,

Structural Morphism for Nested Conditions

Arend Rensink[1], Andrea Corradini[2]

[1]University of Twente, Netherlands
[2]University of Pisa, Italy

Abstract

Nested conditions are used, among other things, as a graphical way to express first order formulas ruling the applicability of a graph transformation rule to a given match. In this paper, we first introduce several operators on conditions mimicking logical connectives. Next we propose an original notion of structural morphism among nested conditions, and we identify circumstances under which morphisms are consistent with the entailment of the corresponding conditions. Finally we frame the results in a categorical context, proving functoriality and universality properties of the various operations.

1. Introduction

Representing formulas of First-Order Logic (FOL) by graphs or more general graphical structures was explored in various areas of Theoretical Computer Science and Logics along the decades. A canonical example is represented by edge-labelled graphs, which can be regarded as an alternative syntax for formulas of a fragment of FOL including just basic (binary) predicates, equality, conjunction and existential quantification: we call this briefly the ∃-fragment.

Let $\mathsf{L} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}$ be a set of binary relation symbols, which we shall use also as edge labels. As an example, let A be the graph $\boxed{x \xrightarrow{\mathsf{a}} y \xrightarrow{\mathsf{b}} z}$. We consider this graph as a sound representation of the formula $\phi_A = \exists x, y, z.\, \mathsf{a}(x, y) \wedge \mathsf{b}(y, z)$, in the following sense: a graph G satisfies ϕ_A (or is a *model* of ϕ_A) if and only if there is a graph morphism h from A to G. Now consider $B = \boxed{x \xrightarrow{\mathsf{a}} y}$, representing formula $\phi_B = \exists x, y.\, \mathsf{a}(x, y)$. Graph B has an obvious inclusion morphism into A, viz. $i : B \to A$. Therefore every morphism $h : A \to G$ gives rise to a composed morphism $i; h : B \to G$,[1] implying that every graph that satisfies ϕ_A also satisfies ϕ_B, thus ϕ_A *entails* ϕ_B (written $\phi_A \vDash \phi_B$). It is worth noting that, in this elementary framework, graph morphisms can represent both a satisfaction relation (between formulas and models) and an entailment relation among formulas.

These concepts were exploited for example by Chandra and Merlin in [1] in the framework of relational database queries. They show there that every *conjunctive query* (a formula of the ∃-fragment, but with relations of any arity) has a natural model, viz. a graph, and query inclusion is equivalent to the existence of a graph homomorphism between those natural models. Therefore morphisms are not only sound, but also complete with respect to entailment, and it follows that query inclusion is decidable, even if NP-complete: an interesting logical result obtained with graph theoretical techniques.

In the realm of Graph Transformation Systems (GTSs) [2, 3], the need of representing formulas by graphs arose in a natural way. Indeed, in any approach a rule consists of at least two graphs, $L \rightsquigarrow R$, and to apply it to a graph G, first a morphism $m : L \to G$ has to be found. By the above discussion, we can consider L as a formula of the ∃-fragment that has to be satisfied by G, as an application condition of the rule.

It soon turned out that in order to use GTSs for even simple specifications, more expressive application conditions were needed. In [4] the authors introduced *Negative Application Conditions (NACs)*, allowing to express (to some extent) negation and disjunction. A NAC $\mathcal{N}$ is a finite set of morphisms from

Joint Proceedings of the STAF 2026 Workshops: AgileMDE, GCM, ICMM, LLM4SE, TTC. Rennes, France, June 29-July 3, 2026
arend.rensink@utwente.nl (A. Rensink); andrea.corradini@unipi.it (A. Corradini)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

[1]Along the paper, we denote by $f; g : A \to C$ the composition of arrows $f : A \to B$ and $g : B \to C$, using diagrammatic order.

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-26

CEUR-WS.org/Vol-4338/GCM-paper3.pdf

L, $\mathcal{N} = \{n_i : L \to Q_i\}_{i\in[1,k]}$, and a morphism $m : L \to G$ *satisfies* $\mathcal{N}$ if for all $i \in [1,k]$ there is no morphism $m_i : Q_i \to G$ such that $n_i; m_i = m$. It follows that such a NAC represents a formula of the shape $\exists \bar{x}.\, \phi_L \wedge \neg(\exists \bar{y}_1.\, \phi_{Q_1} \vee \ldots \vee \exists \bar{y}_n.\, \phi_{Q_n})$: as in [5], we call this the $\exists\neg\exists$-fragment (of FOL).

Note that differently from the $\exists$-fragment, the structures representing the $\exists\neg\exists$-fragment are no longer graphs, but diagrams ("stars") in **Graph**, the category of graphs; and satisfaction does not require just the existence of a matching morphism from L, but also the non-existence of certain other morphisms. NACs were generalized in [5, 6] to *Nested (Application) Conditions*, where the structure of a condition is a finite tree of arbitrary depth rooted at L, and satisfaction is defined like for NACs, but iterating further at each level of the tree. Interestingly, nested conditions were proved to have the same expressive power of full FOL.

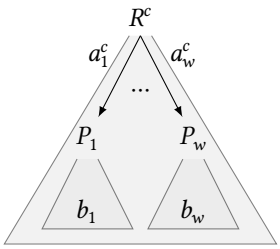
Since nested conditions denote formulas, they are the objects of an obvious category (actually, a preorder) where arrows represent entailment: paper [7] includes a comprehensive presentation of this preorder and of the categorical properties of logical operators on conditions.[2] But differently from the case of the $\exists$-fragment, where entailment can be "explained" by the existence of a graph morphism, we are not aware of similar results for the larger $\exists\neg\exists$-fragment nor for FOL. More explicitly, despite the fact that NACs first and Nested Conditions next were defined as suitable diagrams in a category of graphs (or of similar structures), we are not aware of definitions of *structural morphisms* among such application conditions, providing evidence for (some cases of) entailment like simple graph morphisms do for the $\exists$-fragment. We addressed this issue for the first time in [8] by considering two classes of nested conditions, the *arrow-based* ones (those of [5]) and the more refined *span-based* ones, introducing various kinds of morphisms for them. In this paper we stick to the first kind of conditions, but we propose a more systematic and general approach, which is the main contribution of this paper. In particular, the morphisms we define here at the same time are simpler than and subsume those of [8] for arrow-based conditions. Their generalization to span-based conditions is ongoing work.

We start by recalling in Sec. 2 the main definitions related to nested conditions and their satisfaction from [5], and we present several constructions on them aimed at capturing logical operators in a categorical setting. The section is concluded by a completeness result showing that every condition can be built in a canonical way using the proposed operators. Section 3 presents the main original contribution, i.e. a very general and weak notion of structural morphisms among conditions, which satisfies the desired properties of being composable and having identities. Such morphisms include collection of arrows between the source and target condition, flipping direction at each nesting level, and forming certain (possibly not commuting) squares with the arrows of the conditions themselves. By imposing suitable conditions on such squares, we identify two subclasses of morphisms, *reflective* and the *preservative* ones, which are shown to reflect and preserve satisfaction, respectively, and thus witness entailment between the formulas associated with the related conditions. We also show that the composition of morphisms of the same kind is again a morphism of that kind, and that identities are morphisms of both kinds. Finally we show that under mild assumptions for each preservative morphism there is a reflective one in the opposite direction. In Sec. 4 we first frame the definitions of the various kind of morphisms of Sec. 3 and their relations in a categorical context, introducing several categories of conditions and identity-on-object functors among them. Next, this allows us to state and prove some functoriality and universality properties of the constructions on conditions of Sec. 2. Section 5 summarizes the contributions of the paper, discusses some related work and hints at future developments.

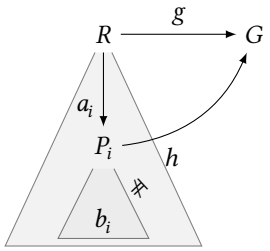
2. Nested conditions and satisfaction

We start by recalling some basic concepts and the standard notion of nested condition from [5]. We will drop the adjective "nested" and simply speak of *conditions*, as we will not consider any other kind of conditions along the paper.

[2] Nested conditions are defined in a quite different way in this paper, but the expressive power is the same.



(a) Condition $c = (R, p_1 \cdots p_w)$, with $p_i = (a_i, b_i)$ for $1 \leq i \leq w$



(b) $g \vDash c$, with responsible branch $p_i = (a_i, b_i)$ and witness h such that $g = a_i; h$

Figure 1: Visualisations for conditions

Assumption For the sake of generality, along the paper formal definitions and results will be phrased in terms of objects and arrows of a generic category $\mathbf{C}$ that we assume to be a *category of presheaves*, i.e., a category of contravariant functors from a small category $\mathbf{S}$ to **Set**, thus $\mathbf{C} = [\mathbf{S}^{\mathrm{op}} \to \mathbf{Set}]$. Categories of presheaves are known to be toposes [9]: we will call them *presheaf toposes*.

Assuming that $\mathbf{C}$ is a presheaf topos ensures several properties we need in the constructions of this paper: in particular, that all limits and colimits exist (and can be computed pointwise), and also that epis are stable under pullback [9]. Furthermore, $\mathbf{C}$ is *adhesive* [10], enjoying several properties exploited in the algebraic theory of graph rewriting, where the results of this paper have potential interesting applications. Note that requiring $\mathbf{C}$ to be just adhesive would not suffice: for example, we need arbitrary pushouts, while adhesivity only guarantees pushouts along monos.

Example 2.1 (presheaf toposes). Several categories of graphs and hypergraphs are presheaf toposes, including *graph structures* [11]. For example, directed unlabelled graphs are obtained with $\mathbf{S}$ the free category generated by $\bullet \rightrightarrows \bullet$. Furthermore, presheaf toposes are closed under the construction of slice and functor categories (see Sec. 5 of [12] for a recap), thus they also include *triple graphs* [13], *typed graphs* [2], and in particular **Graph**, the category of directed, edge-labelled multigraphs introduced below, that will be used for examples and intuitions in the rest of the paper.

Definition 2.2 (Category of Graphs)**.** A *(directed, edge-labelled) graph* $G = (V_G, E_G, s_G, t_G, \ell_G)$ has sets V_G of nodes and E_G of edges, along with source and target functions $s_G, t_G : E_G \to V_G$, and an edge-labelling function $\ell_G : E_G \to \mathsf{L}$. A *graph morphism* $f : G \to H$ is a pair of functions $f = (f_V : V_G \to V_H, f_E : E_G \to E_H)$ that preserve incidence (that is, $s_G; f_V = f_E; s_H$ and $t_G; f_V = f_E; t_H$) and labels ($f_E; \ell_H = \ell_G$). Graphs and graph morphisms determine category **Graph**. □

Conditions are inductively defined as follows:

Definition 2.3 (condition)**.** Let R be an object of $\mathbf{C}$. $\mathsf{NC}(R)$ (the set of *conditions* over R) and $\mathsf{NB}(R)$ (the set of *branches* over R) are the smallest sets such that

- $c \in \mathsf{NC}(R)$ if $c = (R, p_1 \cdots p_w)$ is a pair with $p_i \in \mathsf{NB}(R)$ for all $1 \leq i \leq w$, where $w \geq 0$;
- $p \in \mathsf{NB}(R)$ if $p = (a, b)$ where $a : R \to P$ is the *branch arrow* and $b \in \mathsf{NC}(P)$ the *branch condition*. □

We call R the *root* of a condition or branch, and P the *pattern* of a branch (which is simultaneously the root of its branch condition). Figure 1a provides a visualisation of a condition c. We use c to range over conditions and p, q to range over branches. We use $|c| = w$ to denote the width of a condition c, R^c to denote its root, and $p_i^c = (a_i^c, b_i^c)$ its i-th branch; for a condition c_j, the notations are R_j for the root and $p_i^j = (a_i^j, b_i^j)$ for the i-th branch. Finally, we use P_i^c ($= R^{b_i}$) for the pattern of branch p_i^c.

Note that, as a consequence of the inductive nature of Def. 2.3, every condition has a finite *depth* $dp(c)$, defined as 0 if $|c| = 0$ and $1 + \max_{1 \leq i \leq |c|} dp(c_i)$ otherwise. The depth will provide a basis for inductive proofs.

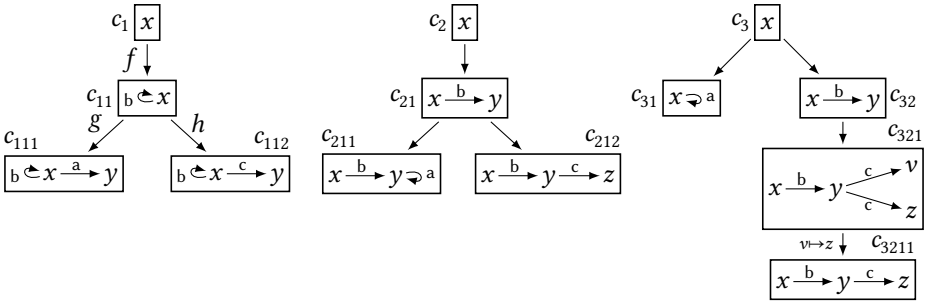


Figure 2: Examples of conditions (see Ex. 2.4)

Example 2.4. Fig. 2 shows the graphical representation of three conditions, rooted in the discrete one-node graph $\boxed{x}$. The graph morphisms are in all cases implied by the graph structure and node names. The leftmost one, $c_1 \in \mathsf{NC}(\boxed{x})$, is defined according to Def. 2.3 as

$$c_1 = (\boxed{x}, (f, c_{11})) \qquad c_{11} = (\boxed{\mathrm{b} \curvearrowleft x}, (g, c_{111})(h, c_{112}))$$
$$c_{111} = (\boxed{\mathrm{b} \curvearrowleft x \xrightarrow{\mathrm{a}} y}, \epsilon) \quad c_{112} = (\boxed{\mathrm{b} \curvearrowleft x \xrightarrow{\mathrm{c}} y}, \epsilon)$$

For the depths, $dp(c_{111}) = dp(c_{112}) = 0$, $dp(c_{11}) = 1$ and $dp(c_1) = 2$.

Based on the notion of satisfaction that we are about to introduce, every condition can be seen as a property on morphisms from its root to a graph, expressed in the first-order logic on graphs, see [5]. Due to space constraints, we do not discuss the precise correspondence between conditions and FOL formulas here, but we present the formulas that correspond to the conditions of Fig. 2:

- c_1 is equivalent to $\mathrm{b}(x, x) \wedge \neg\exists y.\, (\mathrm{a}(x, y) \vee \mathrm{c}(x, y))$
- c_2 is equivalent to $\exists y.\, \mathrm{b}(x, y) \wedge \neg\mathrm{a}(y, y) \wedge \neg\exists z.\, \mathrm{c}(y, z)$
- c_3 is equivalent to $\mathrm{a}(x, x) \vee (\exists y.\, \mathrm{b}(x, y) \wedge (\forall v, z.\, \mathrm{c}(y, v) \wedge \mathrm{c}(y, z) \to v = z))$ □

2.1. Satisfaction

A condition expresses a property of arrows from its root to arbitrary objects. This is operationalised through the notion of *satisfaction*.

Definition 2.5 (satisfaction)**.** Let $g : R \to G$ be an arbitrary arrow.

- Let $c = (R, p_1 \cdots p_w) \in \mathsf{NC}(R)$. We say that g *satisfies* c, denoted $g \vDash c$, if there is a branch p_i such that $g \vDash p_i$.
- Let $p = (a : R \to P, c) \in \mathsf{NB}(R)$. We say that g *satisfies* p, denoted $g \vDash p$, if there is an arrow $h : P \to G$ such that *(i)* $g = a; h$ and *(ii)* $h \nvDash c$. □

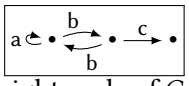
If $g \vDash c$, we also say that g is a *model* for c. We call $p_i = (a_i, c_i)$ the *responsible branch* and h such that $g = a_i; h$ and $h \nvDash c_i$ the *witness* for $g \vDash c$. Pictorially, $g \vDash c$ with responsible branch p_i and witness h can be visualised as in Fig. 1b.

Definition 2.6 (entailment, equivalence)**.** For conditions $c_1, c_2 \in \mathsf{NC}(R)$ over the same root R we define *semantic entailment* $c_1 \vDash c_2$ and *semantic equivalence* $c_1 \equiv c_2$:

$$c_1 \vDash c_2 \text{ if for all arrows } g : R \to G,\ g \vDash c_1 \text{ implies } g \vDash c_2$$
$$c_1 \equiv c_2 \text{ if for all arrows } g : R \to G,\ g \vDash c_1 \text{ if and only if } g \vDash c_2 \ .$$ □

Example 2.7. Let us consider some models for the conditions in Fig. 2.

- Let $G_1 = \boxed{\mathrm{c} \curvearrowleft \bullet \underset{\mathrm{b}}{\overset{\mathrm{b}}{\rightleftarrows}} \bullet}$ and let g be the morphism from the one-node discrete graph $\boxed{x}$ to the left hand node of G_1. It is clear that $g \nvDash c_1$ because there is no witness for c_{11} (G_1 does not have the required b-loop). Instead, we have $g \vDash c_2$: the witness h (for c_{21}) maps y to the right hand node of G_1; and h does not satisfy the subconditions of c_{21} because it cannot be extended with either the a-loop specified by c_{211} or the outgoing c-edge specified by c_{212}. Similarly, $g \vDash c_3$.

- Let $G_2 =$  and let g, g', g'' be the morphisms from $\boxed{x}$ to G_2 mapping $\boxed{x}$ to the left, mid and right node of G_2, respectively. None of these models satisfy c_1, for the same reason as above. Moreover, none satisfy c_2: though g and g' have witnesses for c_{21}, these are ruled out by either c_{212} (in the case of g) or c_{211} (in the case of g'); g'' does not even have a witness for c_{21}. Instead, both $g \models c_3$ (in fact there are two distinct witnesses, one for c_{31} and one for c_{32}) and $g' \models c_3$ (due to c_{32}); but again $g'' \not\models c_3$.
- If $G_3 =$ , then the only morphism g from $\boxed{x}$ to G_3 has $g \models c_1$, $g \models c_2$ and $g \models c_3$.

In general, it can be checked that every model of c_1 is also a model of c_2 and every model of c_2 is a model of c_3, thus $c_1 \models c_2$ and $c_2 \models c_3$. On the other hand, $c_2 \not\models c_1$ as shown by $g : \boxed{x} \to G_1$, and $c_3 \not\models c_2$ as shown by $g : \boxed{x} \to G_2$. □

2.2. Constructions over conditions

We recall and introduce some special conditions and constructions that are used in the developments of this paper, together with their properties with respect to satisfaction.

Falsehood. For any graph R, there is an obvious "trivial" condition rooted in R, namely the one without any branches.

$$\bot_R \triangleq (R, \epsilon) \ . \tag{1}$$

The following result is immediate; it confirms that $\bot_R$ represents *false* for models rooted in R.

Proposition 2.8 ($\bot$ captures falsehood)**.** *Let R be a graph. Then $g \not\models \bot_R$ for any $g : R \to G$.*

Inversion. Given a condition over R, a straightforward construction is to "push it down a level" by making it a subcondition of the identity arrow id_R.

Definition 2.9 (inverse)**.** Let $c \in \mathsf{NC}(R)$. The *inverse* of c is given by $!c \triangleq (R, (id_R, c)) \in \mathsf{NC}(R)$. □

The following proposition states that inverse behaves like negation: a model satisfies a condition if and only if it does not satisfy its inverse. The proof is straightforward and omitted here.

Proposition 2.10 (inverse captures negation)**.** *Let $c \in \mathsf{NC}(R)$. For any $g : R \to G$, $g \models c$ iff $g \not\models !c$.*

Shifting. We introduce two basic operations over conditions that allow one to change (*shift*) the condition's root by exploiting arrows from or to the root. Given a condition $c \in \mathsf{NC}(R)$ and an incoming arrow $u : S \to R$, we can *upshift* c along u obtaining a condition $c{\uparrow}u \in \mathsf{NC}(S)$. Dually, given an outgoing arrow $v : R \to Q$, we can *downshift* c along v by exploiting the existence of pushouts in $\mathbf{C}$, obtaining a condition $c{\downarrow}v \in \mathsf{NC}(Q)$ (see Fig. 3).[3]

For the components of the pushout of two arrows $v : R \to Q$ and $a : R \to P$ we will use the notation illustrated in the following diagram, assuming that $(P{\downarrow}Q, v{\downarrow}a, a{\downarrow}v)$ is a concretely chosen pushout of v and a (which is defined up to isomorphism).

$$\begin{array}{ccc} R & \xrightarrow{v} & Q \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle a{\downarrow}v} \\ P & \xrightarrow[v{\downarrow}a]{} & P{\downarrow}Q \end{array} \tag{2}$$

Definition 2.11 (upshift)**.** Let $u : S \to R$ be an arrow in $\mathbf{C}$.

- For a condition $c = (R, p_1 \cdots p_w) \in \mathsf{NC}(R)$, the *upshift of c along u* is the condition $c{\uparrow}u \in \mathsf{NC}(S)$ defined by $c{\uparrow}u \triangleq (S, p_1{\uparrow}u \cdots p_w{\uparrow}u)$;
- For a branch $p = (a, b) \in \mathsf{NB}(R)$, the *upshift of p along u* is the branch $p{\uparrow}u \in \mathsf{NB}(S)$ defined by $p{\uparrow}u \triangleq (u; a, b)$. □

[3] In the literature (e.g., [14, 7]), downshift is more commonly just called *shift*.

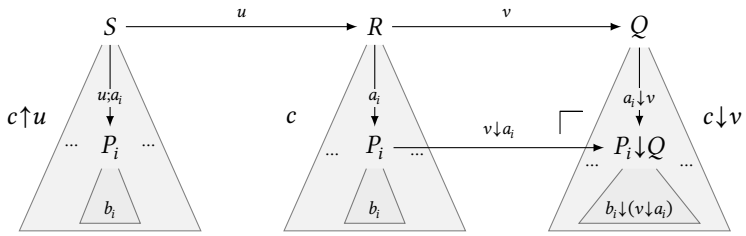


Figure 3: Upshift (left) and downshift (right) of c along u resp. v.

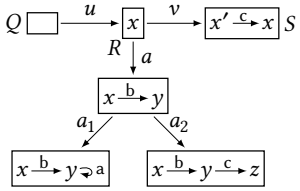
Note that upshifting c along an incoming arrow u is *not* the same as constructing a branch $p = (u, c)$ where c becomes the subcondition of p. The semantic difference will become clear below (Prop. 2.14).

Definition 2.12 (downshift)**.** Let $v : R \to Q$ be an arrow in $\mathbf{C}$.

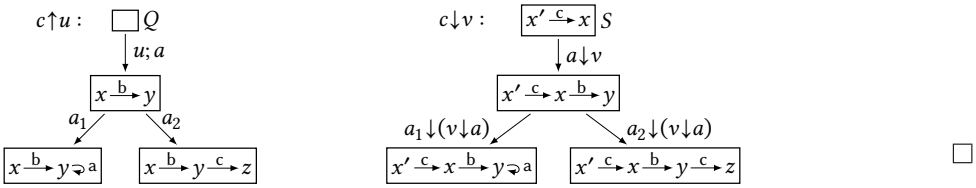
- For a condition $c = (R, p_1 \cdots p_w) \in \mathsf{NC}(R)$, the *downshift of c along v* is the condition $c{\downarrow}v \in \mathsf{NC}(Q)$ defined by $c{\downarrow}v \triangleq (Q, p_1{\downarrow}v \cdots p_w{\downarrow}v)$.
- For a branch $p = (a, b) \in \mathsf{NB}(R)$, the *downshift of p along v* is the branch $p{\downarrow}v \in \mathsf{NB}(Q)$ defined by $p{\downarrow}v \triangleq (a{\downarrow}v, b{\downarrow}(v{\downarrow}a))$. □

Note that *upshifting* a condition along an arrow only affects the top-level branches, while the branch conditions are preserved. Instead, *downshifting* a condition along an arrow also affects all branch conditions, which are recursively shifted along the corresponding pushout arrows.

Example 2.13. Consider the following condition c (which equals c_2 of Fig. 2), with the arrows $u : Q \to R$ and $v : R \to S$ as in the following diagram:



The upward shifted condition $c{\uparrow}u$ and downward shifted condition $c{\downarrow}v$ are then given by:



□

Downshifting a condition along an arrow preserves the models prefixed with such arrow (again see [14, 7]), whereas upshifting along an arrow preserves them, and also reflects them if the arrow is epi.

Proposition 2.14 (satisfaction for upshift and downshift)**.** *Let $c \in \mathsf{NC}(R)$.*

1. *Let $v : R \to S$. Then $v; g \vDash c$ iff $g \vDash c{\downarrow}v$.*
2. *Let $u : Q \to R$. Then $g \vDash c{\uparrow}u$ iff there exists an arrow g' such that $g = u; g'$ and $g' \vDash c$.*
3. *If $u; g \vDash c{\uparrow}u$ and u is epi then $g \vDash c$.*

The notation chosen for the upshift operation is motivated by its symmetry with downshift. Upshift has a flavour of existential quantification, and indeed an operation denoted $\exists u.\, c$ is introduced in [7] having the same semantics, in the sense that both satisfy the second point of Prop. 2.14. However, the notion of nested condition of [7] and the concrete definition of the existential operation differ from ours: a precise comparison is deferred to future work.

Join and meet. We define the *join* and *meet* of two conditions over the same root, and show that these capture the disjunction, respectively the conjunction in terms of satisfaction. The first of these, join, merely takes the union of the branches of the two conditions.

Definition 2.15 (join)**.** Let $c_i = \langle R, p_1^i \cdots p_{w_i}^i \rangle \in \mathsf{NC}(R)$ for $i = 1, 2$. The *join of* c_1 *and* c_2 is defined by $c_1 \sqcup c_2 \triangleq (R, p_1^1 \cdots p_{w_1}^1\ p_1^2 \cdots p_{w_2}^2)$. □

It is straightforward to show that the join captures disjunction.

Proposition 2.16 (join captures disjunction)**.** *Given two conditions* $c_i \in \mathsf{NC}(R)$ *for* $i = 1, 2$ *and an arrow* $g : R \to G$, $g \vDash c_1 \sqcup c_2$ *if and only if either* $g \vDash c_1$ *or* $g \vDash c_2$.

We subsequently define the *meet* of two conditions over the same root. The construction relies as an intermediate step on the meet of two branches.

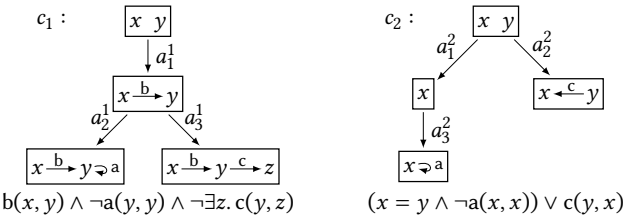
Definition 2.17 (meet)**.**

- Let $p_i = (a_i, b_i) \in \mathsf{NB}(R)$ for $i = 1, 2$. The *meet of* p_1 *and* p_2 is a branch $p_1 \sqcap p_2 \in \mathsf{NB}(R)$, defined by $p_1 \sqcap p_2 \triangleq (a_1; (a_2 \downarrow a_1), b_1 \downarrow (a_2 \downarrow a_1) \sqcup b_2 \downarrow (a_1 \downarrow a_2))$.
- Let $c_i = (R_i, p_1^i \cdots p_{w_i}^i) \in \mathsf{NC}(R)$ for $i = 1, 2$. The *meet of* c_1 *and* c_2 is a condition $c_1 \sqcap c_2 \in \mathsf{NC}(R)$, defined by $c_1 \sqcap c_2 \triangleq \langle R, p_{1,1} \cdots p_{w_1,1} \cdots p_{1,w_2} \cdots p_{w_1,w_2} \rangle$, where $p_{m,n} = p_m^1 \sqcap p_n^2$ for all $m \in [1, w_1], n \in [1, w_2]$. □

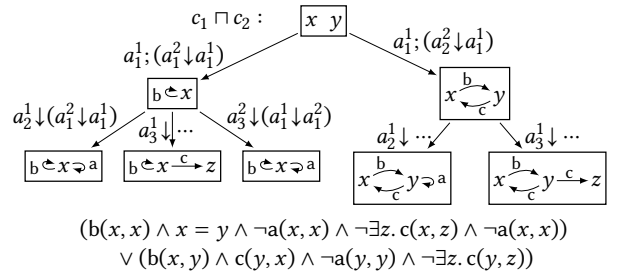
Note that, although the definition of the meet for branches might seem asymmetric due to the use of the arrow $a_1; (a_2 \downarrow a_1)$, this is actually not the case as $a_1; (a_2 \downarrow a_1) = a_2; (a_1 \downarrow a_2)$ (see (2)).

Example 2.18. Since the meet is defined in terms of downshift, one simple case is already given by a reinterpretation of the downshift example in Ex. 2.13: the result of that downshift can be seen as the meet of the condition c and the condition $(R, (v, (S, \epsilon)))$.

For a slightly more elaborate example, consider the following two conditions $c_1, c_2 \in \mathsf{NC}(\boxed{x\ \ y})$ (with FOL interpretations given below the condition):



In this case, the meet can be depicted as



□

From the example, the intuition can be gathered that the construction of the meet is on some level related with the distributivity of logical conjunction over disjunction; and in fact, we can show that $(\bigsqcup_{i \in I} c_i) \sqcap (\bigsqcup_{j \in J} c'_j) \cong \bigsqcup_{i \in I, j \in J} (c_i \sqcap c'_j)$, where $\cong$ stands for isomorphism between conditions.

We show that the meet indeed captures conjunction, as announced.

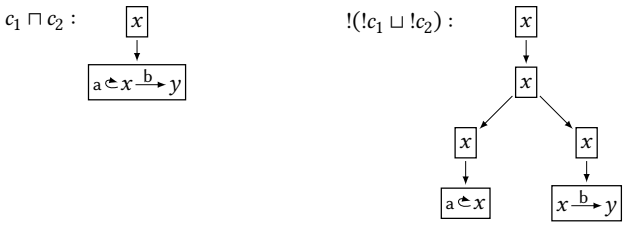
Proposition 2.19 (meet captures conjunction)**.** *Let* $g : R \to G$ *be an arbitrary arrow.*

1. *If* $p_i \in \mathsf{NB}(R)$ *for* $i = 1, 2$, *then* $g \vDash p_1 \sqcap p_2$ *if and only if* $g \vDash p_1$ *and* $g \vDash p_2$.

2. *If* $c_i \in \mathsf{NC}(R)$ *for* $i = 1, 2$*, then* $g \vDash c_1 \sqcap c_2$ *if and only if* $g \vDash c_1$ *and* $g \vDash c_2$*.*

One may wonder how useful the meet is, given that there is obviously another way to capture conjunction, due to De Morgan duality, through a combination of the (technically much simpler) inversion and join constructions. Indeed, Props. 2.10 and 2.16 together guarantee that, for any $c_1, c_2 \in \mathsf{NC}(R)$, if we let $c = !(!c_1 \sqcup !c_2)$, then $g \vDash c$ if and only if $g \vDash c_1$ and $g \vDash c_2$; in other words, $c \equiv c_1 \sqcap c_2$. However, those two constructions give rise to very different conditions, as the following simple example illustrates.

Example 2.20. Consider $a_1 = \boxed{x} \to \boxed{\mathrm{a} \doteq x}$ and $a_2 = \boxed{x} \to \boxed{x \xrightarrow{\mathrm{b}} y}$, and let $c_i = c_{a_i}$ for $i = 1, 2$. The following depicts the two equivalent conditions obtained taking the meet of c_1 and c_2 (left hand side) and applying the De Morgan duality (right hand side).



Clearly, for this particular example, the left hand condition is more concise and understandable than the right hand condition. In general, there is a trade-off: meet ($c_1 \sqcap c_2$) creates a condition of which the depth is the *maximum* of $dp(c_1)$ and $dp(c_2)$, whereas the branching width is the *product* of $|c_1|$ and $|c_2|$, and for each of the new branches $p = (a, b)$ (which always equal $p^1 \sqcap p^2$ for some branch $p^1 = (a^1, b^1)$ of c_1 and some branch $p^2 = (a^2, b^2)$ of c_2) the next-level widths *sum up*: $|b| = |b^1| + |b^2|$. For the De Morgan-based construction ($!(!c_1 \sqcup !c_2)$), on the other hand, the depth is always the maximum of $dp(c_1)$ and $dp(c_2)$ *increased by 2*, whereas the width is merely the *sum* of $|c_1|$ and $|c_2|$ (and the branch subconditions remain unchanged). □

The following proposition shows that the constructions presented in this section are in a sense complete: they can be used to build any condition, using arbitrary arrows as upward-shift operands.

Proposition 2.21. *Any condition* c *can be constructed through a combination of constants* $\bot_R$*, inversion* (!)*, upshift* (↑) *and join* (⊔)*.*

3. Morphisms

The notion of *morphism* of nested conditions has not received much attention in the literature. When considered at all, such as in [7, 15], morphisms are essentially based on the semantics in terms of satisfaction. Indeed, entailment (see Def. 2.6) establishes a preorder on conditions over the same root: given conditions $c_1, c_2 \in \mathsf{NB}(R)$ we have $c_1 \leq c_2$ iff $c_1 \vDash c_2$. The question that we address in this section is to establish a meaningful *structural* notion of condition morphism. That is, given the fact that a condition is essentially a diagram in the category $\mathbf{C}$, a structural morphism from c_1 to c_2 consists of arrows between objects of c_1 and c_2 (when viewed as diagrams), satisfying certain conditions on the resulting subdiagrams.

A morphism will relate the roots of the source and target conditions with an arrow, and will fix a relation between the branches of the source and target conditions. For each pair of related branches the morphism will include a set of "next-level" morphisms between the corresponding branch conditions, which will go the other way around (i.e., from the target branch condition to the source branch condition): this is motivated by our definition of satisfaction, which flips the direction of the relation at every subsequent nesting level. This general notion of structural morphisms will be specialized in the next section to *reflective* and *preservative* morphisms, which will be shown to witness entailment in the expected direction.

Definition 3.1 (condition morphism)**.** Let $c_i = (R_i, (a_1^i, b_1^i) \cdots (a_{w_i}^i, b_{w_i}^i)) \in \mathsf{NC}(R_i)$ for $i = 1, 2$. A *condition morphism* from c_1 to c_2 is a tuple $m = (v, I, N) :\ c_1 \to c_2$ such that

- $v : R_2 \to R_1$ is an arrow, called the *root arrow*;
- $I \subseteq [1, w_1] \times [1, w_2]$ is a relation between the branches of c_1 and c_2;
- N is a family of non-empty sets of condition morphisms indexed by I, that is $N = \{N_{jk} \mid (j,k) \in I\}$, such that $n : b^2_k \to b^1_j$ for each $n \in N_{jk}$.

The components of a condition morphism m are sometimes denoted (v_m, I_m, N_m). □

Note that the root arrow v_m of a morphism $m : c_1 \to c_2$ goes in the *opposite* direction of m, namely from the root of c_2 to the root of c_1. In general, there are no requirements on the relation I of m. Moreover, for every $n \in N_{jk}$, the following n-based subdiagram emerges, which for now does not have to satisfy any constraints — in particular, it does not have to commute.

n-based subsquare

$$\begin{array}{ccc} R_1 & \xleftarrow{v} & R_2 \\ {\scriptstyle a^1_j}\downarrow & & \downarrow{\scriptstyle a^2_k} \\ P^1_j & \xrightarrow{v_n} & P^2_k \end{array} \tag{3}$$

However, in practice we restrict to two subclasses of morphisms that satisfy further properties (defined in Sec. 3.1 below), both on I and on the induced subsquares of (3). First, however, we introduce identities and composition of morphisms and show their expected properties.

Definition 3.2 (identity morphisms and composition)**.** Let $c = (R, (a_1, b_1) \cdots (a_w, b_w)) \in \mathsf{NC}(R)$ be a condition; then $id_c : c \to c$ is defined by

$$id_c = (id_R, Id_{[1,w]}, \{N_{jj} \triangleq \{id_{b_j}\} \mid (j,j) \in Id_{[1,w]}\}) \ . \tag{4}$$

Let $c_i = (R_i, (a^i_1, b^i_1) \cdots (a^i_{w_i}, b^i_{w_i})) \in \mathsf{NC}(R_i)$ for $i = 1, 2, 3$ and let $m_i = (v_i, I_i, N^i) : c_i \to c_{i+1}$ be condition morphisms for $i = 1, 2$. Then $m_1; m_2 : c_1 \to c_3$ is defined by

$$\begin{aligned} m_1; m_2 = (v_2; v_1, I_1; I_2, \{N_{ik} \triangleq \{m; n : b^3_k \to b^1_i \mid \\ \exists j \in [1, w_2]. m : b^3_k \to b^2_j \in N^2_{jk} \wedge n : b^2_j \to b^1_i \in N^1_{ij}\} \mid (i,k) \in I_1; I_2\}) \ . \end{aligned} \tag{5}$$

□

Proposition 3.3. *Condition morphism composition is associative, with left and right identities id_c.*

To conclude this subsection, observe that the potential morphisms between two given conditions are finite, even if their number can be very large. In principle it is possible to design an algorithm that enumerates all potential morphisms between two conditions. Also, by exploting the results of the next section, some of those morphisms could witness the fact that one condition entails the other. However, as discussed in the concluding section we don't expect that this approach can be competitive with existing methods for checking entailment.

3.1. Reflective and preservative morphisms

We specialize here the general notion of morphism in order to capture entailment between conditions. *Preservative* morphisms will witness entailment from the source to the target condition, and *reflective* ones similarly, but in the other direction. We obtain this by constraining the relations between the branches of the conditions and by imposing certain properties on the resulting squares formed by the arrows of the morphism and the branches of the conditions, as in (3). The properties are defined in terms of *reflection squares* and *preservation squares*, which are defined below. Intuitively, such properties ensure that the morphism preserves the satisfaction of branches in the expected direction.

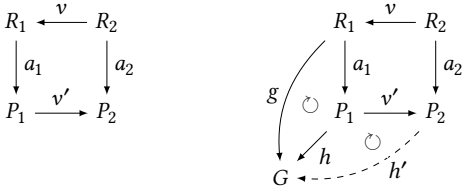
Definition 3.4 (reflection square)**.** Let $a_1 : R_1 \to P_1$, $a_2 : R_2 \to P_2$, $v : R_2 \to R_1$ and $v' : P_1 \to P_2$, and consider the (not necessarily commuting) diagram on the left:

$$\begin{array}{ccc} R_1 & \xleftarrow{v} & R_2 \\ \downarrow{\scriptstyle a_1} & & \downarrow{\scriptstyle a_2} \\ P_1 & \xrightarrow{v'} & P_2 \end{array} \qquad \begin{array}{ccc} R_1 & \xleftarrow{v} & R_2 \\ {\scriptstyle g'}\swarrow\ \downarrow{\scriptstyle a_1} & & \downarrow{\scriptstyle a_2} \\ P_1 & \xrightarrow{v'} & P_2 \\ & {\scriptstyle h'}\searrow & \\ & G' & \end{array} \tag{6}$$

The diagram is called a *reflection square* if for any pair g', h' that makes the outer subdiagram to the right commute (i.e., such that $v; g' = a_2; h'$), its bottom left subdiagram commutes (i.e., $g' = a_1; v'; h'$). □

Therefore a reflection square guarantees that if the branch a_2 satisfies $v; g'$ with witness h', then a_1 potentially satisfies g' with witness $v'; h'$. The dual to a reflection square is a *preservation square*, which ensures that if a_1 satisfies g with witness h then a_2 can potentially satisfy $v; g$ with some witness h' such that $h = v'; h'$:

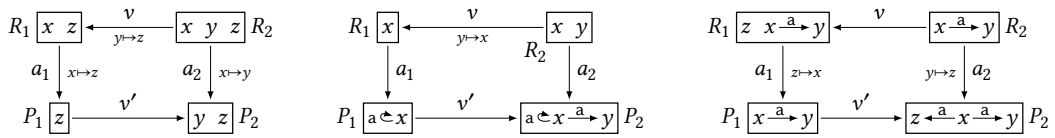
Definition 3.5 (preservation square)**.** Let $a_1 : R_1 \to P_1$, $a_2 : R_2 \to P_2$, $v : R_2 \to R_1$ and $v' : P_1 \to P_2$, and consider the (not necessarily commuting) diagram on the left:



(7)

The diagram is called a *preservation square* if for any pair of arrows g, h such that $g = a_1; h$, there is an arrow h' such that $h = v'; h'$ and $v; g = a_2; h'$. □

Example 3.6 (reflection and preservation squares). Consider the following (non-commuting) diagrams:



(8)

The left hand side and middle diagrams are reflection squares, whereas the right hand side is a preservation square. (This will follow from Props. 3.12 and 3.14 below; see Exs. 3.13 and 3.15).

The underlying principle for a morphism m from c_1 to c_2 that reflects satisfaction is that it must identify, for every c_2-branch p_i^2, a c_1-branch p_j^1 that it entails. Therefore we require the relation between branches to induce a function from c_2-branches to c_1-branches. Additionally, the subsquares formed by the root arrows and the branches of c_1 and c_2 (see (3)) must be reflection squares. Dually, for a morphism that preserves satisfaction, we require a function from c_1-branches to c_2-branches, and the subsquares should be preservation squares.

Definition 3.7 (reflective and preservative morphisms)**.** Let $c_i = (R_i, p_1^i \cdots p_{w_i}^i) \in \mathsf{NC}(R_i)$ for $i = 1, 2$, and let $m = (v, I, N) : c_1 \to c_2$ be a condition morphism.

- m is called *reflective* if the relation $I \subseteq [1, w_1] \times [1, w_2]$ is a function from c_2-branches to c_1-branches (i.e., $\forall k \in [1, w_2].\, \exists!\ j \in [1, w_1].\, (j, k) \in I$), and for all $(j, k) \in I$ it holds that $N_{jk} = \{n\}$ for some reflection morphism n such that the n-based subsquare (see (3)) is a reflection square;
- m is called *preservative* if relation I is a function from c_1-branches to c_2-branches (i.e., $\forall j \in [1, w_1].\, \exists!\ k \in [1, w_2].\, (j, k) \in I$), and for all $(j, k) \in I$ it holds that $N_{jk} = \{n\}$ for some preservation morphism n such that the n-based subsquare is a preservation square. □

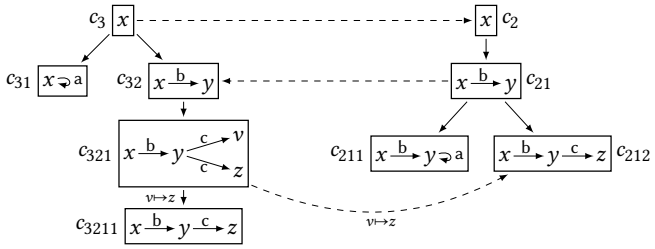
Note that if $dp(c_2) = 0$ [resp. $dp(c_1) = 0$] for the target [resp. source] of some reflective [preservative] condition morphism $m : c_1 \to c_2$, then $N = \emptyset$, hence the requirement on the family N is vacuously fulfilled. We will show later in Prop. 3.16 that reflective and preservative morphisms compose.

We now come to the core property of reflective and preservative morphisms, namely that they guarantee a one-way correspondence between the models satisfying their source and target conditions: from target to source for reflective morphisms, and from source to target for preservative ones.

Proposition 3.8 (reflective morphisms reflect satisfaction)**.** *Let $c_i \in \mathsf{NC}(R_i)$ be conditions for $i = 1, 2$. If $m : c_1 \to c_2$ is a reflective condition morphism, then $v_m; g \vDash c_2$ implies $g \vDash c_1$ for all arrows $g : R_1 \to G$.*

The following example shows a case in which a reflection morphism can indeed serve as a proof of entailment between conditions, but also a case in which there is entailment but no reflection morphism.

Example 3.9. In Ex. 2.7 we have argued that there exist entailments $c_1 \models c_2$ and $c_2 \models c_3$ for the conditions of Fig. 2. In fact, there exists a reflective morphism $m : c_2 \to c_3$, depicted by the dashed arrows in the following diagram, which by Prop. 3.8, given that $v_m = id$, establishes $c_2 \models c_3$:



In this case, the two subdiagrams that need to be reflection squares both commute and have identities as their top arrow; it follows from Prop. 3.12.3 below that they are indeed reflection squares (*simple* ones, in terms of that proposition).

There does *not*, however, exist a morphism from c_2 to c_1 that establishes the entailment $c_1 \models c_2$, even though that entailment holds. To see this, note that such a morphism would have to include arrows from both $c_{111} = \boxed{x \xleftarrow{b} \xrightarrow{a} y}$ and $c_{112} = \boxed{x \xleftarrow{b} \xrightarrow{c} y}$ to either $c_{211} = \boxed{x \xrightarrow{b} y \circlearrowright a}$ or $c_{212} = \boxed{x \xrightarrow{b} y \xrightarrow{c} z}$ (see Fig. 2), which obviously do not exist. □

Proposition 3.10 (preservative morphisms preserve satisfaction)**.** *Let $c_i \in \mathsf{NC}(R_i)$ be conditions for $i = 1, 2$. If $m : c_1 \to c_2$ is a preservative condition morphism, then $g \models c_1$ implies $v_m; g \models c_2$ for all arrows $g : R_1 \to G$.*

Thus, both reflective and preservative morphisms provide proof techniques for entailment: in particular, every root-preserving reflective morphism $m : c_1 \to c_2$ as well as every root-preserving preservative morphism $m : c_2 \to c_1$ constitutes evidence for $c_2 \models c_1$. However, it turns out that reflective morphisms are strictly more powerful than preservative morphisms, in the following sense.

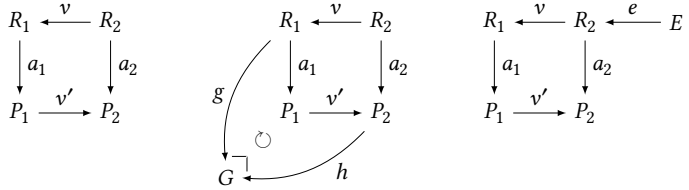
Proposition 3.11 (Preservative morphism inversion)**.** *If $m : c \to c'$ is a preservative morphism of which v_m is split mono, then there is a reflective morphism $m' : c' \to c$ such that, moreover, $m; m' = id_c$.*

3.2. Other properties of squares and morphisms

We present here some useful characterizations of reflection and preservation squares, as well as some properties that are used, for example, to show that reflective and preservative morphisms compose.

Proposition 3.12 (reflection squares)**.**

1. *The left hand diagram of (9) is a reflection square if and only if, for the pushout g, h of the span v, a_2 as drawn in the middle diagram, the bottom left subdiagram commutes, i.e., $g = a_1; v'; h$.*

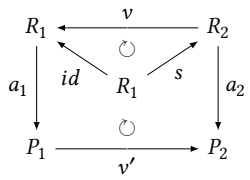


(9)

The left hand diagram of (9) is called a simple reflection square *if there is an equalising arrow e as in the right hand diagram (i.e., such that $e; a_2 = e; v; a_1 v'$) such that $e; v$ is epi.*

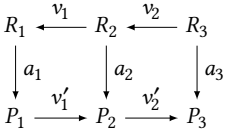
2. *Every simple reflection square is a reflection square.*
3. *If the left hand diagram of (9) commutes (i.e., $a_2 = v; a_1; v'$) and v is epi, then it is a simple reflection square.*
4. *In a simple reflection square, if e is the equaliser of a_2 and $v; a_1; v'$ then $e; v$ is epi.*

5. *In the outer diagram of* (10)*, if v is split epi with section s such that $s; a_2 = a_1; v'$, then the diagram is a simple reflection square.*



(10)

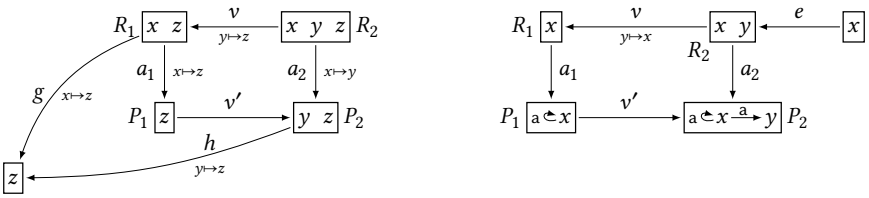
6. *Simple reflection squares compose; i.e., in*



(11)

if the left and right hand subdiagrams are simple reflection squares, then so is the outer subdiagram.
7. *In a reflection square, if the upper arrow v is an isomorphism, then the diagram commutes.*
8. *Reflection squares compose; i.e., in* (11)*, if both the left hand and the right hand subdiagrams are reflection squares, then so is the outer subdiagram.*
9. *In a reflection square as in* (6)*, the arrow v is epi. (Proof due to Daniel Schepler [16].)*

Example 3.13. To see that the first diagram shown in Ex. 3.6 (left hand side of (8)) is a reflection square, note that in the following (left hand) diagram, where g, h form a pushout for v, a_2, we have $g = a_1; v'; h$ and Prop. 3.12.1 applies. It is not a *simple* reflection square because the equaliser e of $v; a_1; v'$ and a_2 maps a single-node discrete graph to node z of R_2, hence $e; v$ is not epi.

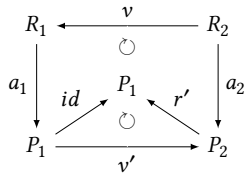


The second diagram of Ex. 3.6 (centre of (8)), repeated on the right hand above, is a *simple* reflection square: the equalizer e of $v; a_1; v'$ and a_2 is such that $e; v$ is epi. □

The following proposition states some useful facts regarding preservation squares.

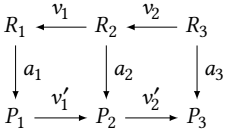
Proposition 3.14 (preservation squares)**.**

1. *In the left hand diagram of* (7)*, the diagram is a preservation square if and only if v' is a split mono with a retract r' (meaning $v'; r' = id_{P_1}$) such that the upper pentagon in* (12) *commutes (meaning $a_2; r' = v; a_1$).*



(12)

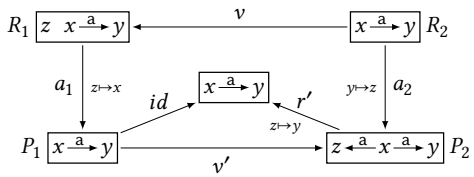
2. *In the left hand diagram of* (7)*, if the bottom arrow v' is iso then the diagram is a preservation square if and only if it commutes.*
3. *Preservation squares compose; i.e., in*



(13)

if the left and right hand subdiagrams are preservation squares, then so is the outer subdiagram.

Example 3.15. To see that the diagram shown in Ex. 3.6 (right hand side of (8), repeated below) is a preservation square, note that r' is a retract of v' and $v; a_1 = a_2; r'$; hence Prop. 3.14.1 applies.



□

We exploit the results just introduced to show that reflective and preservative morphisms compose.

Proposition 3.16. *Let $c \in \mathsf{NC}(R)$ and $c_i \in \mathsf{NC}(R_i)$ for $i = 1, 2, 3$ and let $m_i : c_i \to c_{i+1}$ be condition morphisms for $i = 1, 2$.*

1. *id_c is a reflective condition morphism, and if m_1 and m_2 are reflective then so is $m_1; m_2$.*
2. *id_c is a preservative condition morphism, and if m_1 and m_2 are preservative then so is $m_1; m_2$.*

4. Categorical properties

In this section, we frame the concepts and results introduced in the previous ones in a categorical setting, presenting several categories of conditions and functors relating them. This allows us to show the functoriality of the operations on conditions introduced in Sec. 2, with a few exceptions, and to characterize some of them as universal constructions.

As mentioned earlier, in the literature, morphisms among conditions are usually based on the semantic concept of entailment. Entailment (see Def. 2.6) establishes a preorder on conditions over the same root, and it can be generalized to entailment of conditions over different roots related by an arrow in two different ways.

Definition 4.1 (categories based on entailment)**.** Given conditions $c_i \in \mathsf{NC}(R_i)$ for $i = 1, 2$ and an arrow $f : R_1 \to R_2$ of $\mathbf{C}$, define $\leq^f$ as

$$c_1 \leq^f c_2 \text{ iff for all } g : R_2 \to G : f; g \vDash c_1 \text{ implies } g \vDash c_2$$

Category $\mathbf{NC}^{\vDash}$ has conditions (over arbitrary roots) as objects. An arrow from c_1 to c_2 is a triple $(c_1, f : R_{c_1} \to R_{c_2}, c_2)$ such that $c_1 \leq^f c_2$. Identities and composition are defined in the expected way.

Dually, for $c_i \in \mathsf{NC}(R_i)$ for $i = 1, 2$ and an arrow $f : R_2 \to R_1$, define $\leq_f$ as

$$c_1 \leq_f c_2 \text{ iff for all } g : R_1 \to G : g \vDash c_1 \text{ implies } f; g \vDash c_2$$

Category $\mathbf{NC}_{\vDash}$ has conditions as objects; an arrow from c_1 to c_2 is a triple $(c_1, f : R_{c_2} \to R_{c_1}, c_2)$ such that $c_1 \leq_f c_2$. Composition is defined as $(c_1, f, c_2); (c_2, g, c_3) = (c_1, g; f, c_3)$. □

The entailment preorders among conditions over the same root of Def. 2.6 are subcategories of both $\mathbf{NC}^{\vDash}$ and $\mathbf{NC}_{\vDash}$.

Fact 1 (entailment preorder)**.** *Let $\mathbf{NC}^{\vDash}(R)$ be the subcategory of $\mathbf{NC}^{\vDash}$ containing as objects the conditions over R and as arrows only triples of the form (c_1, id_R, c_2). Let $\mathbf{NC}_{\vDash}(R)$ be defined similarly as a subcategory of $\mathbf{NC}_{\vDash}$. Then $\mathbf{NC}^{\vDash}(R) = \mathbf{NC}_{\vDash}(R)$, and both coincide with the preorder of semantic entailment of Def. 2.6, i.e. (c_1, id_R, c_2) iff $c_1 \vDash c_2$.* □

We introduce now the categories of conditions based on the structural morphism proposed in Sec. 3.

Definition 4.2 (categories of structural morphisms)**.** The following categories are defined as having conditions over arbitrary roots as objects, and

- **NC** has as arrows condition morphisms as defined in Def. 3.1;
- $\mathbf{NC}^{\leftarrow}$ and $\mathbf{NC}^{\rightarrow}$, subcategories of **NC**, have as arrows reflective and preservative morphisms, as defined in Def. 3.7.

Identities and composition are defined as in Def. 3.2.

Given an object R, $\mathbf{NC}(R)$ is the subcategory of **NC** where objects are conditions over R, and arrows are morphisms m of which $v_m = id_R$. $\mathbf{NC}^{\leftarrow}(R)$ and $\mathbf{NC}^{\rightarrow}(R)$ are defined similarly. □

The well-definedness of the three categories just introduced follows from Props. 3.3 and 3.16.

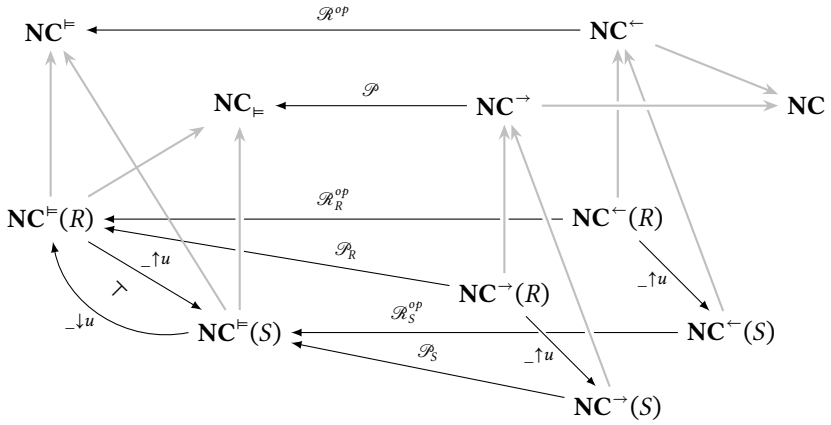


Figure 4: Overview of the functors of Prop. 4.3 and Prop. 4.4 (5-7). Gray arrows indicate inclusions of subcategories. The upshift and downshift functors are relative to an arrow $u : S \to R$ of $\mathbf{C}$

.

Functors. The various categories of conditions are related by various, mostly obvious, functors, depicted in Fig. 4.

Proposition 4.3 (relating categories of conditions)**.** *The followings are identity-on-object functors:*

1. $\mathscr{R} : \mathbf{NC}^{\leftarrow op} \to \mathbf{NC}^{\vDash}$, *contravariant, mapping* $(v, I, N) : c_1 \to c_2$ *to* (c_2, v, c_1)*;*
2. $\mathscr{P} : \mathbf{NC}^{\rightarrow} \to \mathbf{NC}_{\vDash}$, *mapping* $(v, I, N) : c_1 \to c_2$ *to* (c_1, v, c_2)*;*
3. $\mathscr{R}_R : \mathbf{NC}^{\leftarrow}(R)^{op} \to \mathbf{NC}^{\vDash}(R)$, *contravariant, restriction of* $\mathscr{R}$ *to* $\mathbf{NC}^{\leftarrow}(R)$*;*
4. $\mathscr{P}_R : \mathbf{NC}^{\rightarrow}(R) \to \mathbf{NC}_{\vDash}(R)[= \mathbf{NC}^{\vDash}(R)]$, *restriction of* $\mathscr{P}$ *to* $\mathbf{NC}^{\leftarrow}(R)$*.*

Proposition 3.11 shows that for each preservative morphism $m : c_1 \to c_2$ having a split-mono root arrow, there is a reflective morphism $m' : c_2 \to c_1$ such that $m; m' = id_{c_1}$. However the construction involves the choice of a retract at each level of the morphism, thus the result is not uniquely determined. This implies that even restricting to conditions over the same objects with the identity as root arrow (which is split-mono) we don't obtain a proper contravariant functor from $\mathbf{NC}^{\rightarrow}(R)$ to $\mathbf{NC}^{\leftarrow}(R)$. There are two possible directions to model this situation formally: resorting to a suitable coherent choice of retractions, making the construction deterministic and ensuring functoriality, or using a weaker notion of functor that allows to map an arrow to a set of arrows. We leave this as a topic of future work.

4.1. Functoriality and universality of operations on conditions

The categories of conditions just introduced allow us to analyse some categorical properties of the constructions on conditions introduced in Sec. 2. The next proposition summarizes the main universal and functoriality properties.

Proposition 4.4 (functoriality of operations)**.**

1. *Condition* $\bot_R \triangleq (R, \epsilon)$ *is initial in* $\mathbf{NC}^{\vDash}(R)$ *and in* $\mathbf{NC}^{\rightarrow}(R)$*, and terminal in* $\mathbf{NC}^{\leftarrow}(R)$*.*
2. *The inverse operator* $!$ *induces a contravariant functor* $!_R : \mathbf{NC}^{\vDash}(R)^{op} \to \mathbf{NC}^{\vDash}(R)$*.*
3. *The operator* $!$ *also induces a contravariant endofunctor on* $\mathbf{NC}(R)$*, which restricts to corresponding contravariant endofunctors on the subcategories* $\mathbf{NC}^{\leftarrow}(R)$ *and* $\mathbf{NC}^{\rightarrow}(R)$*.*
4. $!_R$ *commutes with functors* $\mathscr{R}_R$ *and* $\mathscr{P}_R$*, i.e.,* $!_R; \mathscr{R}_R = \mathscr{R}_R; !_R : \mathbf{NC}^{\leftarrow}(R)^{op} \to \mathbf{NC}^{\vDash}(R)$*, and* $!_R; \mathscr{P}_R = \mathscr{P}_R; !_R : \mathbf{NC}^{\rightarrow}(R)^{op} \to \mathbf{NC}^{\vDash}(R)$*.*
5. *For* $u : S \to R$*, the upshift operator* $_{\uparrow}u : \mathsf{NC}(R) \to \mathsf{NC}(S)$ *of Def. 2.11 is always functorial. More precisely, if* $c_1, c_2 \in \mathsf{NC}(R)$ *and* $k : c_1 \to c_2$ *is an arrow in* $\mathbf{NC}^{\vDash}(R), \mathbf{NC}(R), \mathbf{NC}^{\leftarrow}(R)$ *or* $\mathbf{NC}^{\rightarrow}(R)$*, then* $k{\uparrow}u : c_1{\uparrow}u \to c_2{\uparrow}u$ *is an arrow in* $\mathbf{NC}^{\vDash}(S), \mathbf{NC}(S), \mathbf{NC}^{\leftarrow}(S)$ *or* $\mathbf{NC}^{\rightarrow}(S)$*, respectively.*
6. *For* $v : R \to S$*, the downshift operator* $_{\downarrow}v : \mathsf{NC}(R) \to \mathsf{NC}(S)$ *of Def. 2.12 is functorial on* $\mathbf{NC}^{\vDash}(R)$*, but it is not functorial on* $\mathbf{NC}(R)$*,* $\mathbf{NC}^{\leftarrow}(R)$ *or* $\mathbf{NC}^{\rightarrow}(R)$*.*
7. *For* $u : R \to S$*, functor* $_{\uparrow}u : \mathbf{NC}^{\vDash}(S) \to \mathbf{NC}^{\vDash}(R)$ *is left adjoint to* $_{\downarrow}u : \mathbf{NC}^{\vDash}(R) \to \mathbf{NC}^{\vDash}(S)$*, i.e., for all* $c \in \mathbf{NC}^{\vDash}(R)$ *and* $b \in \mathbf{NC}^{\vDash}(S)$ *it holds*

$$b \vDash c{\downarrow}u \iff b{\uparrow}u \vDash c$$

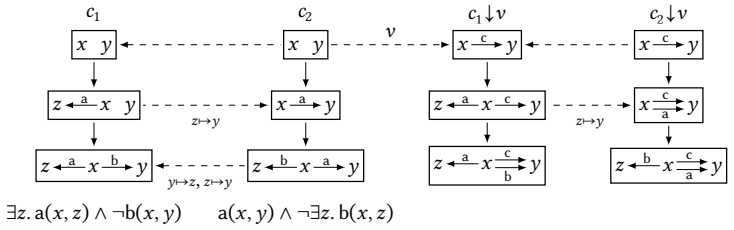


Figure 5: Condition downshift is not functorial: there is a morphism from c_1 to c_2 but not from $c_1{\downarrow}f$ to $c_2{\downarrow}f$

.

8. *The join operator (concatenation of branches)* $\sqcup : \mathsf{NC}(R) \times \mathsf{NC}(R) \to \mathsf{NC}(R)$ *of Def. 2.15 is a categorical coproduct in* $\mathbf{NC}^{\models}(R)$*, both product and coproduct in* $\mathbf{NC}(R)$*, product in* $\mathbf{NC}^{\leftarrow}(R)$*, and coproduct in* $\mathbf{NC}^{\rightarrow}(R)$*. It is functorial in both arguments in all these categories.*
9. *The meet operator* $\sqcap : \mathsf{NC}(R) \times \mathsf{NC}(R) \to \mathsf{NC}(R)$ *of Def. 2.17 is a product in* $\mathbf{NC}^{\models}(R)$*, but it is not functorial in the other categories, due to non-functoriality of downshift (point 6 above).*

Example 4.5. To see that downshift is not functorial (Prop. 4.4.6), consider the conditions on the left hand of Fig. 5. The dashed arrows between c_1 and c_2 together establish a valid reflective morphism $m : c_1 \to c_2$ with the identity as top arrow, establishing $c_2 \models c_1$. However, between the downshifted versions $c_1{\downarrow}v$ and $c_2{\downarrow}v$ (where the arrow v is the one shown in the figure), no morphism exists: on the third level, there is no arrow from $\boxed{z \xleftarrow{\mathrm{b}} x \underset{\mathrm{a}}{\overset{\mathrm{c}}{\rightrightarrows}} y}$ to $\boxed{z \xleftarrow{\mathrm{a}} x \underset{\mathrm{b}}{\overset{\mathrm{c}}{\rightrightarrows}} y}$. □

5. Conclusion, Related and Future work

The main contributions of this paper can be summarized as follows:

- After presenting (nested) conditions as in [5], in Sec. 2 we introduce several operations on them showing the induced logical meaning on the corresponding FOL formulas. Such operators are shown to be complete.
- In Sec. 3 we propose an original notion of structural morphisms among conditions, which are shown to be composable and to have identities. We identify two subclasses of morphisms, the *reflective* and the *preservative* ones, which are shown to reflect and preserve satisfaction, respectively, and thus witness entailment between the formulas associated with the related conditions. We also show that, under mild assumptions, for each preservative morphism there is a reflective one in the opposite direction.
- In Sec. 4 we define several categories having conditions as objects and as arrows the various kinds of morphisms defined in Sec. 3. These are related with suitable identity-on-object functors to the preorder of entailment among conditions. We also show some functoriality and universality properties of the constructions on conditions of Sec. 2 with respect to these categories.

Discussion. The work reported here was primarily motivated by curiosity: given that entailment in the ∃-fragment of FOL (see Sec. 1) can be fully characterised by structural morphisms between graphs, we try to push this idea by considering, not graphs, but nested conditions, which (over **Graph**) are known to be expressively equivalent to full FOL. If successful, the resulting extended morphism would provide an understanding of *why* one condition entails another; this can then potentially be manipulated and give rise to new forms of reasoning. One concrete case of this has serendipitously presented itself in the guise of *Craig interpolants*, which we speculate (in the future work discussion below) might emerge from our structural morphisms. If one is interested in entailment checking per se, we do not think that searching for a morphism between two given conditions is likely to be superior to existing methods — certainly not based on the precise form of morphism studied in this paper, which we have observed to be rather weak. (How weak precisely is again one of the questions we ask ourselves in the future work discussion.)

One observation that has been made multiple times (see the PhD theses of Pennemann [17] and Wallentin [18]) is that the use of state-of-the-art satisfiability checkers such as Z3 to the FOL translation of nested condition yields poorer performance than (academic) tools that work directly on the nested (graph) structure itself. To our knowledge, this phenomenon has not been studied in depth and so one should be careful in drawing conclusions; nevertheless, it suggests that the graph-based structure of nested conditions might actually have some performance advantages. The investigation in this paper can be seen as a contribution to the theoretical framework built on top of that graph-based structure.

Related work. We made precise along the paper the relationship between our morphisms and the preorder of entailment among conditions presented in [7]. Unfortunately, space constraints do not allow us to discuss in detail the relationship with our previous work [8]. Anyway, we can explain at and abstract level why the morphisms proposed here are simpler and more general than those of [8]. In that paper we assumed that morphisms could relate only conditions over the same root, consistently with the standard notion of entailment (Def. 2.6). This led to some complications, as in the inductive definition even if two conditions have the same root, two related subconditions (branch conditions) could have different roots, and these were reconciled with the use of so-called "shifters". In this paper, instead, we defined morphisms immediately between conditions over different roots (but related by an arrow) obtaining a simpler definition. We recovered the constraint on entailment by requiring the top arrow of a morphisms to be an identity, and we also generalized the standard notion of entailment for conditions having different roots (Def. 4.1).

As a matter of fact, in [8] we also introduced a different kind of conditions, so-called *span-based* ones, which were there posed as a variant that in principle allows more powerful reasoning. A natural question is whether the morphisms defined here for arrow-based conditions can be generalized to span-based ones. Likewise, the same question can be asked for other variants of nested conditions proposed in the literature, such as those of [6] and of [7].

Future work. The expressive power of the reflective morphisms we defined is quite limited, in the sense that they only explain a small fragment of entailment among formulas. A question that we intend to address in the future is whether there is any independent characterisation of the fragment of entailment that is explained by our condition morphisms and, connected to this, whether the proposed morphisms can be generalized to explain a larger fragment of entailment.

Another topic we plan to explore, inspired by discussions with Barbara König, is related to the existence of *Craig interpolants* for implications among formulas induced by conditions. Craig interpolants play an important role in improving abstraction-based reasoning (cf. [19]), and being able to characterise them in the case of nested conditions could therefore strengthening the method studied in [20]. In particular, we are interested in the relationship between the existence of a morphism between two conditions and the existence of an interpolant for the corresponding implication.

Finally, let us observe that in [7, 15] nested conditions are defined over an arbitrary category $\mathbf{C}$, which allows to instantiate the framework beyond presheaf toposes: for example, to the category of graphs and injective morphisms, or to the category of left-linear cospans of an adhesive category. In some cases where pushouts do not (sufficiently) exist, their role can be taken over by (finite) *sets of representative squares*. We intend to explore to what extent our assumptions on category $\mathbf{C}$ can be relaxed in order to be able to apply our results to other, more general settings.

Acknowledgements. We would like to thank Filippo Bonchi for providing the original inspiration for this work, and together with Nicolas Behr and Barbara König for providing comments in earlier stages; and Nicholas and Barbara again for a series of inspiring brainstorm sessions on nested conditions.

Declaration on Generative AI. The authors have not employed any Generative AI tools.

References

[1] A. K. Chandra, P. M. Merlin, Optimal implementation of conjunctive queries in relational data bases, in: J. E. Hopcroft, E. P. Friedman, M. A. Harrison (Eds.), STOC, ACM, 1977, pp. 77–90. doi:10.1145/800105.803397.

[2] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, Monographs in Theoretical Computer Science, Springer, 2006. doi:10.1007/3-540-31188-2.

[3] R. Heckel, G. Taentzer, Graph Transformation for Software Engineers - With Applications to Model-Based Development and Domain-Specific Language Engineering, Springer, 2020. doi:10.1007/978-3-030-43916-3.

[4] A. Habel, R. Heckel, G. Taentzer, Graph grammars with negative application conditions, Fundamenta Informaticae 26 (1996) 287–313. doi:10.3233/FI-1996-263404.

[5] A. Rensink, Representing first-order logic using graphs, in: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), ICGT, LNCS, Springer, 2004. doi:10.1007/978-3-540-30203-2_23.

[6] A. Habel, K.-H. Pennemann, Correctness of high-level transformation systems relative to nested conditions, Math. Struct. Comput. Sci. 19 (2009) 245–296. doi:10.1017/S0960129508007202.

[7] H. S. Bruggink, R. Cauderlier, M. Hülsbusch, B. König, Conditional reactive systems, in: S. Chakraborty, A. Kumar (Eds.), FSTTCS, volume 13 of *LIPIcs*, Leibniz-Zentrum für Informatik, 2011, pp. 191–203. doi:10.4230/LIPICS.FSTTCS.2011.191.

[8] A. Rensink, A. Corradini, On categories of nested conditions, in: N. Jansen, S. Junges, B. L. Kaminski, C. Matheja, T. Noll, T. Quatmann, M. Stoelinga, M. Volk (Eds.), Principles of Verification: Cycling the Probabilistic Landscape, Part I, LNCS, Springer, 2024, pp. 393–418. doi:10.1007/978-3-031-75783-9_16.

[9] S. M. Lane, I. Moerdijk, Sheaves in Geometry and Logic: A First Introduction to Topos Theory, Springer, 1992. doi:10.1007/978-1-4612-0927-0.

[10] S. Lack, P. Sobociński, Adhesive and quasiadhesive categories, RAIRO Theor. Informatics Appl. 39 (2005) 511–545. doi:10.1051/ITA:2005028.

[11] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, A. Corradini, Algebraic approaches to graph transformation II: single pushout approach and comparison with double pushout approach, in: G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 1: Foundations, World Scientific, 1997. doi:https://doi.org/10.1142/3303.

[12] G. G. Azzi, A. Corradini, L. Ribeiro, On the essence and initiality of conflicts in M-adhesive transformation systems, J. Log. Algebraic Methods Program. 109 (2019). doi:10.1016/J.JLAMP.2019.100482.

[13] A. Anjorin, E. Leblebici, A. Schürr, 20 years of triple graph grammars: A roadmap for future research, Electronic Communications of the EASST 73 (2016). doi:10.14279/tuj.eceasst.73.1031.

[14] K. Pennemann, Development of correct graph transformation systems, Ph.D. thesis, University of Oldenburg, Germany, 2009. URL: https://nbn-resolving.org/urn:nbn:de:gbv:715-oops-9483.

[15] L. Stoltenow, B. König, S. Schneider, A. Corradini, L. Lambers, F. Orejas, Coinductive techniques for checking satisfiability of generalized nested conditions, in: R. Majumdar, A. Silva (Eds.), CONCUR, LIPIcs, Leibniz-Zentrum für Informatik, 2024, pp. 1–39. doi:10.4230/LIPICS.CONCUR.2024.39.

[16] D. Schepler, Does the existence of a commuting diagonal in a pushout imply that one leg is epi?, Answer on math.stackexchange.com, 2026.

[17] K.-H. Pennemann, Development of Correct Graph Transformation Systems, Ph.D. thesis, Universität Oldenburg, 2009.

[18] L. Wallentin, Verifying Graph Rewriting — Coinductive Techniques for Nested Conditions, Ph.D. thesis, University Duisburg-Essen, Germany, 2026.

[19] T. A. Henzinger, R. Jhala, R. Majumdar, K. L. McMillan, Abstractions from proofs, in: POPL, 2004, pp. 232–244. doi:10.1145/964001.964021.

[20] B. König, A. Rensink, L. Stoltenow, F. Urrigshardt, Counterexample-guided abstraction refinement for generalized graph transformation systems, in: J. Endrullis, M. Tichy (Eds.), ICGT, LNCS, Springer, 2025, pp. 157–176. doi:10.1007/978-3-031-94706-3_8.