} \in V_G(Z)$. However, for a host graph G and a match $m\colon Z \rightarrow G$, there can be an edge $e'_a \in E(G)$ with $s(e'_a) = m(\overline{v})$ and $t(e_a) = d$, for some $d \in VD_a$, $d \neq \overline{d}$, where VD_a is the set of values of sort a in the domain D. The application of p would then produce two values for the same attribute. To avoid this problem, we transform any such rule into a version with attribute update $Z^d \overset{l}{\leftarrow} Z \overset{r}{\rightarrow} Y$, for each $d \in VD_a$. In particular, Z^d contains an edge e_a with $s(e_a) = \overline{v}$ and $t(e_a) = d$, so that the overall effect of the rule is to produce an occurrence of Y while updating the previous attribute assignment for $\overline{v}$. We call rules with $R = Y$ *final*.

Following Theorem 1, given a constraint c the resulting set $P(c)$ is consistent with c.

Theorem 1 *Given a constraint $c: X \rightarrow Y$, a graph G with $G \models c$, and a rule $p: L = K \rightarrow R \in P(c)$, for any graph H s.t. $G \Rightarrow_p H$ we have $H \models c$.*

Proof. If $G \models c$, then one of two cases occur.

1. $\exists m\colon X \rightarrow G, g\colon Y \rightarrow G$, with $m = g \circ c$. As all rules are increasing and c is positive, each application of p preserves m and g. We are left to prove that no further match $m'\colon X \rightarrow G$ is created for which a match $g'\colon Y \rightarrow G$ with $m' = g' \circ c$ does not exist. We first observe that an additional match would require the addition of an edge (either structural or attribute) e to G, with $s(e) = v$, $v = m'(\overline{v})$. But, due to the construction in `CompleteToConclusion` and the NACs added in `CompleteWithNacs`, this only happens in a final rule, which would also generate the required part to complete the match g'.

2. $\nexists m\colon X \rightarrow G$. By the same argument as before, we observe that any non final rule would not create such a match, while a final rule would also create the required match $g\colon Y \rightarrow G$.

No other case can occur and this concludes the proof. □

To maintain consistency over the whole policy, a rule $p: L \overset{=}{\leftarrow} L \overset{r}{\rightarrow} R \in P(c)$ should be checked against any constraint $c'\colon X' \rightarrow Y'$, with $c' \neq c$, for situations where applying p would cause the formation of X' without ensuring the formation of Y'. Figure 10 shows the possible solutions (for reasons of space we omit the morphisms $L \overset{=}{\leftarrow} L$). The first (left) generates a NAC where W' is a graph s.t. the span $R \overset{w'}{\leftarrow} W' \overset{w}{\rightarrow} X'$ exists and $R \overset{r_z}{\rightarrow} Z' \leftarrow X'$ is the cospan s.t. the resulting square is a pushout. Then the NAC $n\colon L \rightarrow N$ is generated, with N the pushout complement of $L \overset{r}{\rightarrow} R \overset{r_z}{\rightarrow} Z'$. This construction can be applied both for negative and positive constraints.

Alternatively, in Figure 10 (center), one extends L and R by adding to both of them the “missing part” of Y'. The morphisms $L \overset{r}{\rightarrow} R$ and $W' \overset{w'}{\rightarrow} R$ are jointly surjective on R, the triangle commutes, and all squares are pushouts. L' is constructed as the pushout complement for $L \overset{r}{\rightarrow} R \overset{r_w}{\rightarrow} R'$.

These constructions (adapted from those in [KP06]) can be applied if the required pushout complements exist. Otherwise, Figure 10 (right) presents an alternative construction generating a more restrictive NAC, with $n = r_z \circ r$. A negative constraint can only generate a NAC.

A word on complexity is in order. The upper bound for $\mid \mathcal{Q} \mid$ is $\mid Q \mid_E \times 2^{(|Q|_V - |X'|_V)}$ (reached when all node types are different). However, given the form of the constraints we are consider-

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "n"] & R \arrow[d, "r_z"] & W' \arrow[l, "w'"'] \arrow[d, "w"] & \\
N \arrow[r] & Z' & X' \arrow[l] \arrow[r, "c'"] & Y'
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[r, "r"] \arrow[d] & R \arrow[d, "r_w"] & W' \arrow[l, "w'"'] \arrow[d, "w"] \arrow[r] & X' \arrow[dl, "c'"] \\
L' \arrow[r, "r'"] & R' & Y' \arrow[l] &
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[r, "r"] \arrow[dr, "n"] & R \arrow[d, "r_z"] & W' \arrow[l] \arrow[d, "w"] & \\
& Z' & X' \arrow[l] \arrow[r, "c'"] & Y'
\end{tikzcd}$$

Figure 10: Permissive NAC (left), rule extension (centre) and restrictive NAC (right) for c'.

ing, that security policies are usually not exceedingly large in their structural size, and that the procedure has to be computed once, the approach remains feasible in many practical situations.

Example Figure 11 shows the construction of the boundary and context graphs for $c_1 : X_1 \to Y_1$ from Figure 4, while Figure 12 presents the construction of the sets $\mathscr{Q}$ and $\mathscr{H}$ for this case. We have indicated only direct morphisms, and omitted those which can be derived by transitivity.

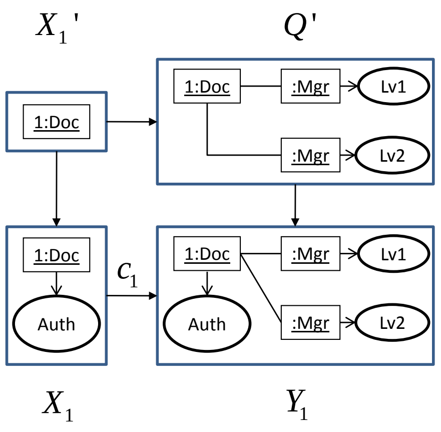


Figure 11: The construction of the context for $c_1 : X_1 \to Y_1$ in Figure 4.

The top of Figure 13 shows how the last rule derived from morphism h_6^Q would be modified. As the whole context for $c : X \to Y$ is present, we can complete the right-hand side to be the full Y, i.e. presenting also an instance of X. Underneath this rule, we show one of its possible final versions, assuming that *preAuth* $\in$ **AuthState**. Such a rule would be generated only if such a state is compatible with the presence of the association of the document with a manager of Level 1, i.e. if the *preAuth* state is not associated with some constraint forbidding the presence of the association with such a manager. In the policy in our original scenario, the constraint $c_2 : X_2 \to Y_2$ from Figure 4 will have to be used to modify the rules $p(h_2^4)$ and $p(h_5^Q)$.

5.2 Repairing inconsistencies

The above procedure ensures that secured elements are treated consistently with respect to constraints, while the constructions described in the next Section can be used to modify rules in a way consistent with policy updates. However, it is also possible that after the adoption of a new policy, some occurrence of the secured element $\overline{v}$ is in an inconsistent state for a new constraint $c : X \to Y$, i.e. with G a graph describing the current situation of the organisation, we would have

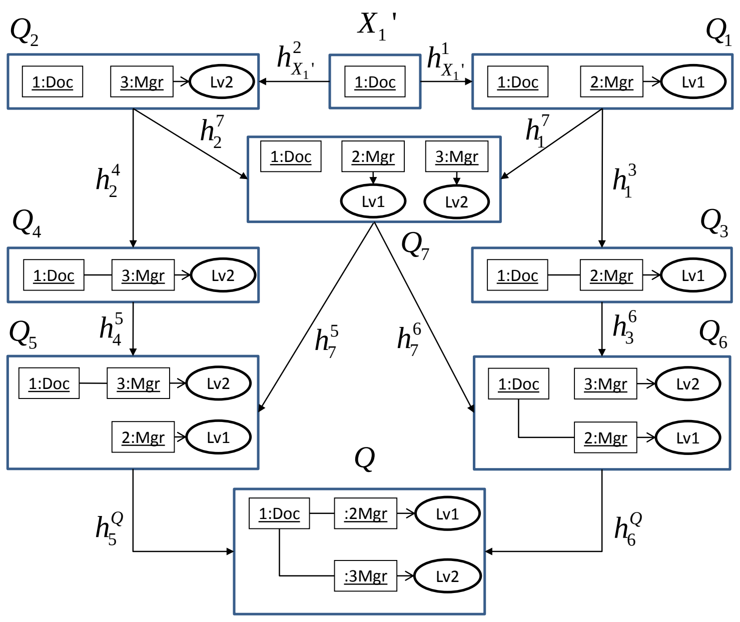


Figure 12: The sets $\mathscr{Q}$ and $\mathscr{H}$ in `ConstructRules(`c_1`).`

a match $x : X \to G$, but no match $y : Y \to G$ s.t. $x = y \circ c$. Considering Figure 4 (left), a document which was authorised after being reviewed only by a manager of level 1 would be inconsistent with the new policy. There are two possible ways to repair this inconsistency. The first is to complete the configuration, so that the missing review is provided. This is achievable by forcing the application of a sequence of rules, constructed in a similar manner to that of the procedure above, to produce an instance of the conclusion for the existing instance of the premise.

An alternative method destroys all the occurrences of the premise, so that a securing process can now be applied according to the new policy. Given a positive constraint $c : X \to Y$, and depending on the form of the premise X, two possible types of modification can be identified.

1. If $\overline{v}$ is associated in X with some attribute, the rule $X \xleftarrow{l} X' \xrightarrow{r} X'$ (with X' boundary object for the initial pushout over c) removes all the associations of this element with its attributes.

2. If the secured element has a set S of structural edges incident with it in X, generate the collection of edge-removing rules $R_S = \{X \xleftarrow{l} X' \xrightarrow{r} X' \mid E(X') \subsetneq E(X)\}$.

Each rule constructed in either way can be associated with a NAC of the form $n : X \to Y$ and applied to G. If the NAC fails, then c is satisfied, as the occurrence of X is associated with an occurrence of Y and no modification is required. Otherwise, the rule is applied on a match for X, but its application invalidates this match. With reference to the construction of $P(c)$, we can observe that a match for some $Q_i \in \mathscr{Q}$ has now been generated, so that a consistent model can be restored by applying some rule in $P(c)$ with $L = Q_i$.

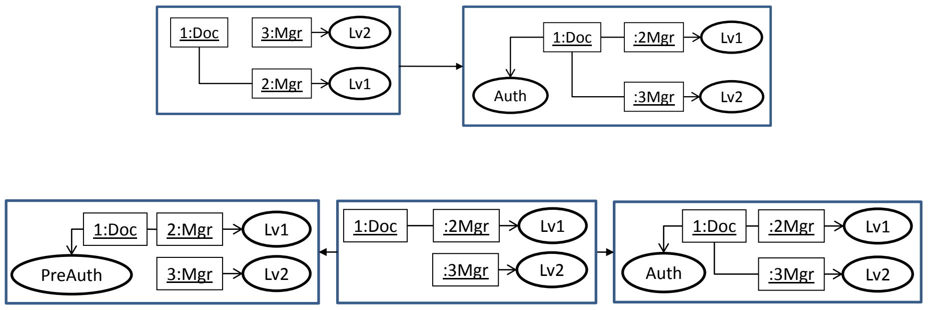


Figure 13: Ensuring constraint c_1 (top) and one version updating the state (bottom).

5.3 Rule-Constraint conflict

In this subsection, we complete our study by considering possible conflicts between decreasing rules and arbitrary atomic constraints (i.e. positive or negative).

While a deleting rule never violates a negative constraint, since it does not add forbidden graph elements, it may violate a positive constraint if the rule deletes conditionally required graph elements (parts of the conclusion) but preserves the premise of the constraint.

One way to resolve conflicts between rules p and constraints $c : X \to Y$ is by adding NACs to the rules p. We present next the construction of these NACs in a general case (generalising that in Figure 10) and then show how it is used to resolve conflicts.

Definition 1 (Conversion) Given a rule $p : L \overset{l_p}{\leftarrow} K \overset{r_p}{\rightarrow} R$ and a constraint $c : X \to Y$, let S be a nonempty overlap between R and the conclusion Y (preserving X) as in the diagram of Figure 14 (left). Let $D = R +_S Y$ be the pushout object of $s_1 : S \to R$ and $c \circ s_2 : S \to Y$, and let $D \overset{p^{-1},h}{\Rightarrow} N$ be the derivation with the inverse rule $p^{-1} : R \overset{r_p}{\leftarrow} K \overset{l_p}{\rightarrow} L$ with match h. Define $AC(p,c) = \{n\colon L \to N \mid D \overset{(p^{-1},h)}{\Rightarrow} N, D = R +_S Y \text{ for some overlap } S\}$. The rule p^c consists of the original rule $p : L \overset{l_p}{\leftarrow} K \overset{r_p}{\rightarrow} R$ together with the set $AC(p,c)$ of NACs, and is called the *conversion of p by c*.

$$\begin{tikzcd}
L \arrow[d, "n"] & K \arrow[l, "l_p"'] \arrow[r, "r_p"] \arrow[d] & R \arrow[d, "h"'] & S \arrow[l, "s_1"'] \arrow[r, "s_2"] & X \arrow[d, "c"] \\
N & M \arrow[l] \arrow[r, "r_p^*"] & C & & Y \arrow[ll]
\end{tikzcd}$$

$$\begin{tikzcd}
L \arrow[d, "m"] & R = K \arrow[l, "l_p"'] \arrow[d] & \\
G & H \arrow[l, "l_p^*"'] & X \arrow[l, "k"'] \arrow[d, "c"] \\
& & Y \arrow[llu, bend left, "q"] \arrow[lu, "q'"']
\end{tikzcd}$$

Figure 14: Construction for conflicts. NACs (left) and satisfaction by conversion (right).

The construction considers arbitrary rules and constraints, i.e., it is not restricted to deleting or expanding rules, and it reduces to the one in [HW95] if $c : X \rightarrow Y$ is the identity morphism.

This construction may generate redundant application conditions, in the case, for example, where the overlap $S \rightarrow R$ can be decomposed into $S \rightarrow K \rightarrow R$. The graph N generated from such an overlap can be eliminated by requiring only overlaps S for which $s_1(S) \cap (R \setminus r_p(K)) \neq \emptyset$.

Another form of redundancy stems from the fact that if S_1 and S_2 are overlaps with $S_1 \subseteq S_2$ and compatible morphisms s_1^i and s_2^i with $i = 1,2$, then $D_2 = R +_{S_2} Y \subseteq D_1 = R +_{S_1} Y$ and thus $N_2 \subseteq N_1$. Hence, if a match $L \rightarrow G$ satisfies (L, N_2), then it also satisfies (L, N_1) and the application condition (L, N_1) can be removed from $AC(p, c)$.

A negative constraint is a morphism $!c : X \rightarrow Y$, and $G \models !c$ *iff* no morphism $m : X \rightarrow G$ can be extended to a morphism $m' : Y \rightarrow G$ s.t. $m = m' \circ !c$. In the special case where $!c$ is the identity we obtain what we called forbidden graphs. There can be no real conflict between a deleting rule $p : L \overset{l_p}{\leftarrow} K = R$ and a negative constraint $!c : X \rightarrow Y$, since the deleting rule may remove parts of Y, in which case $!c$ is trivially satisfied, or parts of X, in which case $!c$ is vacuously satisfied.

For the potential conflict between deleting rules and positive constraints we can add NACs to prevent the rule from destroying the conclusion Y, by preventing the applicability if X is present and part of the conclusion Y is intended to be deleted by the rule.

Theorem 2 (Satisfaction by conversion) *Let $p : L \overset{l_p}{\leftarrow} K = R$ be a deleting rule and let $AC(id_L, id_Y))$ be the NACs constructed as in Definition 1 for the rule $id_L : L \rightarrow L$ with respect to the constraint $id_Y : Y \rightarrow Y$. Define $p^c = (p, A(id_L, c))$. If $G \models c : X \rightarrow Y$ and $G \overset{p^c}{\Rightarrow} H$ is a derivation with p^c, then $H \models c$.*

Proof. In Figure 14 (right), let $G \overset{p}{\Rightarrow} H$ via the matching morphism $m : L \rightarrow G$ and $k : X \rightarrow H$ a morphism. Since p is deleting, the derivation induces a morphism $l_p^* : H \rightarrow G$ and therefore a morphism $k' = l_p^* \circ k : X \rightarrow H \rightarrow G$. By the assumption that $G \models c$, there exists a morphism $q : Y \rightarrow G$ s.t. $q \circ c = k'$. To complete the proof we need to show that there is a $q' : Y \rightarrow H$ s.t. $k = q' \circ c$. Since l_p^* is injective, it is sufficient to show that $q(y) \in l_p^*(H)$ for every $y \in Y$. Suppose this is not the case. Then, $\exists y \in Y, l \in L \setminus l_p(R)$ s.t. $q(y) = m(l)$ and therefore this y defines an overlap of L and Y, contradicting the assumption that m satisfies the NAC $AC(id_L, c)$ of p^c. □

6 Incremental update

We analyse the problem of incrementally updating the rules in $P(c)$ when the original positive constraint $c : X \rightarrow Y$ is replaced by a new one, $c^u : X^u \rightarrow Y^u$, according to one of the following modifications, with inequalities indicating inclusion from the smallest to the biggest graph. We study modifications preserving the image of the original premise in the original conclusion for the common parts of X and X^u and of Y and Y^u, and in particular preserving the secured node.

1. **(conclusion expansion)** $X = X^u, Y < Y^u$
2. **(conclusion reduction)** $X = X^u, Y > Y^u$
3. **(premise expansion)** $X < X^u, Y = Y^u$

4. **(premise reduction)** $X > X^u$, $Y = Y^u$
5. **(semantic reduction)** $X > X^u$, $Y < Y^u$
6. **(semantic expansion)** $X < X^u$, $Y > Y^u$
7. **(common expansion)** $X < X^u$, $Y < Y^u$
8. **(common reduction)** $X > X^u$, $Y > Y^u$

We do not consider modifications which simultaneously add some elements and delete others from the premise or the conclusion of a constraint. Hence, if a premise is expanded, the conclusion is reduced only by removing parts neither in the original nor in the expanded premise. We consider the creation of new rules or updates of the existing L, R or NAC parts of the rules, while new NACs can be generated according to the constructions in the Section 5.

From Figure 15 (left), modeling premise expansion, we observe that if a graph G satisfies c non trivially, i.e. G has a subgraph which is a match for Y, it will also satisfy c^u under the same match. On the other hand, the only rules in $P(c)$ which can produce an occurrence of X^u are final rules (producing the corresponding original occurrence of X by placing an edge attached to the secured node), so no modification is required to them.

For the case of conclusion reduction, centre of Figure 15, we observe that we can only remove elements which are not in X. Hence, we can simply modify a rule $p : L \xrightarrow{l} R \in P(c)$ by removing all elements in $V(Y) \setminus V(Y^u) \cup E(Y) \setminus E(Y^u)$ from both L and R.

The combination of premise expansion and conclusion reduction defines what is called semantic expansion, as $M(c) \subset M(c^u)$. Indeed, considering the diagram on the right of Figure 15, we observe that for each graph G, $G \models c \implies G \models c^u$. On the other hand, we can have a graph G with $G \not\models c$ (i.e. a match $m\colon X \to G$ exists which cannot be extended to a match $y\colon Y \to G$) and $G \models c^u$, if no match $m^u : x^u \to G$ exists (i.e. G trivially satisfies c^u).

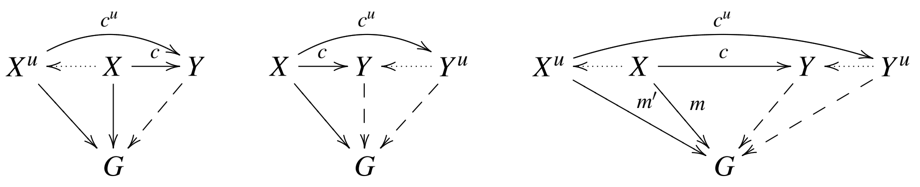

$$\begin{tikzcd}
X^u \arrow[dr] & X \arrow[l, dotted] \arrow[r, "c"] \arrow[d] \arrow[r, bend left=40, phantom] & Y \arrow[dl, dashed] \\
& G &
\end{tikzcd}
\quad
\begin{tikzcd}
X \arrow[r, "c"] \arrow[dr] \arrow[rr, bend left, "c^u"] & Y \arrow[d, dashed] & Y^u \arrow[l, dotted] \arrow[dl, dashed] \\
& G &
\end{tikzcd}
\quad
\begin{tikzcd}
X^u \arrow[drr, "m'"'] \arrow[rrrr, bend left, "c^u"] & X \arrow[l, dotted] \arrow[rr, "c"] \arrow[dr, "m"] & & Y \arrow[dl, dashed] & Y^u \arrow[l, dotted] \arrow[dll, dashed] \\
& & G & &
\end{tikzcd}$$

Figure 15: Premise expansion (left), conclusion reduction (centre), semantic expansion (right).

Figure 16 (left) presents premise reduction, where one notices that the evaluation of the initial pushout for c^u would give a smaller $X^{u\prime}$ and a bigger Q^u. Hence, one has to perform a construction similar to that of `FromBoundaryToContext` and `FromMorphismsToRules` to generate the rules creating the intermediate graphs between $X^{u\prime}$ and X'. Furthermore, for a rule $p : L \xleftarrow{l} L \xrightarrow{r} R \in P(c)$ with X^u included in L or R, a new rule p' is created with each such occurrence of X^u replaced by an occurrence of $X^{u\prime}$.

For conclusion expansion, Figure 16 (center), one notes that the construction of the initial pushout would now give a Q^u greater than Q. Hence, we have to construct the rules for the morphisms for graphs intermediate between Q and Q^u and between Q^u and Y^u, while removing those derived from graphs intermediate between Q and Y. These can be easily identified, as they present any of the elements in Y^u and not in Y, but they do not present at least one element in X.

Semantic reduction in Figure 16 (right) combines premise reduction and conclusion expansion. The relative modifications can be derived from the composition of the two processes.

$$\begin{tikzcd}
X^u \arrow[r, dotted] \arrow[rr, bend left, "c^u"] \arrow[dr] & X \arrow[r, "c"] \arrow[d] & Y \arrow[dl, dashed] \\
& G &
\end{tikzcd}
\quad
\begin{tikzcd}
X \arrow[r, "c"] \arrow[rr, bend left, "c^u"] \arrow[dr] & Y \arrow[r, dotted] \arrow[d, dashed] & Y^u \arrow[dl, dashed] \\
& G &
\end{tikzcd}
\quad
\begin{tikzcd}
X^u \arrow[r, dotted] \arrow[rrr, bend left, "c^u"] \arrow[drr] & X \arrow[r, "c"] \arrow[dr] & Y \arrow[r, dotted] \arrow[d, dashed] & Y^u \arrow[dl, dashed] \\
& & G &
\end{tikzcd}$$

Figure 16: Premise reduction (left), conclusion expansion (centre), semantic reduction (right).

In a similar way, common expansion and common reduction can be obtained by combining the above processes, in the following order:

- For common expansion, expand Y to Y^u first, and then expand X to X^u, in both cases expanding the morphism so as to preserve the image of X in Y.
- For common reduction, reduce X to X^u first, and then reduce Y to Y^u, in both cases reducing the morphism so as to preserve the image of X in Y.

One wonders whether a similar construction can be defined when simultaneously removing and including elements in both premise and conclusion (while preserving the secured node). Figure 17 illustrates the problem with this. In order to maintain incrementality, one needs to first identify the intersections X' and Y' between the original and the updated versions, update under common reduction for $X' \rightarrow Y'$, and then use common expansion to generate $P(c^u)$ for $c^u : X^u \rightarrow Y^u$ (left). Alternatively, one can first use common expansions and then common reduction, where the intersections are X'' and Y'' (right). But in this case there is no guarantee that the nodes which receive the additional edges in Y and Y' are the same. Hence, depending on the order in which the transformations are applied, one can obtain different rule sets.

$$\begin{tikzcd}
& X' \arrow[r] \arrow[dl] \arrow[d] & Y' \arrow[d] \arrow[dr] & \\
X^u \arrow[rrr, bend right, "c^u"'] & X \arrow[r, "c"] & Y & Y^u
\end{tikzcd}
\qquad
\begin{tikzcd}
X' & X \arrow[l] \arrow[r, "c"] & Y \arrow[r] & Y' \\
X^u \arrow[u] \arrow[rrr, bend right, "c^u"'] & X'' \arrow[l] \arrow[r] \arrow[u] & Y'' \arrow[r] \arrow[u] & Y^u \arrow[u]
\end{tikzcd}$$

Figure 17: Incorrect constructions for simultaneous removal and increment.

Example Considering the scenario in Section 4, Figure 8 presents a case of conclusion expansion, making stricter requests on the authorisation process. Following the algorithm above, the construction of the new rules for this policy starts with the identification of the rules in $L \in \mathscr{L}_{Y_1}$; in this case rule $r_2 : L_2 \rightarrow R_2$ in Figure 7 (omitting the study of the modification of NACs). We then need to replace r_2 with the set of rules constructing the intermediate graphs between L_2 and the modified conclusion Y_6, with the proviso that X can appear only in the right-hand side of the rule building the whole Y_6. Figure 18 shows the process replacing rule r_2 with rules r_2' and r_2^2.

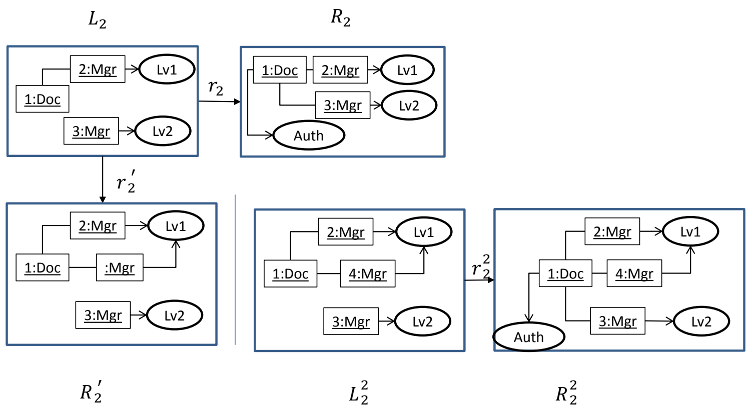


Figure 18: Revising rule r_2 from the scenario.

7 Conclusions and future work

We have presented an approach to the construction of incremental procedures ensuring the application of rules consistently with atomic constraints and to the modification of such procedures in an incremental way after the modification of the original constraints. The approach works under a number of assumptions, which are reasonably met in practical cases. First, conclusions in constraints are defined by graphs with a specific relation with the premise, viz. only one node can receive additional edges. Second, constraint updates preserve the images for the preserved elements in the premise (i.e. we do not deal with revocation [HJPW01]). Third, constraints and rules are typed according to a metamodel maintaining distinct finite domains for each attribute and allowing only equality comparison operations among values. The procedures are defined as sets of rules, which can be exploited in different contexts, or organised into transformation units, typically consistent with the partial order induced by injections on structural parts of graphs.

The study of more general cases is a complex one. We prevent rules from creating matches for X if they are not final rules. However, if we relax the form of the constraints to consider secured configurations instead of secured nodes, situations may arise in which matches can be created other than just the one for which the extension to Y is provided. In this case more complex repair actions must be defined. Also, one needs to consider what happens when arbitrary graph morphisms are considered instead of only injective ones. This problem can be investigated in versions where arbitrary morphisms can be used for matches of constraints, of rules, or both. A final line of investigation concerns the possibility of lifting the constructions in the paper to high level structures. While we basically rely on pushouts (in particular the initial pushout) and colimits for the construction of the incremental procedure, we require a form for constraints which is related to their setting in the category of graphs. The study of a more general type of constraints could then give indications on how to generalise the approach.

Acknowledgments. Partially funded by Progetto Sapienza 2007: ”Sistemi interattivi per la fruizione e la produzione di eventi culturali”.

Bibliography

[BFP10] P. Bottoni, A. Fish, F. Parisi-Presicce. Preserving constraints in horizontal model transformations. *ECEASST* 29, 2010.

[EEHP06] H. Ehrig, K. Ehrig, A. Habel, K.-H. Pennemann. Theory of Constraints and Application Conditions: From Graphs to High-Level Structures. *Fundam. Inform.* 74(1):135–166, 2006.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[EKTW06] K. Ehrig, J. M. Küster, G. Taentzer, J. Winkelmann. Generating Instance Models from Meta Models. In *Proc. FMOODS 2006*. LNCS 4037, pp. 156–170. Springer, 2006.

[HJPW01] Å. Hagström, S. Jajodia, F. Parisi-Presicce, D. Wijesekera. Revocations-A Classification. In *CSFW*. Pp. 44–58. 2001.

[HP09] A. Habel, K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science* 19(2):245–296, 2009.

[HW95] R. Heckel, A. Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. *ENTCS* 2:118–126, 1995.

[JKW08] M. Janota, V. Kuzina, A. Wasowski. Model Construction with External Constraints: An Interactive Journey from Semantics to Syntax. In *Proc. MoDELS 2008*. Pp. 431–445. 2008.

[KMP05] M. Koch, L. V. Mancini, F. Parisi-Presicce. Graph-based specification of access control policies. *J. Comput. Syst. Sci.* 71(1):1–33, 2005.

[KP06] M. Koch, F. Parisi-Presicce. UML specification of access control policies and their formal verification. *Software and System Modeling* 5(4):429–447, 2006.

[OL10] F. Orejas, L. Lambers. Symbolic Attributed Graphs for Attributed Graph Transformation. *ECEASST* 30, 2010.

[Ren04] A. Rensink. Representing First-Order Logic Using Graphs. In *Proc. ICGT*. LNCS 3256, pp. 319–335. Springer, 2004.

[WWP08] Y. Wei, C. Wang, W. Peng. Graph Transformations for the Specification of Access Control in Workflow. In *Proc. WiCOM ’08*. Pp. 1–5. 2008.