Encoding Incremental NACs in Safe Graph Grammars using Complementation

Andrea Corradini

Dipartimento di Informatica, University of Pisa, Italy

andrea@di.unipi.it

Maryam Ghaffari Saadat   Reiko Heckel

Department of Informatics, University of Leicester, UK

{mgs17,rh122}@leicester.ac.uk

In modelling complex systems with graph grammars (GGs), it is convenient to restrict the application of rules using attribute constraints and negative application conditions (NACs). However, having both attributes and NACs in GGs renders the behavioural analysis (e.g. unfolding) of such systems more complicated. We address this issue by an approach to encode NACs using a complementation technique. We consider the correctness of our encoding under the assumption that the grammar is safe and NACs are incremental, and outline how this result can be extended to unsafe, attributed grammars.

1   Introduction

Motivation. Graph grammars (GGs) can model complex systems or networks where data transformations are tightly coupled with structural changes. A defining feature of complex systems is that they show emergent behaviour arising from the actions of decentralised, distributed, often autonomous agents individually described by means of simple rules [22, 17, 16]. One purpose of modelling such systems is to investigate this emergent behaviour. This requires semantic models and analysis techniques able to account for concurrency properties, such as the unfolding, which captures in a single structure the full behaviour of the system, and its partial approximations [3, 1].

It is often natural to impose conditions on the rules governing the actions of individual agents to restrict their application using both constraints on data attributes and negative application conditions (NACs). Although allowing for both attributes and NACs enhances the expressiveness of rules and facilitates a concise style of modelling, it renders semantics and behavioural analysis more complicated.

In previous work, authors of this paper have generalised the theory of unfolding from plain graph grammars separately to the case with NACs and to the attributed case [5, 13]. While in the plain case, conflicts and dependencies between transformation steps are based on how nodes and edges are used, created or deleted by these steps, e.g. if created by one step and used by another, in the case with NACs we have to account for additional sources of conflicts and dependencies, e.g. when a rule deletes part of a structure that inhibits the application of another due to a NAC. Also, in the attributed case, dependencies can be based on attributes updated by one step and then read by another. Hence both generalisations give rise to additional complexity both conceptually and in terms of the construction of unfolding.

Our ultimate goal is to provide a comprehensive semantics and analysis approach based on a theory of unfolding that supports both attribution and NACs. As a first contribution, we investigate here the possibility of encoding incremental NACs of a safe grammar (where, up to renaming, all reachable

graphs are subgraphs of the type graph) by exploiting structural properties of the graphs. We also sketch how this technique could be extended to unsafe grammars by exploiting attributes.

Related work. In [10] the authors translate graph grammars with NACs and attributes into event-B models, a state-based formalism based on first-order logic with set theory. The authors prove in two stages that the encoded model is semantically equivalent to the input grammar: 1) encoding NACs only, 2) extending the encoding to attributes. Their goal is to be able to use theorem proving instead of model checking to show behavioural correctness without exploring the state space. Even if our approach is different, our motivations overlap with theirs in attempting to reduce complexity of behavioural analysis.

Another example for transforming negative application conditions over attributed graphs into a semantics-preserving way is [20]. Here, the motivation is to simplify rather than encode the conditions, but the technique of transforming rules in order to ensure that certain constraints are preserved is quite similar to the construction presented later in Section 3.

Methodology. Our problem can be expressed generically as follows: Given a *conditional* graph grammar $\mathscr{CG}$, that is a graph grammar where rules may be equipped with NACs constraining their applicability, construct an *unconditional* grammar $DE(\mathscr{CG})$ that is equivalent to $\mathscr{CG}$ in both the derivations generated and their dependency relations. That means, if two consecutive steps in a derivation of $DE(\mathscr{CG})$ are sequentially dependent (cannot be swapped), the same should hold for the corresponding steps in the derivation over $\mathscr{CG}$, and vice versa.

To express relations between grammars we rely on a notion of grammar morphism that preserves the behaviour of the source grammar, inducing a mapping of derivations and, moreover, preserves the relation of sequential independence between steps within a derivation. Such a morphism is based on a relation between the type graphs of the grammars that allows, among other things, to drop or rename types, and a mapping of rules that permits to reduce rules by dropping context and NACs.

The transformation of a given conditional grammar $\mathscr{CG}$ to its encoding $DE(\mathscr{CG})$ without NACs consists of two steps. First, we define a grammar $E(\mathscr{CG})$ whose NACs are essentially the same as in $\mathscr{CG}$ but are also encoded in a suitable way in the graphs to be transformed. Grammar $E(\mathscr{CG})$ is related to the original grammar by a morphism $e : E(\mathscr{CG}) \to \mathscr{CG}$. It can be shown that the NACs of $E(\mathscr{CG})$ are redundant thanks to their encoding, thus we can remove them from $E(\mathscr{CG})$ obtaining a grammar $DE(\mathscr{CG})$, and we can show that there is a morphism $d : E(\mathscr{CG}) \to DE(\mathscr{CG})$ which is the identity on types, constraints, rules and input graph. Our goal then is to show that both grammar morphisms of the span $\mathscr{CG} \xleftarrow{e} E(\mathscr{CG}) \xrightarrow{d} DE(\mathscr{CG})$ not only *preserve* derivations and sequential independence, as every morphism, but also *reflect* them. In particular, this means that

1. the NACs dropped by d do not restrict the applicability of rules in $E(\mathscr{CG})$ nor create dependencies and conflicts that do not exist on the underlying transformations;

2. the encoding of $\mathscr{CG}$ in $E(\mathscr{CG})$ via e does not restrict the applicability of rules nor create new dependencies and conflicts.

Since preservation of independence implies reflection of causality and conflicts, it ensures that $DE(\mathscr{CG})$ and $\mathscr{CG}$ have not only the same sequential behaviour, but also that their computations have the same branching structure, and thus equivalent unfoldings.

Our approach for showing that morphisms d and e reflect derivations relies on showing that (1) the NACs are redundant in $E(\mathscr{CG})$ so dropping them in $DE(\mathscr{CG})$ does not change the behaviour and that (2) the structure resulting from the encoding of the NACs in $E(\mathscr{CG})$ does not restrict the behaviour more than it was already restricted by the NACs in $\mathscr{CG}$. Concerning independence, it is trivially reflected by d but for morphism e the property is at present only conjectured. Both arguments will rely on suitable

properties satisfied by the reachable graphs in grammar $E(\mathscr{CG})$. To express such properties formally we introduce a set of graph constraints and require that such constraints are invariants, that is, they are satisfied in the start graph and preserved by the application of rules, and hence hold for all reachable graphs. This will play an important role in verifying the correctness of the encoding.

Realisation. In particular, we propose an approach to encode *incremental* NACs (i.e. NACs that can only be created or destroyed in a certain order [7]) into additional rule context. The restriction to *incremental* NACs is motivated by the fact that only with this restriction the notion of sequential independence (and therefore of causality) among steps enjoys some expected properties, which guarantee that a concurrent semantics can be defined properly [6]. Our construction can be considered as a generalisation of the well-known technique of complementation for Elementary Net Systems [23], which transforms a system with possible situations of contact (where we try to place tokens on places that are already marked) into an equivalent one which is contact-free.

    We consider the encoding in detail for safe conditional graph grammars. This is a prerequisite to a semantic comparison between notions of unfolding with [5] and without NACs [2], all resulting in safe (i.e., occurrence) graph grammars. After the presentation of the relevant background in Section 2, the complementation construction is described in Section 3. The main results, showing the equivalence of the original grammar with the one obtained by encoding the NACs and then forgetting them are presented in Section 4. A simple safe grammar modeling a client-server architecture for online meetings is used to illustrate concepts and constructions. Then in Section 5 we discuss informally the general case of (unsafe) conditional attributed graph grammars and their encoding. Here we introduce the idea of using attributes as counters for NAC occurrences reminiscent of reference counters in garbage collection algorithms. We also rely on rules with multiobjects [16] to manage the creation and deletion of complemented edges, but consider such rules as rule schemata that can be instantiated to countable sets of basic rules. This is described with an example of a Token-Curated Registry (TCR), an architectural pattern for smart contracts. In Section 6 we summarise our contributions and outline future work.

2   Background: Conditional Graph Grammars and Their Morphisms

This section first summarizes the basic definitions of typed graph grammars [8] based on the DPO approach [11, 12] extended by negative application conditions (NACs) [14]. In the second part we introduce a quite general definition of conditional grammar morphisms and we prove that they preserve derivations and independence.

Graphs and typed graphs. Formally, a *(directed, unlabelled) graph* is a tuple $G = \langle N, E, s, t \rangle$, where N is a set of *nodes*, E is a set of *arcs*, and $s, t : E \to N$ are the *source* and *target* functions. A *(graph) morphism* $f : G \to G'$ is a pair of functions $f = \langle f_N : N \to N', f_E : E \to E' \rangle$ preserving sources and targets, i.e., such that $f_N \circ s = s' \circ f_E$ and $f_N \circ t = t' \circ f_E$. Morphism f is a *mono(morphism)* if both components are injective, and is an *iso(morphism)* if both are bijective.

    The category of graphs and graph morphisms is denoted by **Graphs**. Given a graph TG, called *type graph*, a *TG-typed (instance) graph* is a pair $\langle G, t_G \rangle$, where G is a graph and $t_G : G \to TG$ is a (typing) morphism. A morphism between typed graphs $f : \langle G_1, t_{G_1} \rangle \to \langle G_2, t_{G_2} \rangle$ is a graph morphisms $f : G_1 \to G_2$ consistent with the typing, i.e., such that $t_{G_1} = t_{G_2} \circ f$. The category of TG-typed graphs and typed graph morphisms is denoted by **Graphs**$_{TG}$.

Double-pushout rewriting. A *(graph transformation) rule* $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ is a span of monos in **Graphs**$_{TG}$. A *match* of rule p in a graph G is a mono $m : L \to G$. Given a match m of p in G, a *double-*

pushout (DPO) transformation $G \overset{p,m}{\Longrightarrow} H$ from G to H exists if we can construct a diagram such as (1) where both squares are pushouts in **Graphs**$_{TG}$.

Derived and ancestor rules. If two rules $p = (L \overset{l}{\leftarrow} K \overset{r}{\to} R)$ and $p' = (G \overset{g}{\leftarrow} D \overset{h}{\to} H)$ are related by a DPO diagram as in (1), then we say that *p' is derived from p* and *p is an ancestor of p'*.

$$
\begin{array}{ccccc}
L & \overset{l}{\leftarrow} & K & \overset{r}{\to} & R \\
{\scriptstyle m}\downarrow & & {\scriptstyle d}\downarrow & & \downarrow{\scriptstyle m^*} \\
G & \underset{g}{\leftarrow} & D & \underset{h}{\to} & H
\end{array} \qquad (1)
$$

Negative constraints and incrementality. The applicability of rules can be restricted by (negative) constraints. A *constraint* over a rule p is a mono $n : L \to N$. A match $m : L \to G$ *satisfies* n (written $m \models n$) if there is no mono $q : N \to G$ such that $q \circ n = m$ (see Diagram (2) (left)). If $\mathscr{N}$ is a set of constraints we write $m \models \mathscr{N}$ if $m \models n$ for each $n \in \mathscr{N}$. If n and n' are constraints over p we say that *n subsumes n'* (written $n \models n'$) if for every match $m : L \to G$, $m \models n$ implies $m \models n'$.

$$
\begin{array}{cc}
\begin{array}{ccc}
N & \overset{n}{\leftarrow} & L \\
{\scriptstyle q}\nwarrow\!\diagdown & & \downarrow{\scriptstyle m} \\
 & & G
\end{array}
&
\begin{array}{ccccc}
N & \overset{n}{\leftarrow} & & & L \\
\uparrow & & A \leftarrow & & B \\
N^- & \underset{n^-}{\leftarrow} & & & L^-
\end{array}
\end{array} \qquad (2)
$$

We will often refer, for a constraint $n : L \to N$, to its "negative items", that is, to the nodes and edges of N that are not in the image of L. The smallest subgraph of N containing those items will be denoted N^-, and is characterized as in Diagram (2) (right) by an initial pushout: The square made of morphisms n and $n^- : L^- \to N^-$ and of the vertical morphisms is an *initial pushout* over n if for each pushout square $N \leftarrow A \leftarrow B \to L \overset{n}{\to} N$ with $B \to L$ mono, there are unique morphisms $N^- \to A$ and $L^- \to B$ making the diagram commute. For a constraint n we will call n^- its *shape*, L^- its *border* and N^- its *body*. Note that a shape would be an iso iff n is an iso, but in this case the rule is never applicable and can be dropped. Thus we will safely assume that the shape of a constraint is not an iso. For a set of constraints $\mathscr{N}$ we denote by $sh(\mathscr{N})$ its set of shapes.

A constraint $n : L \to N$ is *incremental* if whenever it can be decomposed in two different ways, i.e. $L \overset{n}{\to} N = L \overset{a_1}{\to} X_1 \overset{b_1}{\to} N = L \overset{a_2}{\to} X_2 \overset{b_2}{\to} N$ where all morphisms are mono, then there exists either a morphism from X_1 to X_2 or one in the opposite direction making the two triangles commute. For untyped graphs it is possible to show that if n is incremental then its shape n^- can only be one of those shown in (3), where nodes are boxes, L^- is made of the black items, while the red "negative" items belong to $N^- \setminus L^-$. For TG-typed graphs, for every shape in (3) each distinct typing morphism $t_{N^-} : N^- \to TG$ determines a different incremental shape.

$$
\boxed{\begin{array}{c}\text{shape } IN: \\ \square \leftarrow\!\!\times\!\!\boxtimes\end{array}} \;
\boxed{\begin{array}{c}\text{shape } OUT: \\ \square \times\!\!\to\boxtimes\end{array}} \;
\boxed{\begin{array}{c}\text{shape } E: \\ \square \times\!\!\to\square\end{array}} \;
\boxed{\begin{array}{c}\text{shape } L: \\ \square \circlearrowleft\end{array}} \;
\boxed{\begin{array}{c}\text{shape } N: \\ \boxtimes\end{array}} \;
\boxed{\begin{array}{c}\text{shape } NL: \\ \boxtimes \circlearrowleft\end{array}} \qquad (3)
$$

Grammars and derivations. A *(typed) graph grammar (GG)* $\mathscr{G} = \langle TG, G_{in}, P, \pi \rangle$ consists of a type graph TG, a TG-typed input graph G_{in}, a set of rule names P and a function π assigning to each $p \in P$ a rule $\pi(p) = (L_p \leftarrow K_p \to R_p)$.

A *conditional graph grammar (CGG)* $\mathscr{CG} = \langle TG, G_{in}, P, \pi, \mathscr{N} \rangle$ adds to its *underlying graph grammar* $\langle TG, G_{in}, P, \pi \rangle$ a function $\mathscr{N}$ providing for each $p \in P$ a *negative application condition (NAC)* $\mathscr{N}(p)$ over $\pi(p)$, that is a set of constraints over $\pi(p)$. Given a rule[1] $p \in P$ and a match $m : L \to G$, there is

---

[1]For the sake of simplicity, we often identify a rule with its name, leaving the application of π implicit.

a *conditional transformation* $G \overset{p,m}{\Longrightarrow} H$ if the match $m : L \to G$ satisfies all the constraints in $\mathcal{N}(p)$ and a double-pushout diagram such as (1) can be constructed. Sometimes we will call the pair $\langle p, \mathcal{N}(p) \rangle$ a *conditional rule*.

A *derivation* in $\mathcal{G}$ is a finite sequence of transformations $s = (G_{in} = G_0 \overset{p_1,m_1}{\Longrightarrow} \cdots \overset{p_n,m_n}{\Longrightarrow} G_n)$ with $p_i \in P$. A *conditional derivation* in $\mathcal{CG}$ is defined similarly as a sequence of *conditional* transformations. A graph $H \in \mathbf{Graphs}_{TG}$ is *reachable* in a (C)GG if there exists a (conditional) derivation ending with H.

A (conditional) graph grammar is *safe* if all rules (including their constraints, if any) and all reachable graphs have an injective typing morphism to TG. A simple safe conditional grammar is presented in Example 2.1. In a safe grammar, since w.l.o.g. we can consider all typing morphisms as inclusions, we can assume that G_{in} and all reachable graphs are subgraphs of TG. Even if safe grammars enjoy quite a limited expressive power, certain variants of them (*occurrence* grammars) are exploited as semantic domain able to represent, through an *unfolding construction*, causality and independence among transitions of (conditional) grammars, as well as the branching structure of their computations [3, 5].
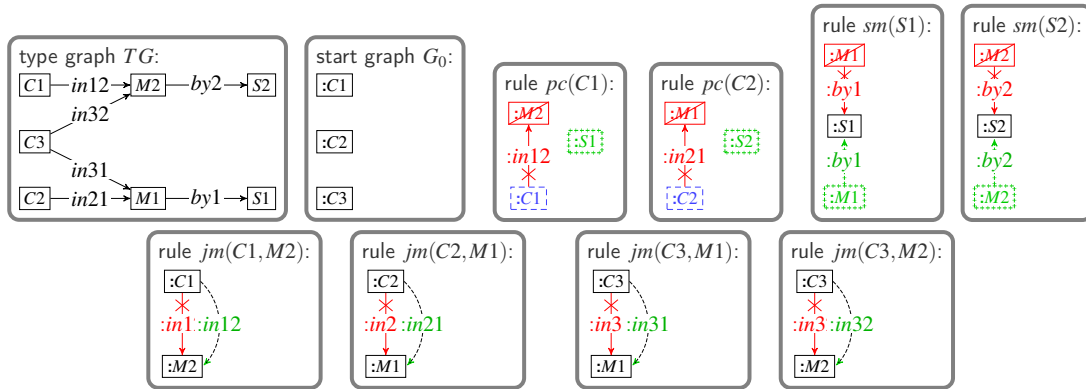
Example 2.1 (Client-Server). The figure above shows a safe grammar which depicts a simple Client-Server model. Rather than providing typing in the usual sense, in a safe grammar such as this one the type graph TG plays the role of a global name space for all graphs reachable from the start graph. This is common in models where we have an upper limit on the number of nodes and edges that may exist during the lifetime of the system, or where the grammar represents a semantic object, such as an occurrence grammar obtained by unfolding where rules represent transformation occurrences.

We use an integrated notation merging left- and right- hand side graphs into a single rule graph $L \cup R$. We indicate by colours and labels which elements are required and preserved (black), required but deleted (blue), newly created (green), and forbidden (red). That means, the left-hand side L is given by all black and blue elements with the forbidden elements of N in red, the right-hand side by all black and green ones, and the interface K by the black elements only.[2] A typical scenario is as follows: initially there are three clients, one for each type. Two of the clients can be promoted to a server if they are not attached to a meeting (using rules $pc(C1)$ and $pc(C2)$ respectively which have a constraint of shape OUT). A server can start a meeting if it doesn't already have one (using rules $sm(S1)$ and $sm(S2)$ respectively which have a constraint of shape IN). A client can join a meeting if it is not already in that meeting (using rules $jm(C1,M2)$, $jm(C2,M1)$, $jm(C3,M1)$, and $jm(C3,M2)$ respectively which have a constraint of shape E).

---

[2]This notation could be ambiguous if a rule has more than one NAC: in that case an additional explanation is needed.

$$
\begin{array}{c}
N_1 \qquad\qquad\qquad\qquad\qquad\qquad\qquad N_2 \qquad\qquad\qquad\qquad (4)\\[4pt]
\begin{array}{ccccccccccc}
& n_1\!\uparrow & & & & & & & n_2\!\uparrow & & \\
L_1 & \xleftarrow{\;l_1\;} & K_1 & \xrightarrow{\;r_1\;} & R_1 \,\text{-}\,\text{-}\,\text{-} & & \text{-}\,\text{-}\,\text{-}\,L_2 & \xleftarrow{\;l_2\;} & K_2 & \xrightarrow{\;r_2\;} & R_2 \\
m_1\!\downarrow & & n_1\!\downarrow & & & j & i & & n_2\!\downarrow & & m_2^*\!\downarrow \\
G & \xleftarrow{\;g_1\;} & D_1 & \xrightarrow{\;h_1\;} & H_1 & & H_1 \xleftarrow{\;g_2\;} & D_2 & \xrightarrow{\;h_2\;} & H_2 &
\end{array}
\end{array}
$$

Independence and causality among transformations. In the analysis of systems modeled as graph grammars, it is often important to identify when two consecutive (conditional) transformations of a derivation are *sequentially independent*, in the sense that their order can be switched obtaining an "equivalent" derivation, and when instead there is a causal relationship among them. For plain transformations, sequential independence of two consecutive transformations like those in Diagram (4) (ignoring constraints n_1 and n_2) is characterized (see e.g. [9]) by requiring the existence of two morphisms $i : R_1 \to D_2$ and $j : L_2 \to D_1$ such that $m_1^* = g_2 \circ i$ and $m_2 = h_1 \circ j$. For the conditional case [19] additionally it is required that $m_2' = g_1 \circ j \models n_2$ for each $n_2 \in \mathscr{N}(p_2)$, and that the match of p_1 in H_1' induced by the transformation $G \overset{p_2, m_2'}{\Longrightarrow} H_1'$ satisfies each $n_1 \in \mathscr{N}(p_1)$.

2.1   Morphisms of Conditional Grammars

The conditional grammar morphisms that we need in this paper are more general than, e.g., those of [5], because we want to relate a grammar where NACs are encoded by adding complemented items to the type graph of the original grammar. Thus retyping the graphs along the morphism should allow to delete part of the structure. This can be obtained by relating the type graph of the source and the target grammar with a span, as e.g. in [2]. Furthermore rules can be mapped to ancestor rules and constraints are reflected only up to subsumption.

A morphism $g : TG_0 \to TG_1$ induces a pullback functor $g^< : \mathbf{Graphs}_{TG_1} \to \mathbf{Graphs}_{TG_0}$ mapping an instance graph $\langle G_1, t_{G_1} \rangle$ over TG_1 to the graph $\langle G_0, t_{G_0} \rangle$ obtained as the pullback $TG_0 \xleftarrow{t_{G_0}} G_0 \to G_1$ of t_{G_1} and g. Dually, a morphism $h : TG_0 \to TG_2$ between type graphs induces a translation of instances obtained by the covariant retyping functor $h^> : \mathbf{Graphs}_{TG_0} \to \mathbf{Graphs}_{TG_2}$, defined by post-composition of h with the typing morphism. Note that this functor only affects the typing, hence $h^>$ acts as identity on morphisms. Combining both actions, given a span $f = (TG_1 \xleftarrow{g} TG_0 \xrightarrow{h} TG_2)$ relating TG_1 and TG_2 we define functor $f^{<>} : \mathbf{Graphs}_{TG_1} \to \mathbf{Graphs}_{TG_2}$ by the composition $h^> \circ g^<$.

Definition 2.2 (Conditional grammar morphisms). Let $\mathscr{CG}_i = \langle TG_i, G_{in,i}, P_i, \pi_i, \mathscr{N}_i \rangle$ for $i \in \{1,2\}$ be conditional graph grammars. A *CGG morphism* $f : \mathscr{CG}_1 \to \mathscr{CG}_2$ is a pair $\langle f_{TG}, f_P \rangle$ where $f_{TG} = (TG_1 \xleftarrow{f_<} TG_0 \xrightarrow{f_>} TG_2)$ is a span between type graphs TG_1 and TG_2 with $f_<$ mono, and $f_P : P_1 \to P_2$ is a mapping of rule names such that

1. The input graph is preserved (up to retyping): $f_{TG}^{<>}(G_{in,1}) = G_{in,2}$

2. Rules are mapped to ancestor rules: For all $p \in P_1$, if $f_P(p) = p'$ and $\pi_2(p') = \langle L' \leftarrow K' \to R' \rangle$, then there is a DPO diagram like (5) where the vertical morphisms are mono.

$$
\begin{array}{ccccc}
L' & \longleftarrow & K' & \longrightarrow & R' \\
i_p\!\downarrow & & \downarrow & & \downarrow \\
f_{TG}^{<>}(L_p) & \longleftarrow & f_{TG}^{<>}(K_p) & \longrightarrow & f_{TG}^{<>}(R_p)
\end{array}
\qquad (5)
$$

3. Constraints are reflected: For each rule $p \in P_1$ (with $f_P(p) = p'$) and each mono $h : L_p \to N$ over TG_1, if there is a constraint $n' \in \mathcal{N}_2(p')$ such that square (6) is a pushout, then there is a constraint $n \in \mathcal{N}_1(p)$ such that $n \models h$.

$$
\begin{array}{ccc}
L' & \xrightarrow{\;n'\;} & N' \\
{\scriptstyle i_p}\downarrow & & \downarrow \\
f_{TG}^{<>}(L_p) & \xrightarrow{\;f_{TG}^{<>}(h)\;} & f_{TG}^{<>}(N)
\end{array}
\tag{6}
$$

The definition of conditional grammar morphisms guarantees that morphisms preserve derivations: this will be pivotal for the results presented later.

Proposition 2.3. *Let* $\langle f_{TG}, f_P \rangle : \mathscr{CG}_1 \to \mathscr{CG}_2$ *be a CGG morphism. Then for each derivation* $G_{in,1} = G_0 \overset{p_1}{\Longrightarrow} \cdots \overset{p_n}{\Longrightarrow} G_n$ *of* $\mathscr{CG}_1$ *there is a derivation* $G_{in,2} = f_{TG}^{<>}(G_0) \overset{f_P(p_1)}{\Longrightarrow} \cdots \overset{f_P(p_n)}{\Longrightarrow} f_{TG}^{<>}(G_n)$ *of* $\mathscr{CG}_2$.

Proof. Let us denote by $f^< : \mathbf{Graphs}_{TG_1} \to \mathbf{Graphs}_{TG_0}$ the pullback functor induced by $f_< : TG_0 \to TG_1$, by $f^> : \mathbf{Graphs}_{TG_0} \to \mathbf{Graphs}_{TG_2}$ the functor obtained by post-composition with $f_> : TG_0 \to TG_2$, and by $\mathbf{f}$ their composition.

We show that if $G \overset{p,m}{\Longrightarrow} H$ is a conditional transformation in $\mathscr{CG}_1$ then $\mathbf{f}(G) \overset{p',\mathbf{f}(m)\circ i_p}{\Longrightarrow} \mathbf{f}(H)$ is a conditional transformation in $\mathscr{CG}_2$, where $p' = f_P(p)$ and $i_p : L_{p'} \to \mathbf{f}(L_p)$ is as in Diagram (5). This fact extends to arbitrary derivations by concatenation and by observing that $\mathbf{f}(G_{in,1}) = G_{in,2}$. For the DPO diagram of transformation $G \overset{p,m}{\Longrightarrow} H$ it is sufficient to observe that pushouts are preserved both by $f^<$, because $\mathbf{Graphs}$ is an adhesive category [18], and by functor $f^>$ because colimits are computed pointwise in slice categories. Therefore we obtain the required DPO diagram witnessing $\mathbf{f}(G) \overset{p',\mathbf{f}(m)\circ i_p}{\Longrightarrow} \mathbf{f}(H)$ by composing the DPO of Diagram (5) with the image via $\mathbf{f}$ of $G \overset{p,m}{\Longrightarrow} H$.

$$\tag{7}$$

It remains to show that match $\mathbf{f}(m) \circ i_p$ satisfies all the constraints in $\mathcal{N}_2(p')$, assuming that m satisfies all those in $\mathcal{N}_1(p)$. We proceed by contradiction, assuming that there is a constraint $n' : L_{p'} \to N' \in \mathcal{N}_2(p')$ not satisfied by $\mathbf{f}(m) \circ i_p$, i.e., there exists an injective morphism $q : N' \to \mathbf{f}(G)$ such that $q \circ n' = \mathbf{f}(m) \circ i_p$. In Diagram (7) we show the relevant graphs and morphisms in $\mathbf{Graphs}$, with the corresponding typing morphisms.

Since $q \circ n' = \mathbf{f}(m) \circ i_p$ commutes in $\mathbf{Graphs}_{TG_2}$, it also commutes in $\mathbf{Graphs}$; also, its typing to TG_2 factorizes through TG_0: for $\mathbf{f}(m)$ by the definition of functor $\mathbf{f}$, and for $L_{p'}$ and N' by composing q with the typing of $\mathbf{f}(G)$. Let squares ① and ② be built as pushouts in $\mathbf{Graphs}$. By the pushout property, since the relevant squares are easily shown to commute, there are unique morphisms $t_N : N \to TG_1$ (such that $L_p \to TG_1 = t_N \circ h$, showing that h is in $\mathbf{Graphs}_{TG_1}$) and $q' : N \to G$, such that (†) $m = q' \circ h$.

Recalling that $\mathbf{Graphs}$ is adhesive, observe that h is mono (because n' is mono and pushouts preserve monos) and that $\mathbf{f}(h) = x$ (up to an iso that we can safely ignore, because pushouts along monos are pullbacks). Since ① is a pushout, condition 3 of the definition of morphism applies, thus there is a constraint $n \in \mathcal{N}_1(p)$ such that $n \models h$. But (†) shows that $m \not\models h$ and thus $m \not\models n$, contradicting the assumption. $\qquad\square$

Recall that that sequential independence of conditional transformations is defined in terms of (1) existence of certain morphisms between given graphs and commutativity requirements, and (2) satisfaction of certain NACs by suitable matches. Properties (1) are easily shown to be preserved by any functor, and Proposition 2.3 guarantees that (2) is preserved by grammar morphisms. Therefore grammar morphisms preserve sequential independence.

Corollary 2.4. *Let $\langle f_{TG}, f_P \rangle : \mathscr{CG}_1 \to \mathscr{CG}_2$ be a CGG morphism. If $G_0 \overset{p_1}{\Longrightarrow} G_1 \overset{p_2}{\Longrightarrow} G_2$ are sequential independent transformations for $\mathscr{CG}_1$ then $f_{TG}^{<>}(G_0) \overset{f_P(p_1)}{\Longrightarrow} f_{TG}^{<>}(G_1) \overset{f_P(p_2)}{\Longrightarrow} f_{TG}^{<>}(G_2)$ are sequential independent transformations for $\mathscr{CG}_2$.*

3 Encoding NACs for Safe Conditional Grammars

Following the outline sketched in the Introduction, given a conditional grammar $\mathscr{CG}$ we define now a grammar $E(\mathscr{CG})$ where the NACs are encoded in the reachable graphs using suitable *complemented items*. Grammar $E(\mathscr{CG})$ keeps essentially the same NACs as $\mathscr{CG}$, but they will be shown to be redundant because of their encoding. For this section let $\mathscr{CG} = \langle TG, G_{in}, P, \pi, \mathscr{N} \rangle$ be a fixed *safe* conditional grammar, where all constraints are *incremental*. The first step consists of building an *enriched type graph* $\overline{TG}$ which extends TG by adding complemented items for each constraint: intuitively, the presence of such items in a graph guarantees that the constraint is satisfied.

Definition 3.1 (enriched type graph). Let $sh(\mathscr{CG}) = \{n^- : L_p^- \to N^- \mid p \in P, n \in \mathscr{N}(p)\}$ be the set of all shapes of constraints appearing in rules of $\mathscr{CG}$. Consider the diagram in **Graphs** made of all arrows of $sh(\mathscr{CG})$, of TG, and for each shape $n^- : L_p^- \to N^-$ of the typing morphism $t_{L_p^-} : L_p^- \to TG$. Then the *enriched type graph* $\overline{TG}$ is the colimit of this diagram.

Intuitively, the colimit adds to TG for each shape a copy of all the red items (see Diagram (3)), their *complement*, typed over $\overline{TG}$. Diagram (8) shows that the body N^- of shape n^- has two monos to $\overline{TG}$: the colimit injection $\bar{t}_{N^-}$ as well as $in_{TG} \circ t_{N^-}$. These monos are different (because the shape is not an iso by assumption) but coincide on the border L_p^-.

$$
\begin{array}{ccc}
N & \xleftarrow{\ n\ } & L_p \quad (8) \\
\end{array}
$$

They determine two different objects of $\mathbf{Graphs}_{\overline{TG}}$ that we shall denote $N^- (= \langle N^-, in_{TG} \circ t_{N^-} \rangle)$ and $\overline{N}^- (= \langle N^-, \bar{t}_{N^-} \rangle)$, respectively, calling the second *the body's complement*. Similarly, since the square in (8) is a pushout, there is a mediating morphism $\bar{t}_n$ from N to $\overline{TG}$ (which is a mono by adhesivity), besides the mono $in_{TG} \circ t_N$, giving rise to two objects in $\mathbf{Graphs}_{\overline{TG}}$ that we will denote $\overline{N}$ and N, respectively. We call $\bar{n} : L_p \to \overline{N}$ the *complemented constraint* of n.

We now introduce a property on objects of $\mathbf{Graphs}_{\overline{TG}}$, called the *complementation invariant*. Intuitively, a graph G over $\overline{TG}$ satisfies the invariant if for each constraint $n : L \to N$ of the original grammar, whenever the border of its shape (L^-) is present in G, then the body N^- of the shape is present as well if and only if its complement $\overline{N}^-$ is not present.

Definition 3.2 (complementation invariant). Let $sh(\mathscr{CG})$ and $\overline{TG}$ be as in Definition 3.1. Then, using a syntax reminiscent of *nested application conditions* [15], the *complementation invariant* is defined as
$$\Phi_{inv} = \forall (n^- : L^- \to N^-) \in sh(\mathscr{CG}) . \forall L^- . \exists (L^- \to N^-) \text{ XOR } \exists (L^- \to \overline{N}^-)$$

Given a graph G over TG we can enrich it with the needed complemented items in order to obtain a graph $inv(G)$ over $\overline{TG}$ that satisfies the invariant.

Definition 3.3 (invariant closure). Given a graph G typed over TG, its *invariant closure inv*(G) is the graph typed over $\overline{TG}$ obtained as the colimit of the diagram including (a) graph G typed over $\overline{TG}$ by $in_{TG} \circ t_G$, (b) for each shape $n^- : L^- \to N^- \in sh(\mathcal{CG})$, the complemented shape $n^- : L^- \to \overline{N}^-$ and the inclusion $L^- \to G$ if and only if the border L^- is contained in G but the body N^- is not.

It is easy to check that for each graph G over TG, $in_{TG}^{\leq}(inv(G)) \cong G$. Thus $inv(\_)$ is an inverse to the object component of the retyping functor along in_{TG}, but it is not itself a functor.

Example 3.4 (Complemented Client-Server). The grammar of the figure below is obtained by applying to the Client-Server grammar of Example 2.1 the transformations described in this section. The NACs are not represented (they are redundant, as we will see). The type graph $\overline{TG}$ and the start graph $\overline{G}_0 = inv(G_0)$ are obtained according to Defs. 3.1 and 3.3. The rules will be illustrated later on.
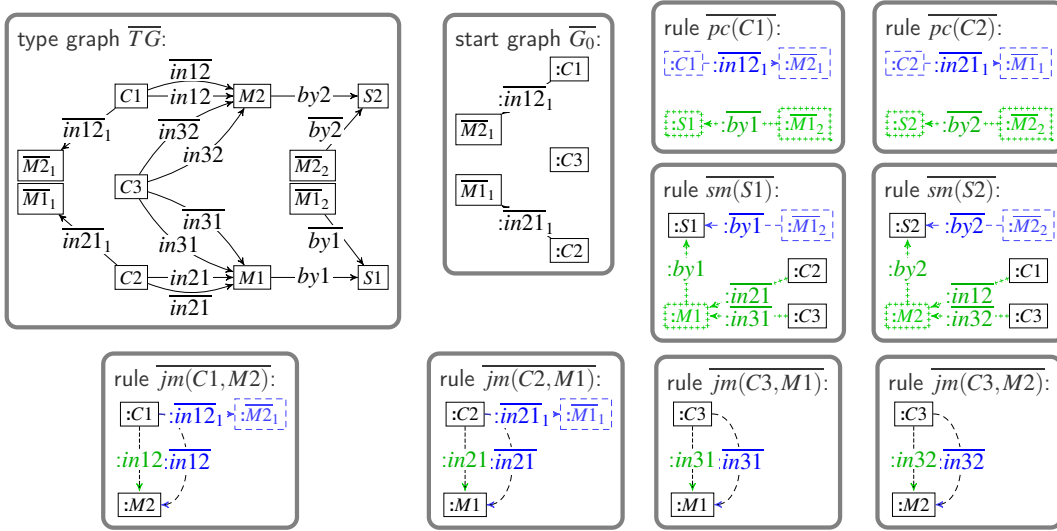
As examples, the invariants for the constraints of rules $pc(C1)$ and $jm(C1, M2)$ of Example 2.1 (which coincide with their shapes) are the following:

For example a graph satisfies the first invariant if, whenever it contains node :$C1$, then it contains an outgoing edge $in12$ to node :$M2$ if and only if it does not contain an outgoing edge $\overline{in12}_1$ to node :$\overline{M2}_1$.

We proceed now describing how the rules of $\mathcal{CG}$ have to be modified exploiting the complemented items in $\overline{TG}$ to encode the constraints in the reachable graphs. We first complement all rules, making them applicable only if the complement of the shape of each constraint is present, which means that the constraint cannot be violated. Next we enrich the resulting rules to ensure that the invariant is preserved.

Construction 3.5 (complemented rules). Given a conditional rule $\langle p, \mathcal{N}_p \rangle$ over TG its complementation is the conditional rule $compl(\langle p, \mathcal{N}_p \rangle)$ over $\overline{TG}$ returned by the following procedure.

1. Set $\tilde{p} = p$, $\mathcal{N}_{todo} = \mathcal{N}_p$, $\mathcal{N}_{done} = \emptyset$.

2. While $\mathcal{N}_{todo}$ is not empty, perform the following step:

    3. Let $n : L \to N \in \mathcal{N}_{todo}$ be a constraint for $\tilde{p} = (L \xleftarrow{l} K \to R)$. We assume that (†) the typing over $\overline{TG}$ of the body N^- of n's shape is $t_{N^-} ; in_{TG}$, i.e. it factorizes through TG. Then

(a) If there is a DPO diagram $\overline{N} \overset{\tilde{p},\tilde{n}}{\Longrightarrow} H$, where $\overline{n} : L \to \overline{N}$ is the complemented constraint, and there is no arrow $N^- \to R$, then set $\tilde{p}$ to be the derived rule (its bottom span). Otherwise, set $\tilde{p} = (\overline{N} \overset{\overline{n} \circ l}{\longleftarrow} K \to R)$.

(b) Set $\mathscr{N}_{todo} = \mathscr{N}_{todo} \setminus \{n\}$, and $\mathscr{N}_{done} = \mathscr{N}_{done} \cup \{n\}$.

(c) Lift each constraint in $\mathscr{N}_{todo}$ or $\mathscr{N}_{done}$ along $\overline{n}$, i.e., replace each constraint $n' : L \to N'$ with the morphism obtained by pushing out n' along $\overline{n} : L \to \overline{N}$.

4. Return $\langle \tilde{p}, \mathscr{N}_{done} \rangle$

Note that assumption (†) above is satisfied by the constraints of the starting rule p because it is typed over TG. Furthermore it is preserved by the modification of constraints in step *3.(c)* because the shape of a constraint does not change when pushing it out along a morphism, as it is defined as an initial pushout.

Note that Construction 3.5 was phrased exploiting pushouts in category **Graphs**$_{\overline{TG}}$. The assumption of safety would allow us to work directly in the category of subgraphs of $\overline{TG}$, using the standard set-theoretical operations of intersection and union of subgraphs (corresponding to pullback over the type graph, and to pushout over the pullback). We refrain from this and continue exploiting categorical constructions in view of generalizing the theory to the unsafe case.

We enrich now the complemented rules just obtained in order to guarantee the preservation of the complementation invariant of Definition 3.2. Recall that the invariant requires that whenever the border L^- of the shape of a constraint is present, then either its body N^- or its body complement $\overline{N}^-$ is present as well. For the shapes of constraints of a rule p, it is easy to check that such property is preserved, by construction, by the rule $compl(\langle p, \mathscr{N}_p \rangle)$ returned by Construction 3.5. With the following construction we replace each rule with one or more rules (all equivalent as far as the items typed over TG are concerned) which preserve the invariant also for the shapes of constraints of other rules.

The following four cases are possible for a rule p and a shape $n^- : L^- \to N^-$ not in $sh(\mathscr{N}_p)$: (1) If rule p deletes the body N^- preserving its border L^-, then p is enriched to create the corresponding body complement $\overline{N}^-$. (2) Dually, if p creates the body N^- preserving the border L^-, then it is extended to delete the body complement $\overline{N}^-$. (3) If p creates the border L^- but not the body N^-, then it is extended to create the body complement $\overline{N}^-$. (4) Finally, in the case of constraints of shape E, the border (which is made of two nodes, see Diagram (3)) can also be created or deleted only in part (just one node). If a rule creates only one node of the border L^- (i.e. the second node is not in R), then to preserve the invariant we need two rules: if the other node is already present in the current graph, we need to add the body complement, but if it is not present we don't need to add anything.

Construction 3.6 (making the rules invariant-preserving). Let $CP_{\mathscr{CG}} = \{compl(\langle p, \mathscr{N}(p) \rangle) \mid p \in P\}$ be the set of complemented rules of grammar $\mathscr{CG}$ typed over $\overline{TG}$. Also, for $p \in P$ let $sh(\mathscr{CG}_{\overline{p}})$ be the set of all shapes of constraints of rules different from p: as observed above, shapes are not affected by complementation.

For each conditional rule $\langle p = L \leftarrow K \to R, \mathscr{N}_p \rangle$ in $CP_{\mathscr{CG}}$ we obtain a set of *invariant preserving* conditional rules $IP(p)$ by applying in sequence the following transformations.

1. *[Compensate body deletion]* For each shape $n_i^- : L_i^- \to N_i^-$ in $sh(\mathscr{CG}_{\overline{p}})$ such that there is an arrow $N_i^- \to L$, check if in Diagram (9) (left) the top arrow of pullback (1) is not an isomorphism (the body N_i^- is deleted), but the top arrow of pullback (1) + (2) is an iso (the border L_i^- is preserved). In this case take the pushout (3) of arrows $L_i^- \cong Y \to K \to R$ and $Y \to \overline{N_i}^-$, and set $p' := L \leftarrow K \to R'$.

After all shapes are considered, continue to the next step with $\langle p', \mathcal{N}_p \rangle$.

$$
\begin{array}{ccc}
L_i^- \xleftarrow{\cong} Y \longrightarrow \overline{N}_i^- & X \xrightarrow{\not\equiv} L_i^- \longrightarrow \overline{N}_i^- & Y \xrightarrow{\not\equiv} X \xrightarrow{\not\equiv} L_i^- \longrightarrow \overline{N}_i^- \\
\end{array}
\tag{9}
$$

2. *[Compensate body creation]* This step can be formalized with a diagram symmetric to the previous one: if p creates the body N_i^- of a shape and preserves its border L_i^-, p is enriched in order to delete the body's complement by replacing L with the graph L' obtained as pushout of $L_i^- \to N_i^-$ and $L_i^- \to K \to L$. Also, all constraints have to be lifted along $L \to L'$ as in step *(c)* of Construction 3.5.

3. *[Complete border creation]* Continuing with the conditional rule $\langle p, \mathcal{N}_p \rangle$ resulting from the previous step, for each shape $n_i^- : L_i^- \to N_i^-$ in $sh(\mathscr{CG}_{\overline{p}})$ consider Diagram (9) (middle): if there is an arrow $L_i^- \to R$ and (4) is a pullback where the top morphism is not an iso (the rule completes the creation of the border), then build the pushout (5) and continue with $\langle L \leftarrow K \to R', \mathcal{N}_p \rangle$.

4. *[Complete partial border creation]* Let $\mathscr{R} = \{\langle p, \mathcal{N}_p \rangle\}$ be a set of conditional rules, initialized with the rule resulting from the previous step. For each shape n_i^- of type E in $sh(\mathscr{CG}_{\overline{p}})$, do the following for each rule $p = L \leftarrow K \to R$ in $\mathscr{R}$: Consider Diagram (9) (right). Graph X is the intersection of R and L_i^-, obtained as the pullback of the typing morphisms (because the grammar is safe). If it is not isomorphic to L_i^- and also the top morphism of pullback (6) is not an iso, then the rule creates part of the border, but not all of it. In this case add to $\mathscr{R}$ rule $\langle L' \leftarrow K' \to R'', \mathcal{N}' \rangle$, obtained as follows: build (7) and (8) as pushouts, (10) as pushout complement, (9) as pushout, while $\mathcal{N}'$ is obtained by lifting all the constraint of $\mathcal{N}$ along $L \to L'$.

Note that the last step could generate for each conditional rule a large set of derived rules needed to cover all the possible situations (presence or absence of nodes of E shapes). Whenever p has a match in a graph G, only the largest derived rule of the set having a match in G should be applied. This requirement of maximality of the match will be needed to guarantee correctness.

For example, rule $\overline{jm(C2,M1)}$ of Example 3.4 is obtained by first complementing rule $jm(C2,M1)$ of Example 2.1, which adds edge $\overline{in21}$ to L, and then applying step 2, because the rule creates $in21$ which is also the body of the shape of the constraint of rule $pc(C2)$. In Example 3.4 for each group of rules generated according to step 4 above only the maximal rule is depicted. For example, rule $\overline{sm(S1)}$ also has three subrules, obtained by deleting node $C2$, $C3$ or both from L, K and R, and the corresponding complemented edges to $M1$ from R. Such rules are obtained starting from rule $sm(S1)$ of Example 2.1, which generates a node, $M1$, belonging to the border of two shapes.

Exploiting the constructions just described, we can complete the definition of grammar $E(\mathscr{CG})$.

Definition 3.7 (the enriched grammar). Given $\mathscr{CG} = \langle TG, G_{in}, P, \pi, \mathcal{N} \rangle$, its enriched grammar is defined as $E(\mathscr{CG}) = \langle \overline{TG}, inv(G_{in}), P', \pi', \mathcal{N}' \rangle$ where $\overline{TG}$ is as for Defintion 3.1, $inv(G_{in})$ is as for Definition 3.3, and the conditional rules determined by P', π' and $\mathcal{N}'$ are obtained from those of $\mathscr{CG}$ by applying Constructions 3.5 and 3.6.

As desired, all the reachable graphs of grammar $E(\mathscr{CG})$ satisfy the invariant Φ_{inv}. In fact this is obvious for the start graph by construction, and each of the rules can be shown to preserve the invariant by a detailed analysis of Constructions 3.5 and 3.6, also exploiting the maximality requirement of matches mentioned above.

Fact 3.8 (the complementation invariant is satisfied). All the reachable graphs of conditional grammar $E(\mathscr{CG})$ satisfy the complementation invariant Φ_{inv}.

4 Equivalence of enriched and original grammars

All along this section let $\mathscr{CG} = \langle TG, G_{in}, P, \pi, \mathscr{N} \rangle$ be a grammar and $E(\mathscr{CG}) = \langle \overline{TG}, \overline{G}_{in}, P', \pi', \mathscr{N}' \rangle$ be the corresponding enriched grammar of Definition 3.7. We show that there is a grammar morphism from $E(\mathscr{CG})$ to $\mathscr{CG}$. Interestingly, this morphism not only preserves but also reflects derivations and, we conjecture, sequential independence.

Proposition 4.1. *Let* $e_{TG} : (\overline{TG} \overset{in_{TG}}{\leftarrow} TG \overset{id}{\rightarrow} TG)$ *be a span of type graphs, and* $e_P : P' \rightarrow P$ *be the mapping that associates each rule obtained from Construction 3.6 with the original rule in P. Then* $e = \langle e_{TG}, e_P \rangle :$ $E(\mathscr{CG}) \rightarrow \mathscr{CG}$ *is a well-defined morphism.*

Proof sketch. Note that the pullback functor $in_{TG}^{\leq}$ deletes from a graph all the complemented items, which are typed over $\overline{TG}$ but not over TG. The proof proceeds by analyzing all the transformations of the previous section showing that only complemented items are ever added (to the left- and right-hand sides of rules, to constraints and to the start graph). Only the transformation for shape E in step 4 of Construction 3.6 may add new non-complemented nodes to a rule p. Such nodes are preserved by p, and become isolated after retyping along in_{TG}. Therefore the retyped rule $e_{TG}^{<>}(p)$ is derived from the target rule $e_P(p)$, as desired.
Furthermore the NACs are reflected by e: this follows from the observation that the NACs of a rule p of $E(\mathscr{CG})$ can be obtained by lifting all and only the NACs of the original rule $e_P(p)$ along the morphism embedding the left-hand side $L_{e_P(p)}$ in L_p. $\square$

The next result essentially shows that the encoding of NACs is correct: the NACs in $E(\mathscr{CG})$ are redundant as they can never be violated by a match to a graph satisfying the invariant.

Lemma 4.2 (extended NACs are redundant). *Let* $p \in P'$ *be a rule of* $E(\mathscr{CG})$, $G \in \mathbf{Graph}_{\overline{TG}}$ *be a graph such that* $G \models \Phi_{inv}$, *and* $m : L_p \rightarrow G$ *be a match. Then* $m \models \mathscr{N}'(p)$.

Proof sketch. By Construction 3.5, each constraint $n : L_p \rightarrow N$ is lifted along $L_p \rightarrow \overline{N}$ obtaining an extended constraint $\overline{N} \rightarrow \tilde{N}$. Therefore $\tilde{N}$ contains both the body N^- of the shape n^- and its complement $\overline{N}^-$, violating the invariant Φ_{inv}, and this fact cannot be changed by other possible transformations of the constraint in Construction 3.6, because they can only extend it further. We can conclude that G cannot violate the constraint because there cannot be an injective morphism from a graph that doesn't satisfy Φ_{inv} to a graph that satisfies it, and G satisfies the invariant by assumption. $\square$

We now show that, as desired, morphism e also reflect derivations.

Theorem 4.3 (reflection of derivations). *The conditional grammar morphism* $e : E(\mathscr{CG}) \rightarrow \mathscr{CG}$ *reflects derivations: if* $G \overset{p,m}{\Longrightarrow} H$ *in* $\mathscr{CG}$, *then there are graphs* G' *and* H' *over* $\overline{TG}$, *a rule* $p' \in P'$ *and a match* $m' : L_{p'} \rightarrow G'$ *such that* $G' \overset{p',m'}{\Longrightarrow} H'$ *in* $E(\mathscr{CG})$, $e_{TG}^{<>}(G') \cong G$, $e_{TG}^{<>}(H') \cong H$, *and* $e_p(p') = p$.

Proof sketch. Let $G' = inv(G)$. Since $G \overset{p,m}{\Longrightarrow} H$ in $\mathscr{CG}$, it is possible to show that the rule obtained by complementing $\langle p, \mathscr{N}(p) \rangle$ according to Construction 3.5 has a match m' in G. In fact, all the constraints of p are satisfied by m by assumption, meaning that the corresponding shape bodies are not present in $inv(G)$, and thus by the invariant their complement bodies are present in $inv(G)$. The complemented rule

could have been transformed further by Construction 3.6: a case analysis shows that its left-hand side may only be extended, and in such case the match m' can be extended as well because the additional structure must be present in $inv(G)$ by the invariant. Finally if p has a corresponding family of enriched rules by step 4 of Construction 3.6 then the representative having the largest match in $inv(G)$ has to be chosen. If p' and m' are the rule and match found via this procedure, then $inv(G) \xRightarrow{p',m'} H'$ because (a) the applicability to the TG-typed part of $inv(G)$ is given by assumption; (b) the addition of complemented items in Constructions 3.5 and 3.6 does not introduce new constraints, and (c) the NACs of $E(\mathscr{CG})$ are redundant by Lemma 4.2. Finally, $e_{TG}^{<>}(inv(G)) \cong G$ by the properties of $inv(\_)$, $e_p(p') = p$ by construction, and $e_{TG}^{<>}(H') \cong H$ because e is a grammar morphism. $\qquad\square$

We conjecture that the two grammars are equivalent in an even stronger sense, because sequential independence is not only preserved by morphism e, but also reflected. This will be a topic of future investigation. To complete the picture sketched in the introduction, we formalize the idea of "dropping the redundant NACs" from $E(\mathscr{CG})$ with a grammar morphism, useful to guarantee that the resulting grammar is equivalent in the strong sense described above.

Proposition 4.4 (forgetting the NACs preserves and reflects derivations and independence)**.** *Let* $DE(\mathscr{CG})$ *be the conditional grammar obtained from* $E(\mathscr{CG})$ *by deleting all NACs, i.e., such that* $\mathscr{N}(p) = \emptyset$ *for each rule* p. *Let* $d : E(\mathscr{CG}) \to DE(\mathscr{CG})$ *be defined as* $d = \langle (\overline{TG} \xleftarrow{id} \overline{TG} \xrightarrow{id} \overline{TG}), id_{P'} \rangle$. *Then* d *is a well-defined morphism. Furthermore it reflects derivations and sequential independence.*

Proof sketch. Conditions 1 and 2 of grammar morphism hold trivially, and condition 3 holds vacuously because $DE(\mathscr{CG})$ has no NACs. Reflection of derivations holds for the double-pushout part trivially, and for the satisfaction of NACs because they are redundant in $E(\mathscr{CG})$. Reflection of sequential independence holds for the existence of certain morphims and commutativity requirements because retyping is along an iso, and for the additional conditions involving NACs again because they cannot be violated by graphs satisfying the invariant. $\qquad\square$

5   Encoding of NACs in Unsafe Attributed Graph Grammars

Many complex models require a formalism more expressive than safe grammars over typed graphs. In particular, safety limits instance graphs to be subgraphs of the type graph and hence, assuming this is finite, only allows us to model systems with a finite state space. Real-world applications also usually require that structural features, conveniently represented by nodes and edges in a graph, are augmented by data. This combination of structure and data leads us to attributed graph grammars.

In this section we will discuss the problem of encoding a conditional attributed graph grammar into an attributed graph grammar without NACs. The constructions proposed are applicable to all mainstream notions of attributed graph transformation, in particular those based on E-graphs [12] and ground symbolic attributed graphs [21].

As in the non-attributed case we work with typed graphs over a given type graph declaring, apart from node and edge types, also the attributes for nodes and edges and their respective domains given by the sorts of a given data algebra. The data algebra is fixed across all grammars, representing predefined basic data types such as strings and numbers.

Attributed graph transformation is defined based on the transformation of the underlying graph structure with the possibility of attribute constraints restricting possible matches and attribute assignments de-

termining the update of attribute values. In the following subsections we consider an example, describe the encoding for attributed condition grammars first in general and then apply it to the example.

## 5.1 Case Study: Token Curated Registry

In this section we show how our encoding applies to an unsafe grammar using a model of a Token-Curated Registry (TCR) based on [24]. TCR is an architectural pattern for smart contracts where a set of curators ensure the quality of the entries in a list of services or products. A typical scenario starts with a candidate applying to a registry to be listed, if they are not on the list yet. A curator of the registry can challenge a candidate if they are not being challenged at the time. Every curator of the registry hosting a challenge can vote provided that they are not the challenger and have not already voted.

For simplicity, we assume there is a single registry, and we have majority voting. That means, for the challenge *ch* to succeed, the number of voters in support of the challenge must be larger than half of the number of potential voters (all curators of the registry except the challenger), that is $maj(r,ch) := 1 \; if \; \left(ch.noVotes > \frac{r.noCurs - 1}{2} \right) \; else \; 0$. Note that by this definition in the event of a tie the challenged candidate is considered the winner.

Curators who supported the majority position receive a reward. In the start graph G_0, all reward attributes are initialised with zero. For all objects, the *rwds* attribute value must be non-negative at all times. We left this condition implicit here but it can easily be asserted for every rule that subtracts from *rwds* of an object.
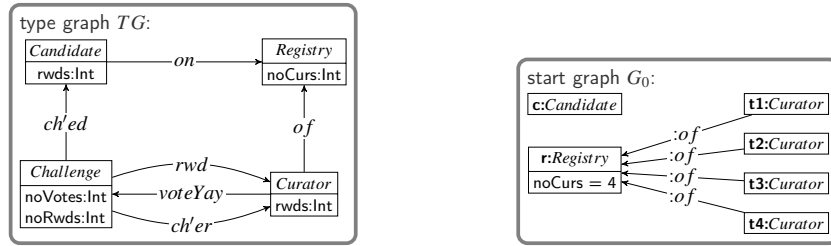
In a challenge, either the challenger (curator) wins or the challenged (candidate). In the first case the candidate is dropped from the list whereas in the second case it stays on. In any case the winner and curators who supported the winning party get a reward. In this example all constraints are incremental, thus if in a rule there is more than one red edge, each of them has to be considered as a distinct constraint.
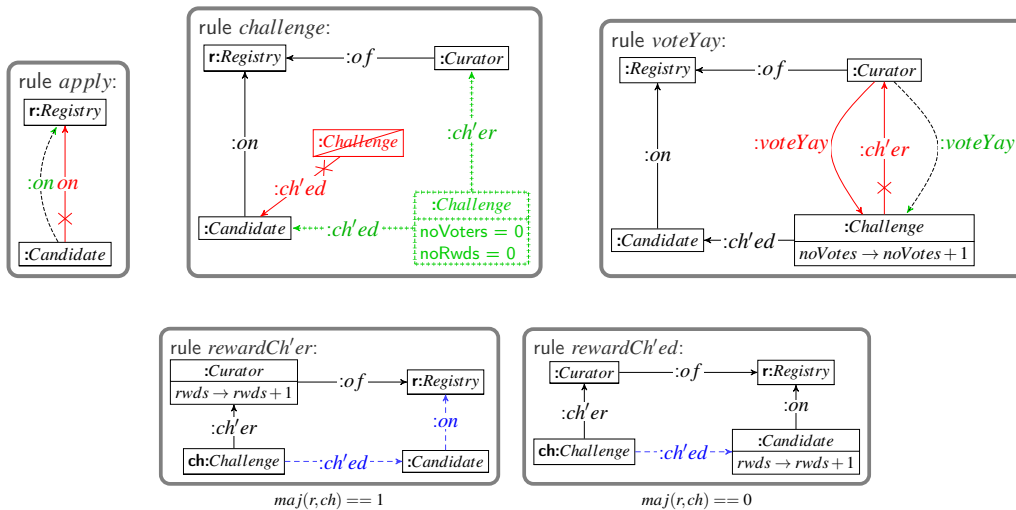
The *vote* rule is disabled once either the challenger or the challenged candidate is rewarded by the above rules since the link from the challenged candidate to the challenge is deleted at this stage. By deleting this link, the rules for rewarding the challenger or challenged also enable the rewards for curators who voted in favour or against the proposal.
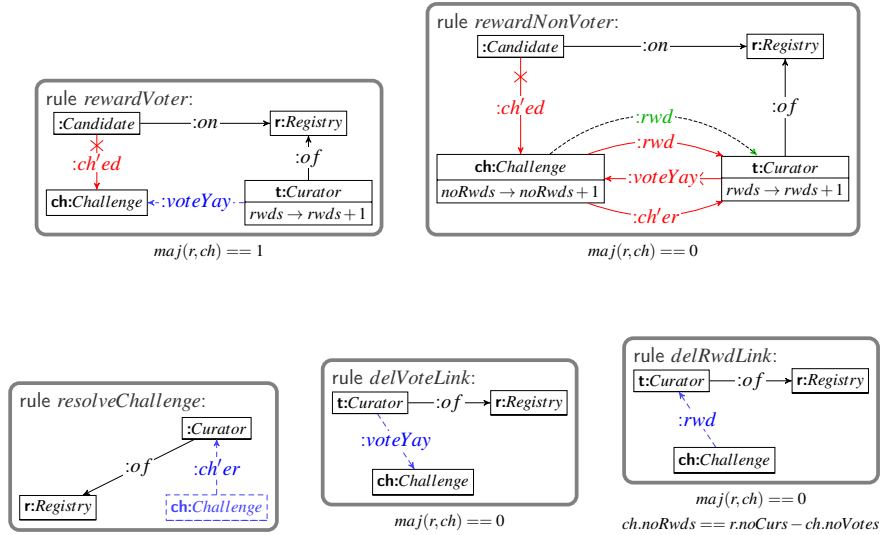
There are two scenarios after voting has finished. If the challenge succeeds, i.e. $maj(ch,r) == 1$

- Every voter is rewarded and the corresponding vote link is deleted.

- Once all vote links are deleted, the challenge can be resolved by rule *resolveChallenge*. Due to the dangling condition, this rule is only enabled once all vote links are deleted.

If the challenge fails, i.e. $maj(ch,r) == 0$

- Every non-voter who isn't the challenger is rewarded and a reward link is pointed to them from the challenge to make sure none is rewarded more than once. Here, the vote links are intact through the rewarding process.

- To safely resolve the challenge, we delete all reward and vote edges by applying *delRwdLink* and *delVoteLink* repeatedly. The order does not matter. *delRwdLink* is only enabled once all non-voters have been rewarded (i.e. $ch.noRwds == r.noCurs - ch.noVotes$). This condition does not change when *delRwdLink* is applied so the rule remains enabled until all reward edges are deleted.

- And finally apply *resolveChallenge* as before.

5.2 Encoding of NACs

Next, we discuss how the NACs of a conditional attributed grammar like the TCR model can be encoded. This encoding extends the analogous construction for safe unattributed grammars. First, let us look at the simpler case of complementation in the case of incremental NACs that have unique occurrences. A NAC $n : L \to N$ has *unique occurrences* if for every reachable graph G and match $m : L \to G$ there is at most

one occurrence $q : N \to G$ of n such that $m \circ n = q$. This can be seen as a form of local safety, permitting an encoding using complement edges similar to the safe case. That means, for all NACs n with unique occurrences we add complement types to the type graph and derive constraints and complemented start graph as described in Definitions 3.1, 3.2 and 3.3. Similarly, we replace each rule p with NAC n by its derived rule obtained by applying p to the complemented version of n as in Construction 3.5. Then, if a rule p deletes/creates an occurrence of n, we extend p to create/delete a parallel occurrence of n's complement as described in Construction 3.6 1/2.

The encoding starts to diverge from the safe case where deletion and creation of boundary nodes is concerned. If n is of shape IN or OUT and p creates/deletes a boundary node in $v \in L_n$, we extend p to create/delete the complement structure $\overline{N_- \setminus n_-(L_n)}$ with it. If n is of shape E and p creates/deletes a boundary node in $v \in L_n$, we extend p by an occurrence of the complement of the negative edge $e \in N_-$ attached to v, and add the other boundary node $u \in L_n$ as a multiobject such that edge e between v and u is created/deleted along with v.

To address the full TCR example, we also have to consider cases where the NACs do not have unique occurrences only. In this case, we introduce reference counters to keep track of how many occurrences there are. That means, we compute the body of the NAC as before, but then, if n is of shape E:

(1) We add to TG a complement type $t_{N^-}(e)^{n^-}$ for the type $t_{N^-}(e)$ of the edge $e \in E_{N^-}$ and introduce an attribute $\#n^- : nat$ to the new edge type. We add a constraint $\#n^- = \mathrm{card}(n^-)$ as an invariant, where $\mathrm{card}(n^-)$ is the number of occurrences of n^- parallel to $\bar{n}^-$.[3]

(2) If there exists a rule that creates or deletes a boundary node in $v \in L^-$, we add an attribute $\#\bar{n}^-$ to the type of v, subject to the invariant $\#\bar{n}^- = \mathrm{card}\{\bar{e} \mid e \text{ is the edge attached to } v \text{ in } N^-\}$ (this is counting the complement edges attached to v).

If n is of shape IN or OUT:

(3) We introduce an attribute $\#n^- : nat$ to the type of the single boundary node v in L^-.

(4) We add constraint $\#n^- = \mathrm{card}(n^-)$ as invariant ($\mathrm{card}(n^-)$ is the number of occurrences of n^-).

Then, for each NAC n and rule p:

(5) If $n \in \mathcal{N}(p)$ we replace p by the derived rule resulting from p's application to $\bar{n}$ with $\#n^- := 0$.

(6) If p creates/deletes k occurrences of the NAC, we extend p to increase/decrease $\#n^-$ by k.

(7) If n is of shape IN or OUT and p creates a boundary node in $v \in L^-$, we extend p by the attribute assignment $\#n^- := 0$.

(8) If n is of shape E and p creates/deletes a boundary node in $v \in L^-$, extend p by an occurrence of the complement of the negative edge $e \in N^-$ attached to v, add the other boundary node $u \in L^n$ as a multiobject such that edge e between v and u is created/deleted along with v. Set $\#\bar{n}^- := 0$ when v is created.

Finally, extend the start graph to include complements of all missing NAC occurrences, add all newly declared attributes and initialise them such that their constraints are satisfied. Together with the adaptations to the rules, this ensures that the constraints are invariants.

Note that we consider rules with multiobjects as rule schemata that unfold into a set of instance rules using amalgamation [4], such that for each rule instance with k copies of a multiobject arising from a boundary node of a NAC n, the value of the attribute $\#\bar{n}^-$ on the other boundary node of n is k.

---

[3] Formally, such a constraint is satisfied in a graph if for all occurrences $o : L^- \to G$ of the shared boundary of n^- and $\bar{n}^-$, the number of compatible occurrences of n^- equals $\#n^-$.

5.3   Encoding the NACs of the TCR

We apply this construction to the TCR grammar as follows (see the figures below). As before, the attributes not mentioned in $\overline{G_0}$ are initialised with zero. There are five rules with NACs, namely *apply*, *challenge*, *voteYay*, *rewardVoter*, and *rewardNonVoter*. With the exception of *challenge* which has a NAC of shape *IN*, all NACs are of shape *E* for which we add an attributed complement edge in $\overline{TG}$ (step (1) in the encoding). Since the boundary node *Challenge* is created by $\overline{challenge}$ and deleted by $\overline{resolveChallenge}$, following step (2), we add attributes to *Challenge* with invariants requiring that each attribute always reflects the number of corresponding edges (e.g. $ch.noVoteBar == card\{voteL : \overline{vote} \mid tar(voteL) == ch\}$). For the NAC in *challenge*, we follow step (3) and add attribute *noCh* to the boundary node *Candidate* that counts the number of arcs of type $ch'ed$ that target it. Based on (4), we assert the invariant $c.noCh == card\{chL : ch'ed \mid tar(chL) == c\}$ for every Candidate c. Conditions on attributes are below each rule.
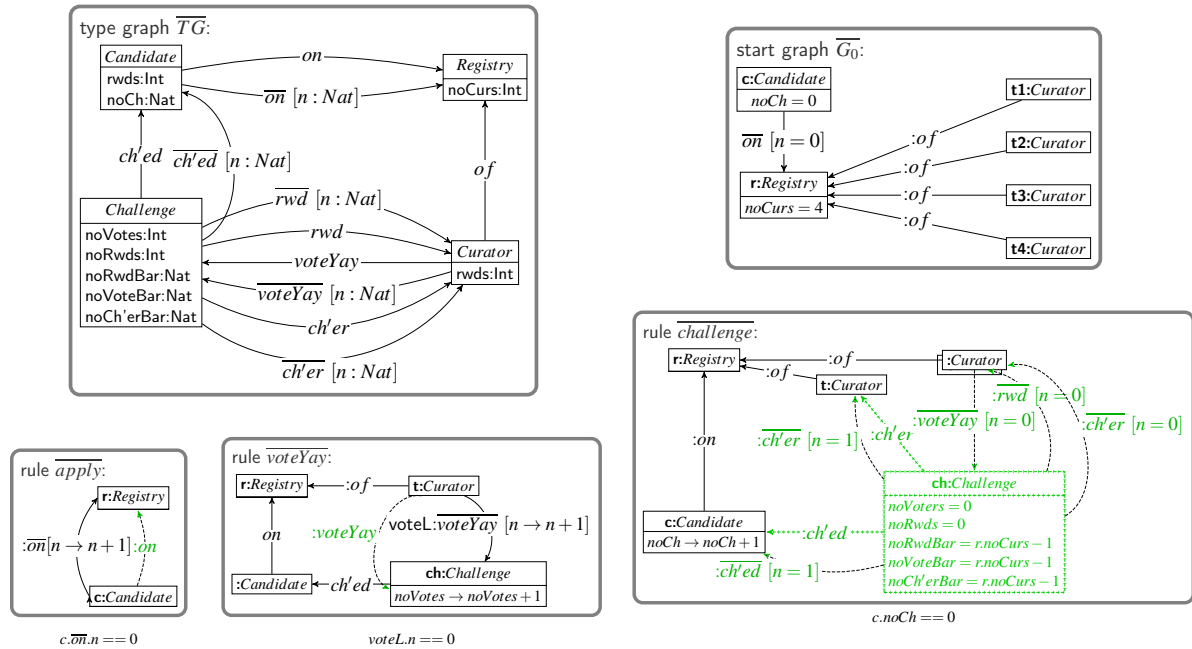

In $\overline{apply}$, if there is no edge of type *on* present (i.e. attribute n of $\overline{on}$ is zero), such an edge is created and the attribute of $\overline{on}$ is incremented by one to reflect this change (cf. (1) and (2) of the encoding). In rule *challenge*, if the candidate c is not currently challenged (i.e. $c.noCh == 0$), a challenge is created and $c.noCh$ is incremented by one. In addition to the creation of the $ch'ed$ edge, its complement $\overline{ch'ed}$ is created with counter initialised to $n = 1$. Since Challenge is a boundary node for complement edges corresponding to NACs in *voteYay* and *rewardNonVoter*, we have a multiobject for curators in rule *challenge* (creates a Challenge node) and *resolveChallenge* (deletes a Challenge node) (step (8)). Rules that contain multiobjects can be interepreted as interaction schemes [4] which expand into countably infinite set of rules with reference count attributes ensuring that the rule with the correct number of instances of the multiobjects is applied in each case.

In the rewarding phase, the rules are the following:
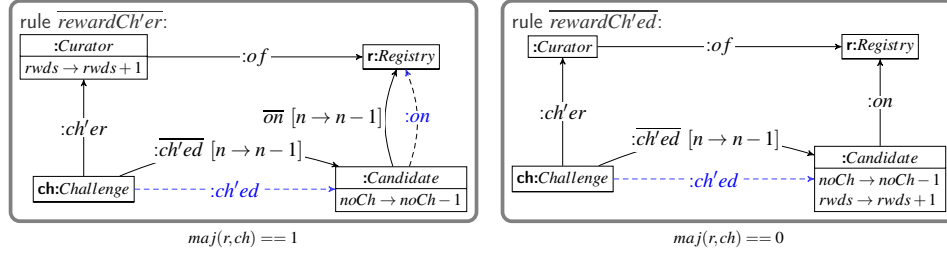
In $\overline{rewardCh'ed}$, we decrement by one the Candidate's reference counter for challenges. In $\overline{rewardCh'er}$, in addition to decrementing $noCh$, we need to decrement the attribute of $\overline{on}$ to reflect deletion of on parallel to it (point 2). Both rules decrement the $\overline{ch'ed}$ counter to reflect the deletion of a $ch'ed$ edge.
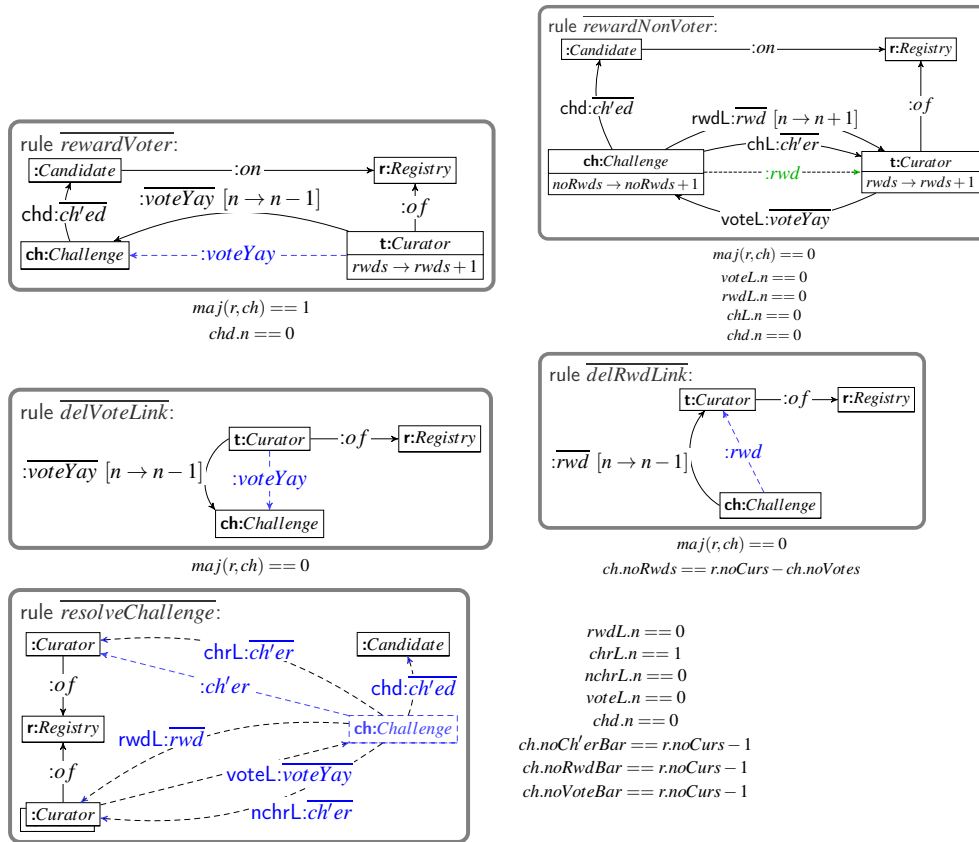
In *rewardNonVoter*, conditions $voteL.n == 0$, $rwdL.n == 0$, $chL.n == 0$, and $chd.n == 0$ ensure that the curator has not voted, has not been rewarded, is not the challenger and that the candidate is no longer linked to the challenge. We increment the attribute of $rwdL$ by one to reflect the creation of an edge of type rwd. The encoding of $\overline{resolveChallenge}$ gives rise to a multiobject because *Challenge* is a boundary node for a constraint of shape E in *voteYay* and *rewardNonVoter* rules (point 4). The challenger has an edge of type $ch'er$ and a complement edge of type $\overline{ch'er}$ whose attribute is 1 whereas every non-challenger curator has three complement edges of types $\overline{ch'er}$, $\overline{voteYay}$, and $\overline{rwd}$ each with an attribute value of 0. The candidate has a $\overline{ch'ed}$ complement edge with counter attribute 0.

6 Conclusion and Future Work

As a contribution towards a comprehensive unfolding semantics applicable also to graph grammars enriched with both attributes and NAC's, in this paper we addressed the problem of encoding the NACs into the graphical structure of the states. We presented this construction formally for the restricted case of unattributed safe grammars and incremental NACs, for which we proved that the construction generates an unconditional grammar having equivalent derivations of the original one and, we conjecture, manifesting the same sequential independence among transformations. This construction is reminiscent of the complementation of Elementary Net Systems [23], a construction known to preserve the branching behavior, thus supporting our confidence that the unfolding semantics of the encoded grammar will correspond precisely to that of the original one. A formal comparison with complementation of nets is left as future work, together with the proof of preservation and reflection of independence.

The key ideas of how to generalize the construction to attributed, possibly unsafe grammars was presented informally. Unsafety requires to count the number of items that could cause the violation of a NAC, and thus it requires the encoded grammar to be attributed. More interestingly, certain shapes of NACs require to encode a rule of the original grammar with a family of rules, that can be generated via an amalgamation. As future work we plan to provide a formalization of the construction in this more general case, and to study to what extent the concurrent semantics is preserved through the construction.

We would also like to generalize our construction to the case of non-incremental NACs, motivated by the observation that in the attributed case incremental NACs have limited expressiveness. But previous work [7] showed that the notion of independence among transitions is not well defined in case of general NACs, thus weakening our overall motivations in this case.

References

[1] Paolo Baldan, Andrea Corradini & Barbara König (2008): *A framework for the verification of infinite-state graph transformation systems*. *Inf. Comput.* 206(7), pp. 869–907, doi:10.1016/j.ic.2008.04.002.

[2] Paolo Baldan, Andrea Corradini & Ugo Montanari (1998): *Unfolding of Double-Pushout Graph Grammars is a Coreflection*. In: *TAGT*, *LNCS* 1764, Springer, pp. 145–163, doi:10.1007/978-3-540-46464-8_11.

[3] Paolo Baldan, Andrea Corradini, Ugo Montanari & Leila Ribeiro (2007): *Unfolding semantics of graph transformation*. *Inf. Comput.* 205(5), pp. 733–782, doi:10.1016/j.ic.2006.11.004.

[4] Enrico Biermann, Hartmut Ehrig, Claudia Ermel, Ulrike Golas & Gabriele Taentzer (2010): *Parallel independence of amalgamated graph transformations applied to model transformation*. In: *Graph transformations and model-driven engineering*, Springer, pp. 121–140, doi:10.1007/978-3-642-17322-6_7.

[5] Andrea Corradini, Maryam Ghaffari Saadat & Reiko Heckel (2019): *Unfolding Graph Grammars with Negative Application Conditions*. In: *ICGT*, Springer, pp. 93–110, doi:10.1007/978-3-030-23611-3_6.

[6] Andrea Corradini & Reiko Heckel (2014): *Canonical Derivations with Negative Application Conditions*. In: *ICGT*, *LNCS* 8571, Springer, pp. 207–221, doi:10.1007/978-3-319-09108-2_14.

[7] Andrea Corradini, Reiko Heckel, Frank Hermann, Susann Gottmann & Nico Nachtigall (2012): *Transformation systems with incremental negative application conditions*. In: *International Workshop on Algebraic Development Techniques*, Springer, pp. 127–142, doi:10.1007/978-3-642-37635-1_8.

[8] Andrea Corradini, Ugo Montanari & Francesca Rossi (1996): *Graph processes*. *Fundamenta Informaticae* 26(3, 4), pp. 241–265, doi:10.3233/FI-1996-263402.

[9] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel & Michael Löwe (1997): *Algebraic approaches to graph transformation–part i: Basic concepts and double pushout approach*. In:

Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations, World Scientific, pp. 163–245, doi:10.1142/3303.

[10] Simone André da Costa Cavalheiro, Luciana Foss & Leila Ribeiro (2017): *Theorem proving graph grammars with attributes and negative application conditions*. *TCS* 686, pp. 25–77, doi:10.1016/j.tcs.2017.04.010.

[11] Hartmut Ehrig, Michael Pfender & Hans Jürgen Schneider (1973): *Graph-Grammars: An Algebraic Approach*. In: *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, IEEE Computer Society, pp. 167–180, doi:10.1109/SWAT.1973.11.

[12] Hartmut Ehrig, Karsten Ehrig ande Ulrike Prange & Gabriele Taentzer (2006): *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/3-540-31188-2.

[13] Maryam Ghaffari Saadat, Reiko Heckel & Fernando Orejas (2020): *Unfolding Symbolic Attributed Graph Grammars*. In: *ICGT, LNCS* 12150, Springer, pp. 75–90, doi:10.1007/978-3-030-51372-6_5.

[14] Annegret Habel, Reiko Heckel & Gabriele Taentzer (1996): *Graph Grammars with Negative Application Conditions*. *Fundam. Inform.* 26(3/4), pp. 287–313, doi:10.3233/FI-1996-263404.

[15] Annegret Habel & Karl-Heinz Pennemann (2009): *Correctness of high-level transformation systems relative to nested conditions*. *Math. Struct. Comput. Sci.* 19(2), pp. 245–296, doi:10.1017/S0960129508007202.

[16] Reiko Heckel & Gabriele Taentzer (2020): *Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering*. 1, Springer International Publishing, doi:10.1007/978-3-030-43916-3.

[17] Manuel Koch & Francesco Parisi-Presicce (2002): *Describing policies with graph constraints and rules*. In: *International Conference on Graph Transformation*, Springer, pp. 223–238, doi:10.1007/3-540-45832-8_18.

[18] Stephen Lack & Paweł Sobociński (2004): *Adhesive categories*. In: *FOSSACS*, Springer, pp. 273–288, doi:10.1007/978-3-540-24727-2_20.

[19] Leen Lambers (2009): *Certifying rule-based models using graph transformation*. Ph.D. thesis, Berlin Institute of Technology.

[20] Nebras Nassar, Jens Kosiol, Thorsten Arendt & Gabriele Taentzer (2020): *Constructing optimized constraint-preserving application conditions for model transformation rules*. *J. Log. Algebraic Methods Program.* 114, doi:10.1016/j.jlamp.2020.100564.

[21] Fernando Orejas & Leen Lambers (2010): *Symbolic Attributed Graphs for Attributed Graph Transformation*. *ECEASST* 30, doi:10.14279/tuj.eceasst.30.405.

[22] Bruno Pinaud, Guy Melançon & Jonathan Dubois (2012): *PORGY: a visual graph rewriting environment for complex systems*. In: *Computer Graphics Forum*, 31, Wiley Online Library, pp. 1265–1274, doi:10.1111/j.1467-8659.2012.03119.x.

[23] Grzegorz Rozenberg (1986): *Behaviour of Elementary Net Systems*. In: *ACPN, LNCS* 254, Springer, pp. 60–94, doi:10.1007/BFb0046836.

[24] Fabian Vogelsteller & Vitalik Buterin (2015): *ERC-20 Token Standard: A standard interface for tokens*. Available at `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md`.