

Coinductive Techniques for Checking Satisfiability of Generalized Nested Conditions

Lara Stoltenow ✉ 

University of Duisburg-Essen, Germany

Barbara König ✉ 


University of Duisburg-Essen, Germany

Sven Schneider ✉ 


Hasso Plattner Institute at the University of Potsdam, Germany

Andrea Corradini ✉ 

Università di Pisa, Italy

Leen Lambers ✉ 

Brandenburgische Technische Universität Cottbus-Senftenberg, Germany

Fernando Orejas ✉ 

Universitat Politècnica de Catalunya, Spain

Abstract

We study nested conditions, a generalization of first-order logic to a categorical setting, and provide a tableau-based (semi-decision) procedure for checking (un)satisfiability and finite model generation. This generalizes earlier results on graph conditions. Furthermore we introduce a notion of witnesses, allowing the detection of infinite models in some cases. To ensure completeness, paths in a tableau must be fair, where fairness requires that all parts of a condition are processed eventually. Since the correctness arguments are non-trivial, we rely on coinductive proof methods and up-to techniques that structure the arguments. We distinguish between two types of categories: categories where all sections are isomorphisms, allowing for a simpler tableau calculus that includes finite model generation; in categories where this requirement does not hold, model generation does not work, but we still obtain a sound and complete calculus.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Program reasoning

Keywords and phrases satisfiability, graph conditions, coinductive techniques, category theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.39

Related Version *Full Version*: <https://arxiv.org/abs/2407.06864> [28]

Funding *Andrea Corradini*: Research partially supported by the Italian MUR under the PRIN 20228KXFN2 “STENDHAL” and by the University of Pisa under the PRA 2022_99 “FM4HD”.

Fernando Orejas: Research partially supported by MCIN/ AEI /10.13039/501100011033 under grant PID2020-112581GB-C21.

1 Introduction

Nested graph conditions (called graph conditions subsequently) are a well-known specification technique for graph transformation systems [8] where they are used, e.g., to specify graph languages and application conditions. While their definition is quite different from first-order logic (FOL), they have been shown to be equivalent to FOL in [23, 8]. They are naturally equipped with operations such as shift, a form of partial evaluation, which is difficult to specify directly in FOL. This operation can be used to compute weakest preconditions and strongest postconditions for graph transformation systems [1].



© Lara Stoltenow, Barbara König, Sven Schneider, Andrea Corradini, Leen Lambers, and Fernando Orejas;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 39; pp. 39:1–39:20

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In [1] it has also been observed that graph conditions can be generalized to the categorical setting of reactive systems [16] as an alternative to the previously considered instantiation to graphs and injective graph morphisms that is equivalent to FOL. Further possible instantiations include cospan categories where the graphs, equipped with an inner and an outer interface, are the arrows, as well as Lawvere theories. To derive analysis techniques for all such instantiations, we consider nested conditions in the general categorical setting.

Here we are in particular interested in satisfiability checks on the general categorical level. As in FOL, satisfiability can be an undecidable problem (depending on the category), and we propose a semi-decision procedure that can simultaneously serve as a model finder. For FOL there are several well-known methods for satisfiability checking, for instance resolution or tableau proofs [6], while model generation is typically performed separately. The realization that satisfiability checking is feasible directly on graph conditions came in [18, 19], and a set of tableau rules was presented [18] without a proof of (refutational) completeness that was later published in [13], together with a model generation procedure [26]. A generalization to non-injective graph morphisms was given in [17].

The contributions of the current paper can be summarized as follows:

- We generalize the tableau-based semi-decision procedure for graph conditions from [13] to the level of general categories, under some mild constraints (such as the existence of so-called representative squares [1]). We present a procedure that has some resemblance to the construction of a tableau in FOL.
- We distinguish between two cases: one simpler case in which all sections (arrows that have a right inverse) in the category under consideration are isomorphisms (Section 3); and a more involved case where this does not necessarily hold (Section 5). The tableau rules of the former case (Section 3) are easier to present and implement, and we can give additional guarantees, such as model generation whenever there exists a so-called finitely decomposable model, generalizing the notion of finite models. The latter case (Section 5) does not guarantee model generation and has more involved tableau rules, but it allows for instantiations to more categories, such as graphs and arbitrary morphisms. The results of both cases generalize [13, 17, 26] from graphs and graph morphisms to an abstract categorical level, which allows application to additional categories such as cospan categories and Lawvere theories (see [28]).
- The completeness argument for the satisfiability checking procedure – in particular showing that non-termination implies the existence of an infinite model – requires that the tableau construction satisfies a fairness constraint. The resulting proof is non-trivial and – compared to the proof in [13] – we show that it can be reformulated using up-to techniques. Here we give it a completely new and hopefully clarifying structure that relies on coinductive methods [20, 22]. The alternative would be to inline the up-to techniques, or to rely on complex ad-hoc notation that are less clear and further complicate the proof.
- Furthermore we use coinductive techniques to display witnesses for infinite models (Section 4): in some cases where only infinite models exist and hence the tableau construction is non-terminating, we can still stop and determine that there does exist an infinite model. Coinductive techniques [24, 22] are reasoning techniques based on greatest fixpoints, suitable to analyze infinite or cyclic structures. To the best of our knowledge, such techniques have not yet been employed in the context of satisfiability checking for FOL and graph conditions.

The main contribution compared to previous work consists of a categorical generalization to reactive systems on the one hand, and the use of coinductive (up-to) techniques on the other hand. The implication of the first type of contribution is that the theory becomes available

for new instantiations such as adhesive categories (which includes all variants of graphs, such as typed graphs, Petri nets, but also algebraic specifications, cf. [3]), as well as other cases such as cospan categories and Lawvere theories. The second type of contribution implies that the proofs (especially for completeness) can now be presented in a more systematic way.

2 Preliminaries

2.1 Coinductive Techniques

A *complete lattice* is a partially ordered set (L, \sqsubseteq) where each subset $Y \subseteq L$ has a greatest lower bound, denoted by $\bigsqcap Y$ and a least upper bound, denoted by $\bigsqcup Y$.

A function $f: L \rightarrow L$ is *monotone* if for all $l_1, l_2 \in L$, $l_1 \sqsubseteq l_2$ implies $f(l_1) \sqsubseteq f(l_2)$, *idempotent* if $f \circ f = f$, and *extensive* if $l \sqsubseteq f(l)$ for all $l \in L$.

Given a monotone function $f: L \rightarrow L$ we are in particular interested in its *greatest fixpoint* νf . By Tarski's Theorem [29], $\nu f = \bigsqcup \{x \mid x \sqsubseteq f(x)\}$, i.e., the greatest fixpoint is the least upper bound of all post-fixpoints. Hence for showing that $l \sqsubseteq \nu f$ (for some $l \in L$), it is sufficient to prove that l is below some post-fixpoint l' , i.e., $l \sqsubseteq l' \sqsubseteq f(l')$.

In order to employ up-to techniques one defines a monotone function $u: L \rightarrow L$ (the up-to function) and checks whether u is *f-compatible*, that is $u \circ f \sqsubseteq f \circ u$. If that holds every post-fixpoint l of $f \circ u$ (that is $l \sqsubseteq f(u(l))$) is below the greatest fixpoint of f ($l \sqsubseteq \nu f$). This simple technique can often greatly simplify checking whether a given element is below the greatest fixpoint. For more details see [20].

2.2 Categories

We will use standard concepts from category theory. Given an arrow $f: A \rightarrow B$, we write $\text{dom}(f) = A$, $\text{cod}(f) = B$. For two arrows $f: A \rightarrow B$, $g: B \rightarrow C$ we denote their composition by $f;g: A \rightarrow C$. An arrow $s: A \rightarrow B$ is a *section* (also known as *split mono*) if there exists $r: B \rightarrow A$ such that $s;r = \text{id}$. That is, sections are those arrows s that have a right-inverse r . Arrows that have a left-inverse (in this case r) are called *retractions*.

As in graph rewriting we will consider the category $\mathbf{Graph}_{\text{fin}}$, which has finite graphs as objects and graph morphisms as arrows. We also consider $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$, the subcategory of $\mathbf{Graph}_{\text{fin}}$ that has the same objects, but only injective, i.e. mono, graph morphisms. In this category the sections are exactly the isos, while this is not the case in $\mathbf{Graph}_{\text{fin}}$.

Another important example category that will be used in Section 4 is based on cospans: note that reactive systems instantiated with cospans [11, 25, 27] yield exactly double-pushout rewriting [5]. Given a base category \mathbf{D} with pushouts, the category $\mathbf{Cospan}(\mathbf{D})$ has as objects the objects of \mathbf{D} and as arrows *cospans*, which are equivalence classes of pairs of arrows of the form $A \xrightarrow{f_L} X \xleftarrow{f_R} B$, where the middle object is considered up to isomorphism. Cospan composition is performed via pushouts (for details see Appendix A).

A cospan is *left-linear* if its left leg f_L is mono. For adhesive categories [12], the composition of left-linear cospans again yields a left-linear cospan, and $\mathbf{ILC}(\mathbf{D})$ is the subcategory of $\mathbf{Cospan}(\mathbf{D})$ where the arrows are restricted to left-linear cospans.

Note that $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ can be embedded into $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}})$ by transforming an injective graph morphism f to a left-linear cospan with f as the left leg and id as the right leg.

Another application are Lawvere theories, where arrows are (tuples of) terms, an approach we explore in the full version of this paper [28].

2.3 Generalized Nested Conditions

We consider (nested) conditions over an arbitrary category \mathbf{C} in the spirit of reactive systems [16, 15]. Following [23, 8], we define conditions as finite tree-like structures, where nodes are annotated with quantifiers and objects, and edges are annotated with arrows.

► **Definition 1 (Condition).** *Let \mathbf{C} be a category. A condition \mathcal{A} over an object A in \mathbf{C} is defined inductively as follows: it is either*

- *a finite conjunction of universals $\bigwedge_{i \in \{1, \dots, n\}} \forall f_i. \mathcal{A}_i = \forall f_1. \mathcal{A}_1 \wedge \dots \wedge \forall f_n. \mathcal{A}_n$, or*
 - *a finite disjunction of existentials $\bigvee_{i \in \{1, \dots, n\}} \exists f_i. \mathcal{A}_i = \exists f_1. \mathcal{A}_1 \vee \dots \vee \exists f_n. \mathcal{A}_n$*
- where $f_i: A \rightarrow A_i$ are arrows in \mathbf{C} and $\mathcal{A}_i \in \text{Cond}(A_i)$ are conditions. We call $A = \text{RO}(\mathcal{A})$ the root object of the condition \mathcal{A} . Each subcondition $\mathcal{Q}f_i. \mathcal{A}_i$ ($\mathcal{Q} \in \{\forall, \exists\}$) is called a child of \mathcal{A} . The constants true_A (empty conjunction) and false_A (empty disjunction) serve as the base cases. We will omit subscripts in true_A and false_A when clear from the context.

The set of all conditions over A is denoted by $\text{Cond}(A)$.

Instantiated with $\mathbf{Graph}_{\text{fin}}$ respectively $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$, conditions are equivalent to graph conditions as defined in [8], and equivalence to first-order logic has been shown in [23]. Intuitively, these conditions require the existence of certain subgraphs or patterns, or that whenever a given subgraph occurs, the surroundings of the match satisfy a child condition. For instance, $\forall \emptyset \rightarrow \textcircled{1} \rightarrow \textcircled{2} . \exists \textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{1} \leftarrow \textcircled{2} . \text{true}$ requires that for every edge, a second edge in the reverse direction also exists. For additional examples of conditions we refer to Examples 19, 24, and 27 given later.

To simplify our algorithms and their proofs, the definition of conditions requires that conjunctions contain only universal children and disjunctions only existential children (e.g., $\exists f. \mathcal{A} \wedge \exists g. \mathcal{B}$ is excluded). However, this can be simulated using $\forall \text{id}. \exists f. \mathcal{A} \wedge \forall \text{id}. \exists g. \mathcal{B}$, and similarly for disjunctions of universals. Hence we sometimes write $\mathcal{A} \wedge \mathcal{B}$ or $\mathcal{A} \vee \mathcal{B}$ for arbitrary conditions in the proofs.

While in [1] a model for a condition was a single arrow, we have to be more general, since there are some satisfiable conditions that have no finite models. Here we want to work in categories of finite graphs (so that conditions are finite), but at the same time we want to consider infinite models. The solution is to evaluate conditions on infinite sequences of arrows $\bar{a} = [a_1, a_2, a_3, \dots]$, where $A \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \dots$, called *composable sequences*.¹ We define $\text{dom}(\bar{a}) = \text{dom}(a_1) = A$ and we call such a sequence *finite* iff for some index k all a_i with $i > k$ are identities.

Intuitively, the model is represented by the “composition” of the infinite sequence of arrows. In the category $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ this would amount to taking the limit of this sequence. As we will later see, it does not play a role how exactly an infinite structure is decomposed into arrows, as all decompositions are equivalent with respect to satisfaction.

► **Definition 2 (Satisfaction).** *Let $\mathcal{A} \in \text{Cond}(A)$. Let $\bar{a} = [a_1, a_2, a_3, \dots]$ be a composable sequence with $A = \text{dom}(\bar{a})$. We define the satisfaction relation $\bar{a} \models \mathcal{A}$ as follows:*

- $\bar{a} \models \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$ iff for every $i \in I$ and every arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and all $n \in \mathbb{N}_0$ we have: if $a_1; \dots; a_n = f_i; g$, then $[g, a_{n+1}, \dots] \models \mathcal{A}_i$.
- $\bar{a} \models \bigvee_i \exists f_i. \mathcal{A}_i$ iff there exists $i \in I$ and an arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and some $n \in \mathbb{N}_0$ such that $a_1; \dots; a_n = f_i; g$ and $[g, a_{n+1}, \dots] \models \mathcal{A}_i$.

¹ Another option would be to work in the category of potentially infinite graphs. However, that would allow conditions based on infinite graphs for which satisfiability checks become algorithmically infeasible.

Note that this covers the base cases ($\bar{a} \models \text{true}$, $\bar{a} \not\models \text{false}$ for every sequence \bar{a}). Furthermore $a_1; \dots; a_n$ equals the identity whenever $n = 0$. For a finite sequence $\bar{a} = [a_1, \dots, a_k, \text{id}, \text{id}, \dots]$ this means that $\bar{a} \models \mathcal{A}$ iff $a = a_1; \dots; a_k$ is a model for \mathcal{A} according to the definition of satisfaction given in [1]. In this case we write $[a_1, \dots, a_k] \models \mathcal{A}$ or simply $a \models \mathcal{A}$.

► **Remark 3.** In the following we use **Cond** to denote the set² of all conditions and **Seq** as the set of all composable sequences of arrows (potential models). ◻

We write $\mathcal{A} \models \mathcal{B}$ (\mathcal{A} implies \mathcal{B}) if $\text{RO}(\mathcal{A}) = \text{RO}(\mathcal{B})$ and for every \bar{a} with $\text{dom}(\bar{a}) = \text{RO}(\mathcal{A})$ we have: if $\bar{a} \models \mathcal{A}$, then $\bar{a} \models \mathcal{B}$. Two conditions are equivalent ($\mathcal{A} \equiv \mathcal{B}$) if $\mathcal{A} \models \mathcal{B}$ and $\mathcal{B} \models \mathcal{A}$.

Every condition can be transformed to an equivalent condition that alternates between \forall, \exists by inserting $\forall \text{id}$ or $\exists \text{id}$ as needed. Such conditions are called *alternating*.

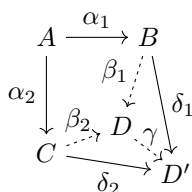
We also define what it means for two conditions to be isomorphic. It is easy to see that isomorphic conditions are equivalent, but not necessarily vice versa.

► **Definition 4 (Isomorphic Conditions).** For conditions \mathcal{A}, \mathcal{B} and an iso $h: \text{RO}(\mathcal{B}) \rightarrow \text{RO}(\mathcal{A})$, we say that \mathcal{A}, \mathcal{B} are isomorphic ($\mathcal{A} \cong \mathcal{B}$) wrt. h , whenever both are universal, i.e., $\mathcal{A} = \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$, $\mathcal{B} = \bigwedge_{j \in J} \forall g_j. \mathcal{B}_j$, and for each $i \in I$ there exists $j \in J$ and an iso $h_{j,i}: \text{RO}(\mathcal{B}_j) \rightarrow \text{RO}(\mathcal{A}_i)$ such that $h; f_i = g_j; h_{j,i}$ and $\mathcal{A}_i \cong \mathcal{B}_j$ wrt. $h_{j,i}$; and vice versa (for each $j \in J$ there exists $i \in I \dots$). Analogously if both conditions are existential.

2.4 Representative Squares and the Shift Operation

We will now define the notion of representative squares, which describe representative ways to close a span of arrows. They generalize idem pushouts [16] and borrowed context diagrams [4].

► **Definition 5 (Representative squares [1]).** A class κ of commuting squares in a category \mathbf{C} is representative if for every commuting square $\alpha_1; \delta_1 = \alpha_2; \delta_2$ in \mathbf{C} there exists a representative square $\alpha_1; \beta_1 = \alpha_2; \beta_2$ in κ and an arrow γ such that $\delta_1 = \beta_1; \gamma$ and $\delta_2 = \beta_2; \gamma$.



For two arrows $\alpha_1: A \rightarrow B$, $\alpha_2: A \rightarrow C$, we define $\kappa(\alpha_1, \alpha_2)$ as the set of pairs of arrows (β_1, β_2) which, together with α_1, α_2 , form representative squares in κ .

Compared to weak pushouts, more than one square might be needed to represent all commuting squares that extend a given span (α_1, α_2) . In categories with pushouts (such as $\mathbf{Graph}_{\text{fin}}$), pushouts are the most natural candidate for representative squares. In $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ pushouts do not exist, but jointly epi squares can be used instead. For cospan categories, one can use borrowed context diagrams [4] (see Appendix A for a summary).

For many categories of interest – such as $\mathbf{Graph}_{\text{fin}}$ and $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}})$ – we can guarantee a choice of κ such that each set $\kappa(\alpha_1, \alpha_2)$ is finite and computable. In the rest of this paper, we assume that we work in such a category, and use such a class κ . Hence the constructions described below are effective since the finiteness of the transformed conditions is preserved.

² Actually, without restrictions these are proper classes rather than sets. We tacitly assume that we are working in the corresponding skeleton category where no two different objects are isomorphic and assume that we can consider **Cond**, **Seq** as sets.

One central operation is the shift of a condition along an arrow. The name shift is taken from an analogous operation for nested application conditions (see [19]).

► **Definition 6** (Shift of a Condition). *Given a fixed class of representative squares κ , the shift of a condition \mathcal{A} along an arrow $c: \text{RO}(\mathcal{A}) \rightarrow B$ is inductively defined as follows:*

$$\left(\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \right)_{\downarrow c} = \bigwedge_{i \in I} \bigwedge_{(\alpha, \beta) \in \kappa(f_i, c)} \forall \beta. (\mathcal{A}_i)_{\downarrow \alpha}$$

Shifting of existential conditions is performed analogously.

The shift operation can be understood as a partial evaluation of \mathcal{A} under the assumption that c is already “present”. In particular it satisfies $[c; d_1, d_2, \dots] \models \mathcal{A} \iff [d_1, d_2, \dots] \models \mathcal{A}_{\downarrow c}$. This implies that while the representation of the shifted condition may differ depending on the chosen class of representative squares, the resulting conditions are equivalent. Since we assume that each set $\kappa(f_i, c)$ is finite, shifting a finite condition will again result in a finite condition.

As an example in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$, shifting $\forall \emptyset \rightarrow \textcircled{1}. \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}. \text{true}$ (every node has an outgoing edge) over $\emptyset \rightarrow \textcircled{1}$ (a node exists) yields $\forall \textcircled{1} \rightarrow \textcircled{1}. \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}. \text{true} \wedge \forall \textcircled{1} \rightarrow \textcircled{1}. (\exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}). \text{true} \vee \exists \textcircled{1} \rightarrow \textcircled{1}. \text{true}$ (the designated node has an outgoing edge, and so does every other node, possibly to the designated node).

In the case where α_1 in Definition 5 is an iso, we can always assume that $\kappa(\alpha_1, \alpha_2) = \{(\alpha_1^{-1}; \alpha_2, \text{id})\}$ and we will use this assumption in the paper.

2.5 Further Concepts

Our goal is to develop a procedure that finds a finite model if one exists, produces unsatisfiability proofs if a condition has neither finite nor infinite models, and otherwise does not terminate. In order to state the correctness of this procedure, we will need an abstract notion of finiteness and to this aim we introduce *finitely decomposable morphisms*. Intuitively this means that every infinite decomposition contains only finitely many non-isomorphisms.

► **Definition 7** (Finitely decomposable morphism). *A morphism $m: A \rightarrow B$ is finitely decomposable if for every infinite sequence of (f_i, g_i) , $i \in \mathbb{N}_0$, such that $f_0 = m$ and $f_i = g_i; f_{i+1}$ (cf. the diagram below), only finitely many g_i are non-isomorphisms.*

$$\begin{array}{c} A \xrightarrow{g_0} \xrightarrow{g_1} \dots \\ m = f_0 \downarrow \swarrow f_1 \quad \searrow f_2 \\ B \end{array}$$

Note that in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ all arrows are finitely decomposable, while this is not the case in $\mathbf{Graph}_{\text{fin}}$. In $\mathbf{Graph}_{\text{fin}}$, there exists a section s (with associated retraction r such that $s; r = \text{id}$) that is *not* an iso (example: $s = \textcircled{1} \rightarrow \textcircled{1} \textcircled{2}$, $r = \textcircled{1} \textcircled{2} \rightarrow \textcircled{1}$). Then, the identity on the domain of s has a decomposition into infinitely many non-isos (an alternating sequence of s and r , more concretely: $g_{2i} = s$, $g_{2i+1} = r$ and $f_{2i} = \text{id}$, $f_{2i+1} = r$) and is hence not finitely decomposable.

While satisfaction is typically defined inductively (as in Definition 2), i.e., as a least fixpoint, we can also view it coinductively, i.e., as a greatest fixpoint, due to the fact that all conditions are finite.

► **Proposition 8** (Fixpoint function for satisfaction). *Let $\bar{a} = [a_1, a_2, a_3, \dots] \in \text{Seq}$ be a composable sequence of arrows. We define the function $s: \mathcal{P}(\text{Seq} \times \text{Cond}) \rightarrow \mathcal{P}(\text{Seq} \times \text{Cond})$ as follows: Let $P \subseteq \text{Seq} \times \text{Cond}$, then*

- $(\bar{a}, \bigwedge_i \forall f_i. \mathcal{A}_i) \in s(P)$ iff for every $i \in I$ and every arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and all $n \in \mathbb{N}_0$ we have: if $a_1; \dots; a_n = f_i; g$, then $([g, a_{n+1}, \dots], \mathcal{A}_i) \in P$.
- $(\bar{a}, \bigvee_i \exists f_i. \mathcal{A}_i) \in s(P)$ iff there exists $i \in I$ and an arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and some $n \in \mathbb{N}_0$ such that $a_1; \dots; a_n = f_i; g$ and $([g, a_{n+1}, \dots], \mathcal{A}_i) \in P$.

The least and greatest fixpoint of s ($\mu s, \nu s$) coincide and they equal the satisfaction relation \models .

3 Satisfiability Checking in the Restricted Case

Given a condition \mathcal{A} , we want to know whether \mathcal{A} is satisfiable and generates a finitely decomposable model, if it exists. Here we provide a procedure that works under the assumption that we are working in a category where all sections are isos. This is for instance true for $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ and $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, where non-trivial right-inverses do not exist. It does not hold for non-injective graph morphisms (see counterexample above) or left-linear cospans (counterexample: $\text{id} = \textcircled{1} \rightarrow \textcircled{1} \leftarrow \textcircled{1} \textcircled{2}$; $\textcircled{1} \textcircled{2} \rightarrow \textcircled{1} \textcircled{2} \leftarrow \textcircled{1}$).

The general case where this assumption does not hold will be treated in Section 5.

3.1 Tableau Calculus

The underlying idea is fairly straightforward: we take an alternating condition \mathcal{A} and whenever it is existential, that is $\mathcal{A} = \bigvee_{i \in I} \exists f_i. \mathcal{A}_i$, we branch and check whether some \mathcal{A}_i is satisfiable. If instead it is universal, i.e., $\mathcal{A} = \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$, we check whether some f_i is an iso. If that is not the case, clearly the sequence of identities on $\text{RO}(\mathcal{A})$ is a model, since there is no arrow g such that $\text{id} = f_i; g$, assuming that all sections are isos. If however some f_i is an iso, we invoke a pull-forward rule (see below for more details) that transforms the universal condition into an existential condition and we continue from there. We will show that this procedure works whenever the pull-forward follows a *fair* strategy: in particular every iso (respectively one of its successors) must be pulled forward eventually.

The pull-forward rule relies on the equivalence $(\mathcal{A} \wedge \exists f. \mathcal{B}) \equiv \exists f. (\mathcal{A} \downarrow_f \wedge \mathcal{B})$.

► **Lemma 9** (Pulling forward isomorphisms). *Let $\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$ be a universal condition and assume for some $p \in I$, f_p is an iso and $\mathcal{A}_p = \bigvee_{j \in J} \exists g_j. \mathcal{B}_j$. Then f_p can be pulled forward:*

$$\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \equiv \exists f_p. \bigvee_{j \in J} \exists g_j. \left(\mathcal{B}_j \wedge \left(\bigwedge_{m \in I \setminus \{p\}} \forall f_m. \mathcal{A}_m \right) \downarrow_{f_p; g_j} \right)$$

► **Definition 10** (SatCheck tableau construction rules). *Given an alternating condition \mathcal{A} , we give rules for the construction of a tableau for \mathcal{A} that has conditions as nodes, \mathcal{A} as root node, and edges (\rightarrow) labeled with arrows. The tableau is extended at its leaf nodes as follows:*

$$\left. \begin{array}{l} \text{For every } p \in I: \\ \bigvee_{i \in I} \exists f_i. \mathcal{A}_i \xrightarrow{f_p} \mathcal{A}_p \end{array} \right| \begin{array}{l} \text{For one } p \in I \text{ such that } f_p \text{ is iso and } \mathcal{A}_p = \bigvee_{j \in J} \exists g_j. \mathcal{B}_j: \\ \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \xrightarrow{f_p} \underbrace{\bigvee_{j \in J} \exists g_j. \mathcal{B}_j}_{\mathcal{A}_p} \wedge \underbrace{\left(\bigwedge_{m \in I \setminus \{p\}} \forall f_m. \mathcal{A}_m \right) \downarrow_{f_p; g_j}}_{\text{other children, shifted to include } g_j} \end{array}$$

- For existential conditions, for each(!) child condition $\exists f_p. \mathcal{A}_p$, add a new descendant.
- For universal conditions, non-deterministically pick one(!) child condition $\forall f_p. \mathcal{A}_p$ that can be pulled forward (f_p is an iso), pull it forward (cf. Lemma 9), and add the result as its (only) descendant. If a universal condition contains no isos, then add no descendant.

A branch of a tableau is a (potentially infinite) path $\mathcal{A}_0 \xrightarrow{u_1} \mathcal{A}_1 \xrightarrow{e_1} \mathcal{A}_2 \xrightarrow{u_2} \dots$ starting from the root node. A finite branch is extendable if one of the tableau construction rules is applicable at its leaf node and would result in new nodes (hence, a branch where the leaf is an empty existential or a universal without isomorphisms is not extendable). A branch is closed if it ends with an empty existential, otherwise it is open.

Due to Lemma 9 each universal condition is (up to iso f_p) equivalent to its (unique) descendant (if one exists), while an existential condition is equivalent to the disjunction of its descendants prefixed with existential quantifiers.

The labels along one branch of the tableau are arrows between the root objects of the conditions. Their composition corresponds to the prefix of a potential model being constructed step by step. Finite paths represent a model if they are open and not extendable. For infinite paths, we need an additional property to make sure that the procedure does not “avoid” a possibility to show unsatisfiability of a condition.

To capture that, we introduce the notion of fairness, meaning that all parts of a condition are eventually used in a proof and are not postponed indefinitely (a related concept is saturation, see e.g. [13], though the definition deviates due to a different setup). For this we first need to track how pulling forward one child condition changes the other children by shifting. We define a *successor relation* that, for each pair of \forall -condition and one of its \forall -grandchildren, relates child conditions of the \forall -condition to their shifted counterparts (the *successors*) in the \forall -condition of the second-next nesting level. The successor relation is similar in spirit to the one used in [13]. In this work, saturation is given in a more descriptive way and has to account for nesting levels in the tableau, a complication that we were able to avoid in the present paper.

► **Definition 11** (Successor relation). *Assume in the construction of a tableau we have a path*

$$\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \xrightarrow{f_p} \mathcal{C} \xrightarrow{g_j} \mathcal{B}_j \wedge \left(\bigwedge_{m \in I \setminus \{p\}} \forall f_m. \mathcal{A}_m \right) \downarrow_{f_p; g_j} = \mathcal{B}_j \wedge \bigwedge_{m \in I \setminus \{p\}} \bigwedge_{(\alpha, \beta) \in \kappa(f_m, f_p; g_j)} \forall \beta. (\mathcal{A}_m)_{\downarrow \alpha}$$

where \mathcal{C} is the existential condition given in Definition 10. Then for each $m \in I \setminus \{p\}$, each $\forall \beta. (\mathcal{A}_m)_{\downarrow \alpha}$ where $(\alpha, \beta) \in \kappa(f_m, f_p; g_j)$ is a successor of $\forall f_m. \mathcal{A}_m$. The transitive closure of the successor relation induces the indirect successor relation.

► **Definition 12** (Fairness). *An infinite branch of a tableau is fair if for each universal condition \mathcal{A} on the branch and each child condition $\forall f_i. \mathcal{A}_i$ of \mathcal{A} where f_i is an iso, it holds that some indirect successor of $\forall f_i. \mathcal{A}_i$ is eventually pulled forward.*

► **Remark 13** (Fairness strategies). One possible strategy that ensures fairness is to maintain for each incomplete branch a queue of child conditions for which a successor must be pulled forward. Then the first entry in this queue is processed. Note that by the assumption on κ made earlier at the end of Section 2.4, each iso in a universal condition that is not pulled forward has exactly one successor and the queue is modified by replacing each condition accordingly and adding newly generated child conditions with isos at the end. ◻

3.2 Up-To Techniques, Fair Branches and Models

While showing soundness of the tableau method is relatively straightforward, the crucial part of the completeness proof is to show that every infinite and fair branch of the tableau corresponds to a model. The proof strategy is the following: given such a branch, we aim to construct a witness for this model, by pairing conditions on this path with the suffix

consisting of the sequence of arrows starting from this condition. If one could show that the set $P \subseteq \text{Seq} \times \text{Cond}$ of pairs so obtained is a post-fixpoint of the satisfaction function s defined in Proposition 8 ($P \subseteq s(P)$), we could conclude, as the satisfaction relation \models is the greatest fixpoint of s (Proposition 8) and hence above any post-fixpoint.

However, P is in general not a post-fixpoint, which is mainly due to the fact that universal conditions are treated “sequentially” one after another and are “pulled forward” only if they become isos. Hence, if we want to show that for a chain $[a_1, a_2, \dots]$ the universal condition of the form $\bigwedge_i \forall f_i. \mathcal{A}_i$ is satisfied, we have to prove for every child $\forall f_i. \mathcal{A}_i$ that whenever $a_1; \dots; a_n = f_i; g$ it holds that $[g, a_{n+1}, \dots] \models \mathcal{A}_i$. If $\forall f_i. \mathcal{A}_i$ actually is the child that is pulled forward in the next tableau step, P contains the tuple required by s . If not, there is a “delay” and intuitively that means that we can not guarantee that P is indeed a post-fixpoint.

However it turns out that it is a post-fixpoint up-to ($P \subseteq s(u(P))$), where u is a combination of one or more suitable up-to functions. We first explore several such up-to functions: $u_\wedge(P)$ obtains new conditions by non-deterministically removing parts of conjunctions; with $u_\S(P)$ we can arbitrarily recompose (decompose and compose) the arrows in a potential model; with $u_\downarrow(P)$ we can undo a shift; and $u_\cong(P)$ allows replacing conditions with isomorphic conditions. For each, we show their s -compatibility (i.e., $u(s(P)) \subseteq s(u(P))$).

► **Theorem 14 (Up-to techniques).** *Let $P \subseteq \text{Seq} \times \text{Cond}$, i.e. tuples of potential model and condition. Then the following four up-to functions are s -compatible:*

- *Conjunction removal: We inductively define a relation \mathcal{U}_\wedge containing a pair of conditions $(\mathcal{A}, \mathcal{T})$ iff \mathcal{T} is the same as \mathcal{A} but with some conjunctions removed. That is, \mathcal{U}_\wedge contains*

$$\begin{aligned} & (\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i, \bigwedge_{j \in J \subseteq I} \forall f_j. \mathcal{T}_j) \quad \text{whenever } (\mathcal{A}_j, \mathcal{T}_j) \in \mathcal{U}_\wedge \text{ for all } j \in J \\ & (\bigvee_{i \in I} \exists f_i. \mathcal{A}_i, \bigvee_{i \in I} \exists f_i. \mathcal{T}_i) \quad \text{whenever } (\mathcal{A}_i, \mathcal{T}_i) \in \mathcal{U}_\wedge \text{ for all } i \in I \end{aligned}$$

Then define: $u_\wedge(P) = \{(\bar{c}, \mathcal{T}) \mid (\bar{c}, \mathcal{A}) \in P, (\mathcal{A}, \mathcal{T}) \in \mathcal{U}_\wedge\}$

- *Recomposition: $u_\S(P) = \{([b_1, \dots, b_\ell, \bar{c}], \mathcal{A}) \mid ([a_1, \dots, a_k, \bar{c}], \mathcal{A}) \in P, a_1; \dots; a_k = b_1; \dots; b_\ell\}$*
- *Shift: $u_\downarrow(P) = \{([c; c_1, c_2, \dots], \mathcal{B}) \mid ([c_1, c_2, \dots], \mathcal{B}_{\downarrow c}) \in P\}$*
- *Isomorphic condition: $u_\cong(P) = \{([h; c_1, c_2, \dots], \mathcal{B}) \mid ([c_1, c_2, \dots], \mathcal{A}) \in P, \mathcal{A} \cong \mathcal{B} \text{ with iso } h: \text{RO}(\mathcal{B}) \rightarrow \text{RO}(\mathcal{A})\}$*

Note however that up-to equivalence u_\cong is *not* a valid up-to technique: let \mathcal{U} be an unsatisfiable condition and let $P = \{([\text{id}, \dots], \forall \text{id}. \mathcal{U})\}$. As $\mathcal{U} \equiv \forall \text{id}. \mathcal{U}$, then also $([\text{id}, \dots], \mathcal{U}) \in u_\cong(P)$ and hence $P \subseteq s(u_\cong(P))$. If the technique were correct, this would imply $\text{id} \models \mathcal{U}$.

A convenient property of compatibility is that it is preserved by various operations, in particular, composition, union and iteration ($f^\omega = \bigcup_{i \in \mathbb{N}_0} f^i$). This can be used to combine multiple up-to techniques into a new one that also has the compatibility property. [21]

► **Lemma 15 (combining up-to techniques).** *Let $u = (u_\S \cup u_\wedge \cup u_\downarrow \cup u_\cong)^\omega$ be the iterated application of the up-to techniques from Theorem 14, then u is s -compatible.*

We are now able to prove the central theorem needed for showing completeness.

► **Theorem 16 (Fair branches are models).** *Let \mathcal{A}_0 be an alternating condition. Let a fixed tableau constructed by the rules of Definition 10 be given. Let $\mathcal{A}_0 \xrightarrow{b_1} \mathcal{A}_1 \xrightarrow{b_2} \mathcal{A}_2 \xrightarrow{b_3} \dots$ be a branch of the tableau that is either not extendable and ends with a universal quantification (i.e., it is open), or is infinite and fair. For such a branch, we define $P = \{(\bar{b}, \mathcal{A}_i) \mid i \in \mathbb{N}_0, \bar{b} = [b_{i+1}, b_{i+2}, \dots]\} \subseteq \text{Seq} \times \text{Cond}$, i.e., the relation P pairs suffixes of the branch with the corresponding conditions. Finally, let u be the combination of up-to techniques defined in Lemma 15. Then, $P \subseteq s(u(P))$, which implies that $P \subseteq \models$. In other words, every such branch in a tableau of Definition 10 corresponds to a model of \mathcal{A}_0 .*

Proof sketch. Let $([c_1, c_2, \dots], \mathcal{C}_0) \in P$, which corresponds to a suffix $\mathcal{C}_0 \xrightarrow{c_1} \mathcal{C}_1 \xrightarrow{c_2} \mathcal{C}_2 \xrightarrow{c_3} \dots$ of the chosen branch. We show that $([c_1, c_2, \dots], \mathcal{C}_0) \in s(u(P))$:

- **\mathcal{C}_0 is existential:** The next label on the branch is the arrow of some child $\exists c_1.\mathcal{C}_1$ of \mathcal{C}_0 and $([c_2, c_3, \dots], \mathcal{C}_1) \in P$ implies $([c_1, c_2, \dots], \mathcal{C}_0) \in s(P) \subseteq s(u(P))$.
- **\mathcal{C}_0 is universal and contains no iso:** The branch ends at this point, so the sequence of arrows in the tuple is empty and represents id , where $(\text{id}, \mathcal{C}_0) \in s(u(P))$: no f_i is an iso, therefore $f_i; g = \text{id}$ is never true and hence the universal condition is trivially satisfied.
- **\mathcal{C}_0 is universal and contains at least one iso:** By definition of s , we need to be able to satisfy any given child $\forall d_0.\mathcal{D}_0$, i.e., show that whenever $c_1; \dots; c_n = d_0; g$ for some g, n , then $([g, c_{n+1}, \dots], \mathcal{D}_0) \in u(P)$.

Fairness guarantees that an indirect successor $\forall d_q.\mathcal{D}_q$ of $\forall d_0.\mathcal{D}_0$ is pulled forward, which results (after up-to conjunction removal) in a tuple $([c_m, \dots], \mathcal{D}_q) \in u(P)$. The intermediate steps on the branch, where other children are pulled forward instead, allow expressing \mathcal{D}_q as $(\mathcal{D}_0)_{\downarrow \alpha_1 \downarrow \dots \downarrow \alpha_q}$. Use up-to shift to transform to $([\alpha_1; \dots; \alpha_q; c_m, \dots], \mathcal{D}_0) \in u(P)$, then up-to recomposition to the required $([g_0, c_{n+1}, \dots], \mathcal{D}_0) \in u(P)$. ◀

3.3 Soundness and Completeness

We are finally ready to show soundness and completeness of our method.

As a condition is essentially equivalent to any of its tableaux, which break it down into existential subconditions, a closed tableau represents an unsatisfiable condition.

► **Theorem 17 (Soundness).** *If there exists a tableau T for a condition \mathcal{A} where all branches are closed, then the condition \mathcal{A} in the root node is unsatisfiable.*

Proof sketch. By induction over the depth of T . Base case is false (obviously unsatisfiable). Induction step for $\exists: \bigvee_i \exists f_i.\mathcal{A}_i$ is unsatisfiable if all \mathcal{A}_i are. For \forall : by construction, the only child contains an equivalent condition. ◀

We now prove completeness, which – to a large extent – is a corollary of Theorem 16.

► **Theorem 18 (Completeness).** *If a condition \mathcal{A} is unsatisfiable, then every tableau constructed by obeying the fairness constraint is a finite tableau where all branches are closed. Furthermore, at least one such tableau exists.*

Proof. The contraposition follows from Theorem 16: If the constructed tableau is finite with open branches or infinite, then \mathcal{A} is satisfiable. Furthermore, a fair tableau must exist and can be constructed by following the strategy described in Remark 13. ◀

In the next section we will show how the open branches in a fully expanded tableau can be interpreted as models, thus giving us a procedure for model finding.

► **Example 19 (Proving unsatisfiability).** We work in $\mathbf{Graph}_{\text{fn}}^{\text{inj}}$. For this example, we use the following shorthand notation for graph morphisms: $[\textcircled{1} \rightarrow \textcircled{2}]$ means $\textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}$, i.e., the morphism is the inclusion from the light-gray graph elements to the full graph.

Consider the condition $\mathcal{A} = \forall[\emptyset].\exists[\textcircled{1}].\text{true} \wedge \forall[\textcircled{2}].\text{false}$, meaning (1) there exists a node and (2) no node must exist. It is easily seen that these contradict each other and hence \mathcal{A} is unsatisfiable. We obtain a tableau with a single branch for this condition:

$$\mathcal{A} \xrightarrow{[\emptyset]} \exists[\textcircled{1}].(\text{true} \wedge (\forall[\textcircled{2}].\text{false})_{\downarrow[\textcircled{1}]}) \xrightarrow{[\textcircled{1}]} \forall[\textcircled{1}].\text{false} \wedge \forall[\textcircled{1} \rightarrow \textcircled{2}].\text{false} \xrightarrow{[\textcircled{1}]} \text{false}$$

In the first step, \mathcal{A} is universal with an isomorphism $\emptyset \rightarrow \emptyset$, which is pulled forward. Together with its (only) existential child, this results in the partial model $[\textcircled{\text{D}}]$ and another universal condition with the meaning: (1) the just created node $\textcircled{\text{D}}$ must not exist, and (2) no additional node $\textcircled{\text{X}}$ may exist either.

This condition includes another isomorphism ($[\textcircled{\text{D}}]$) to be pulled forward. Its child $\mathcal{A}_p = \bigvee_{j \in J} \exists g_j. \mathcal{B}_j = \text{false}$ is an empty disjunction, so the tableau rule for universals adds an empty disjunction (false) as the only descendant. This closes the (only) branch, hence the initial condition \mathcal{A} can be recognized as unsatisfiable. \lrcorner

3.4 Model Finding

We will now discuss the fact that the calculus not only searches for a logical contradiction to show unsatisfiability, but at the same time tries to generate a (possibly infinite) model. We can show that *every* finitely decomposable (i.e., “finite”) model (or a prefix thereof) can be found after finitely many steps in a *fully expanded* tableau, i.e., a tableau where all branches are extended whenever possible, including infinite branches. This is a feature that distinguishes it from other known calculi for first-order logic.

The following lemma shows that an infinite branch always makes progress towards approximating the infinite model.

► **Lemma 20.** *Let \mathcal{A} be a condition and T be a fully expanded tableau for \mathcal{A} . Then, for each branch it holds that it either is finite, or that there always eventually is another non-isomorphism on the branch.*

Proof sketch. We define the size of a condition and show that it decreases if an iso occurs on a path. This means that eventually there will always occur another non-iso on a path. ◀

► **Theorem 21 (Model Finding).** *Let \mathcal{A} be a condition, m a finitely decomposable arrow such that $m \models \mathcal{A}$ and let T be a fully expanded tableau for \mathcal{A} .*

Then, there exists an open and unextendable branch with arrows c_1, \dots, c_n in T , having condition \mathcal{R} in the leaf node, where $m = c_1; \dots; c_n; r$ for some r with $r \models \mathcal{R}$. Furthermore the finite prefix is itself a model for \mathcal{A} (i.e., $[c_1, \dots, c_n] \models \mathcal{A}$).

Note that this finite branch can be found in finite time, assuming a suitable strategy for exploration of the tableau such as breadth-first search or parallel processing.

► **Algorithm 22 (Satisfiability Check).** *Given a condition \mathcal{A} , we define the procedure $\text{SAT}(\mathcal{A})$ that may either produce a model $c: \text{RO}(\mathcal{A}) \rightarrow C$, answer unsat or does not terminate.*

- Initialize the tableau with \mathcal{A} in the root node.
- While the tableau still has open branches:
 - Select one of the open branches as the current branch, using an appropriate strategy that extends each open branch eventually.
 - If the leaf is a universal condition without isomorphisms, terminate and return the labels of the current branch as model.
 - Otherwise, extend the branch according to the rules of Definition 10, obeying the fairness constraint.
- If all branches are closed, terminate and answer unsat.

This procedure has some similarities to the tableau-based reasoning from [13]. The aspect of model generation was in particular considered in [26]. Overall, we obtain the following result:

► **Theorem 23.** *There is a one-to-one correspondence between satisfiability of a condition (\mathcal{A} unsatisfiable; \mathcal{A} has a finitely decomposable model; \mathcal{A} is satisfiable, but has no finitely decomposable model) and the output of Algorithm 22 ($SAT(\mathcal{A})$) (terminates with unsat, terminates with a model, does not terminate).*

Proof.

- \mathcal{A} unsatisfiable \iff algorithm outputs unsat: (\implies) Theorem 18, (\impliedby) Theorem 17
- \mathcal{A} has a finitely decomposable model \iff algorithm finds finitely decomposable model: (\implies) Theorem 21, (\impliedby) Theorem 16
- \mathcal{A} has only models that are not finitely decomposable \iff algorithm does not terminate: (\implies) exclusion of other possibilities for non-termination, Lemma 20 for model in the limit (\impliedby) Theorem 16 ◀

► **Example 24 (Finding finite models).** We now work in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ and use the shorthand notation introduced in Example 19. Let the following condition be given:

$$\begin{aligned} & \forall \emptyset \rightarrow \emptyset. \exists \emptyset \rightarrow \textcircled{1}. \text{true} && \text{(there exists a node } \textcircled{1}\text{)} \\ & \wedge \forall \emptyset \rightarrow \textcircled{1}. \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}. \text{true} && \text{and every node has an outgoing edge to some other node)} \end{aligned}$$

This condition has finite models, the smallest being the cycle $\textcircled{1} \leftrightarrow \textcircled{2}$. When running Algorithm 22 on this condition, it obtains the model in the following way:

1. The given condition is universal with an iso $\emptyset \rightarrow \emptyset$, which is pulled forward. Together with its (only) existential child, this results in the partial model $[\textcircled{1}]$ and the condition

$$\begin{aligned} & \text{true} \wedge (\forall [\textcircled{1}]. \exists [\textcircled{1} \rightarrow \textcircled{2}]. \text{true}) \downarrow_{[\textcircled{1}]} \\ & = \forall [\textcircled{1}]. \exists [\textcircled{1} \rightarrow \textcircled{2}]. \text{true} \wedge \forall [\textcircled{1} \text{ } \textcircled{A}]. \overbrace{(\exists [\textcircled{1} \text{ } \textcircled{A} \rightarrow \textcircled{B}]. \text{true} \vee \exists [\textcircled{1} \leftarrow \textcircled{A}]. \text{true})}^{\mathcal{B}} \end{aligned}$$

meaning: (1) the just created node $\textcircled{1}$ must have an outgoing edge; (2) and every other node A must also have an outgoing edge to either another node or to the existing node.

2. Pull forward iso $[\textcircled{1}]$ and extend the partial model by $[\textcircled{1} \rightarrow \textcircled{2}]$, resulting in:

$$\begin{aligned} & \text{true} \wedge (\forall [\textcircled{1} \text{ } \textcircled{A}]. \mathcal{B}) \downarrow_{[\textcircled{1} \rightarrow \textcircled{2}]} = \forall [\textcircled{1} \rightarrow \textcircled{2}]. \mathcal{B} \downarrow_{[\textcircled{1} \rightarrow \textcircled{A}]} \wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \text{ } \textcircled{A}]. \mathcal{B} \downarrow_{[\textcircled{1} \rightarrow \textcircled{2} \text{ } \textcircled{A}]} \\ & = \forall [\textcircled{1} \rightarrow \textcircled{2}]. (\exists [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3}]. \text{true} \vee \exists [\textcircled{1} \leftarrow \textcircled{2}]. \text{true}) \\ & \quad \wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \text{ } \textcircled{A}]. (\exists [\textcircled{1} \rightarrow \textcircled{2} \text{ } \textcircled{A} \rightarrow \textcircled{B}]. \text{true} \vee \exists [\textcircled{1} \rightarrow \textcircled{2} \leftarrow \textcircled{A}]. \text{true} \vee \exists [\textcircled{1} \rightarrow \textcircled{2} \text{ } \textcircled{A}]. \text{true}) \end{aligned}$$

meaning: (1) the second node has an edge to a third node or to the first one; (2) and every other node A also has an edge to either another node or to one of the existing nodes.

3. Next, we pull forward $[\textcircled{1} \rightarrow \textcircled{2}]$ and extend the model by $[\textcircled{1} \leftrightarrow \textcircled{2}]$:

$$\text{true} \wedge (\forall [\textcircled{1} \rightarrow \textcircled{2} \text{ } \textcircled{A}]. \dots) \downarrow_{[\textcircled{1} \leftrightarrow \textcircled{2}]} = \forall [\textcircled{1} \leftrightarrow \textcircled{2} \text{ } \textcircled{A}]. \dots$$

4. This condition does not have any children with isos, so it is satisfiable by id. Hence the composition of the partial models so far ($[\textcircled{1} \leftrightarrow \textcircled{2}]$) is a model for the original condition. \dashv

4 Witnesses for infinite models

If there is no finitely decomposable model for a satisfiable condition \mathcal{A} (such as in Example 27 below), then the corresponding infinite branch produces a model in the limit. To detect such models in finite time we introduce an additional use of coinductive techniques based on the tableau calculus previously introduced: We will show that under some circumstances, it is

possible to find some of these infinite models while checking for satisfiability. Naturally, one can not detect all models, since this would lead to a decision procedure for an undecidable problem. We first have to generalize the notion of fairness to finite path fragments:

► **Definition 25.** Let $\mathcal{C}_0 \xrightarrow{b_1} \mathcal{C}_1 \xrightarrow{b_2} \dots \xrightarrow{b_r} \mathcal{C}_r$ be a finite path (also called segment) in a tableau (cf. Definition 30). Such a finite path is called fair if for every child of \mathcal{C}_0 where the morphism is an iso, an indirect successor is pulled forward at some point in the path.

This notion of fairness does not preclude that new isos appear. It only states that all isos present at the beginning are pulled forward at some point.

► **Theorem 26 (Witnesses).** Let \mathcal{C}_0 be an alternating, universal condition. Let a fixed tableau constructed by the rules of Definition 10 be given. Let $\mathcal{C}_0 \xrightarrow{b_1} \mathcal{C}_1 \xrightarrow{b_2} \dots \xrightarrow{b_r} \mathcal{C}_r$ be a fair segment of a branch of the tableau where $r > 0$, and let some arrow m be given such that $\mathcal{C}_0 \cong (\mathcal{C}_r)_{\downarrow m}$ for an iso $\iota: \text{RO}(\mathcal{C}_{r\downarrow m}) \rightarrow \text{RO}(\mathcal{C}_0)$. Then, $[b_1, \dots, b_r, m; \iota]^\omega := [b_1, \dots, b_r, m; \iota, b_1, \dots, b_r, m; \iota, \dots]$ is a model for \mathcal{C}_0 .

Proof sketch. Construct the relation

$$P = \{([b_1, b_2, \dots, b_r, m; \iota, (b_1, \dots, b_r, m; \iota)^\omega], \mathcal{C}_0), \\ ([b_2, \dots, b_r, m; \iota, (b_1, \dots, b_r, m; \iota)^\omega], \mathcal{C}_1), \dots, ([m; \iota, (b_1, \dots, b_r, m; \iota)^\omega], \mathcal{C}_r)\}$$

and show that $P \subseteq s(u(P))$, using an approach similar to that of Theorem 16. Steps b_1, \dots, b_r are handled in the same way as in Theorem 16. For the newly introduced step based on $m; \iota$, the next element of the sequence of representative squares and the successor d_i are chosen from a child of $\mathcal{C}_{r\downarrow m}$ instead of from a successor of \mathcal{D}_i . ◀

► **Example 27 (Finding witnesses).** Consider the following condition:

$$\begin{aligned} \forall \emptyset \rightarrow \emptyset. \exists \emptyset \rightarrow \textcircled{1}. \forall \textcircled{1} \rightarrow \textcircled{\rightarrow} \textcircled{1}. \text{false} & \quad (\text{there is a node } \textcircled{1} \text{ without an incoming edge} \\ \wedge \forall \emptyset \rightarrow \textcircled{\rightarrow} \textcircled{\rightarrow} \textcircled{\oplus}. \text{true} & \quad \text{and every node has an outgoing edge to some other node} \\ \wedge \forall \emptyset \rightarrow \textcircled{\rightarrow} \textcircled{\leftarrow} \textcircled{2}. \text{false} & \quad \text{and no node has two incoming edges}) \end{aligned}$$

This condition has an infinite model, namely an infinite path $(\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3} \rightarrow \dots)$. It does not have any finite model.

In order to display a witness for this model, we need to consider the condition in the category of cospans $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, into which $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ can be embedded. We use the following shorthand notation for cospans: $\llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket$ means $\textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2} \leftarrow \textcircled{1} \rightarrow \textcircled{2}$, i.e., the left object consists of only the light-gray graph elements, the center and right objects consist of the full graph, the left leg is the inclusion and the right leg is always the identity.

If we execute our algorithm on the condition, after one step we obtain a condition \mathcal{C}_0 that is rooted at $\textcircled{1}$, and spells out the requirements of the original condition for node $\textcircled{1}$ and all other nodes separately ($\textcircled{1}$ has a successor, and all other nodes have successors, and so on).

After another step, we obtain \mathcal{C}_1 , which is rooted at $\textcircled{1} \rightarrow \textcircled{2}$, and does the same for node $\textcircled{2}$ separately as well. (These steps are displayed more concretely in the appendix, Table 1.)

Now let $m = \textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{1} \rightarrow \textcircled{2} \leftarrow \textcircled{2}$. Then, we can compute $(\mathcal{C}_1)_{\downarrow m}$, which essentially “forgets” node $\textcircled{1}$ from all subconditions of \mathcal{C}_1 . (Subconditions that contain edges from or to this node disappear entirely, which is a consequence of the way borrowed context diagrams are constructed (via pushout complements).) Then, $(\mathcal{C}_1)_{\downarrow m}$ is similar in structure to \mathcal{C}_0 , and in fact, using a renaming isomorphism $\iota = \textcircled{2} \rightarrow \textcircled{\rightarrow} \textcircled{\leftarrow} \textcircled{1}$, it holds that $(\mathcal{C}_1)_{\downarrow m} \cong \mathcal{C}_0$ wrt. ι . ◻

In general this witness construction will almost never be applicable for simple graph categories, we need to work in other categories, such as cospan categories.

5 Satisfiability in the General Case

Section 3 heavily depends on the fact that all sections are isos, i.e., only isos have a right inverse. However, in the general case we might have conditions of the form $\forall s.\mathcal{A}$ where s has a right inverse r with $s;r = \text{id}$ (note that r need not be unique). This would invalidate our reasoning in the previous sections, since the identity is not necessarily a model of $\forall s.\mathcal{A}$.

► **Example 28.** We work in the category $\mathbf{Graph}_{\text{fin}}$. Consider the following condition $\mathcal{A} = \forall \textcircled{1} \rightarrow \textcircled{1} \textcircled{2} . \exists \textcircled{1} \textcircled{2} \rightarrow \textcircled{1} \rightarrow \textcircled{2} . \text{true}$, defined over a single node $\textcircled{1}$ as root object, which states that the distinguished node has an edge to every other node – including itself, since a non-injective match may merge the two nodes. The first morphism of \mathcal{A} is a section, while the second is injective, but not a section.

The identity on the single node is not a model of \mathcal{A} , but $\textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1}$ is. However, the condition $\mathcal{A} \wedge \forall \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1} . \text{false}$ is unsatisfiable, a fact that would not be detected by Algorithm 22, since neither of the universal quantifiers contains an iso. \lrcorner

Pulling forward isos is still sound even in the general case as the equivalence of Lemma 9 still holds, but it is not sufficient for completeness. Hence we will now adapt the tableau calculus to deal with sections.

► **Lemma 29** (Pulling forward sections). *Let $\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i$ be a universal condition and assume that $f_p, p \in I$, is a section, and r_p is a right inverse of f_p (i.e., $f_p ; r_p = \text{id}$). Furthermore let $\mathcal{A}_{p \downarrow r_p} = \bigvee_{j \in J} \exists h_j . \mathcal{H}_j$ be the result of shifting the p -th child over the right inverse. Then f_p can be pulled forward:*

$$\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \equiv \bigvee_{j \in J} \exists h_j . \left(\mathcal{H}_j \wedge \left(\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \right) \downarrow_{h_j} \right)$$

As for the analogous Lemma 9, pulling forward sections produces an equivalent condition. However, here the child being pulled forward is still included in the children shifted by h_j . Hence the condition will increase in size, unlike for the special case. This is necessary, since f_p might have other inverses which can be used in pulling forward and might lead to new results. This leads to the following adapted rules.

► **Definition 30** (SatCheck rules, general case). *Let \mathcal{A} be an alternating condition. We can construct a tableau for \mathcal{A} by extending it at its leaf nodes as follows:*

$$\bigvee_{i \in I} \exists f_i . \mathcal{A}_i \xrightarrow{f_p} \mathcal{A}_p \quad \left| \quad \begin{array}{l} \text{For every } p \in I: \\ \text{For one } p \in I \text{ such that } f_p \text{ is section, } f_p ; r_p = \text{id}, \\ \text{and } \bigvee_{j \in J} \exists h_j . \mathcal{H}_j = \mathcal{A}_{p \downarrow r_p}: \\ \bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \xrightarrow{\text{id}} \bigvee_{j \in J} \exists h_j . \left(\mathcal{H}_j \wedge \left(\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \right) \downarrow_{h_j} \right) \end{array} \right.$$

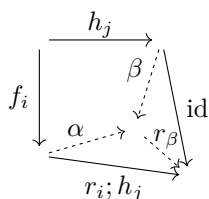
- For existential conditions, for each(!) child condition $\exists f_p . \mathcal{A}_p$, add a new descendant.
- For universal conditions, pick one(!) child condition $\forall f_p . \mathcal{A}_p$ that can be pulled forward in the sense of Lemma 29 and add the result as its (only) descendant.

The rules are similar to those of the specialized case (Definition 10) and as in the special case, we need to define a successor relation on children with sections, that in addition tracks the corresponding right-inverses. The definition of the successor relation is slightly more complex than in the previous case (Definition 11).

► **Definition 31** (Successor relation and tracking right-inverses). We define a relation on pairs of (f_i, r_i) , where f_i is a morphism of a child of a universal condition and r_i is one of its right-inverses. Assume that in the construction of a tableau we have a path

$$\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \xrightarrow{\text{id}} \mathcal{C} \xrightarrow{h_j} \mathcal{H}_j \wedge \left(\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \right) \downarrow_{h_j} = \mathcal{H}_j \wedge \bigwedge_{i \in I} \bigwedge_{(\alpha, \beta) \in \kappa(f_i, h_j)} \forall \beta. (\mathcal{A}_i) \downarrow_{\alpha}$$

where \mathcal{C} is the existential condition given in Lemma 29. Given (f_i, r_i) (where $f_i; r_i = \text{id}$), we can conclude that the outer square below commutes and hence there exists an inner representative square $(\alpha, \beta) \in \kappa(f_i, h_j)$ and r_β such that the diagram below commutes (in particular β is a section and r_β is a retraction). In this situation, we say that (β, r_β) is a (retraction) successor of (f_i, r_i) .



We extend the definition of fairness to cover sections (not just isomorphisms) and also require that all right-inverses of each section are eventually used in a pull-forward step:

► **Definition 32** (Fairness in the general case). A branch of a tableau is fair if for each universal condition \mathcal{A} on the branch, each child condition $\forall f_i. \mathcal{A}_i$ of \mathcal{A} where f_i is a section, and each right-inverse r_i of f_i , there is $n \in \mathbb{N}_0$ such that in the n -th next step, for some indirect successor (f'_i, r'_i) of (f_i, r_i) it holds that f'_i is pulled forward using the right inverse r'_i . (Every universally-quantified section is eventually pulled forward with every inverse.)

For this definition to be effective, we need to require that every section has only finitely many right-inverses (this is true for e.g. **Graph_{fin}**). Given that property, one way to implement a fairness strategy is to use a queue, to which child conditions (and the corresponding right-inverses) are added. This queue has to be arranged in such a way that for each section/retraction pair a successor is processed eventually.

We now show how to adapt the corresponding results of the previous section (Theorems 16–18) and in particular show that infinite and fair branches are always models, from which we can infer soundness and completeness.

► **Theorem 33** (Fair branches are models (general case)). Let \mathcal{A}_0 be an alternating condition. Let a fixed tableau constructed by the rules of Definition 30 be given. Let $\mathcal{A}_0 \xrightarrow{b_1} \mathcal{A}_1 \xrightarrow{b_2} \mathcal{A}_2 \xrightarrow{b_3} \dots$ be a branch of the tableau that is either unextendable and ends with a universal quantification, or is infinite and fair. For such a branch, we define: $P = \{(\bar{b}, \mathcal{A}_i) \mid i \in \mathbb{N}_0, \bar{c} = [b_{i+1}, b_{i+2}, \dots]\} \subseteq \text{Seq} \times \text{Cond}$. Then, $P \subseteq s(u(P))$. (Every open and unextendable or infinite and fair branch in a tableau of Definition 30 corresponds to a model.)

► **Theorem 34** (Soundness and Completeness).

- If all branches in a tableau are closed, then the condition in the root node is unsatisfiable.
- If a condition \mathcal{A} is unsatisfiable, then in every tableau constructed by obeying the fairness constraint all branches are closed.

Proof. Use the proof strategies of Theorems 17 and 18. Tableaux constructed by the rules of Definition 30 have all properties that are required for the proofs (in particular, universal steps lead to an equivalent condition). Use Theorem 33 to obtain models for open branches. ◀

In the general case, although soundness and completeness still hold, we are no longer able to find all finite models as before. This is intuitively due to the fact that a condition might have a finite model, but what we find is a seemingly infinite model that always has the potential to collapse to the finite model.

On the other hand, the weaker requirements on the category imply that more instantiations are possible, such as to $\mathbf{Graph}_{\text{fin}}$ (graphs and arbitrary morphisms). Here, pushouts can be used for representative squares, which allows for a more efficient shift operation that avoids the blowup of the size of the conditions that is associated with jointly epi squares.

6 Conclusion

We introduced a semi-decision procedure for checking satisfiability of nested conditions at the general categorical level. The correctness of this tableau-based procedure has been established using a novel combination of coinductive (up-to) techniques. In the restricted case we also considered the generation of finite models and witnesses for (some) infinite models. Our procedure thereby generalizes prior work [13, 26, 17] on nested graph conditions that are equivalent to first-order logic [6]. As a result, we can also handle cospan categories over adhesive categories (using borrowed context diagrams for representative squares) and other categories, such as Lawvere theories.

There is a notion of Q-trees [7] reminiscent of the nested conditions studied in this paper, but to our knowledge no generic satisfiability procedures have been derived for Q-trees.

We plan to transfer the technique of counterexample-guided abstraction refinement (CEGAR) [9], a program analysis technique based on abstract interpretation and predicate abstraction, to graph transformation and reactive systems. The computation of weakest preconditions and strongest postconditions for nested conditions is fairly straightforward [1] and satisfiability checks give us the necessary machinery to detect and eliminate spurious counterexamples. One still has to work around undecidability issues and understand whether there is a generalization of Craig interpolation, used to simplify conditions.

One further direction for future work is to understand the mechanism for witness generation in more detail. In particular, since it is known that FOL satisfiability for graphs of bounded treewidth is decidable [2], the question arises whether we can find witnesses for all models of bounded treewidth (or a suitable categorical generalization of this notion).

Another direction is to further explore instantiation with a Lawvere theory [14], where arrows are n -tuples of m -ary terms. In this setting representative squares are closely related to unification. The full version [28] contains some initial results.

Finally we plan to complete development of a tool that implements the satisfiability check and explore the potential for optimizations regarding its runtime.

References

- 1 H.J. Sander Bruggink, Raphaël Cauderlier, Mathias Hülsbusch, and Barbara König. Conditional reactive systems. In *Proc. of FSTTCS '11*, volume 13 of *LIPICs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. URL: http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=3325.
- 2 Bruno Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.
- 3 Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.
- 4 Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *MSCS*, 16(6):1133–1163, 2006.

- 5 Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph grammars: An algebraic approach. In *Proc. 14th IEEE Symp. on Switching and Automata Theory*, pages 167–180, 1973.
- 6 Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1996.
- 7 Peter J. Freyd and Andre Scedrov. *Categories, Allegories*. North-Holland, 1990.
- 8 Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.
- 9 Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *Proc. of POPL '04*, pages 232–244. ACM, 2004.
- 10 Mathias Hülsbusch and Barbara König. Deriving bisimulation congruences for conditional reactive systems. In *Proc. of FOSSACS '12*, pages 361–375. Springer, 2012. LNCS/ARCoSS 7213.
- 11 Mathias Hülsbusch, Barbara König, Sebastian Küpper, and Lara Stoltenow. Conditional Bisimilarity for Reactive Systems. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022. doi:10.46298/lmcs-18(1:6)2022.
- 12 Stephen Lack and Paweł Sobociński. Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications*, 39(3), 2005.
- 13 Leen Lambers and Fernando Orejas. Tableau-based reasoning for graph properties. In *Proc. of ICGT '14*, pages 17–32. Springer, 2014. LNCS 8571.
- 14 William Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.
- 15 James J. Leifer. *Operational congruences for reactive systems*. PhD thesis, University of Cambridge Computer Laboratory, September 2001.
- 16 James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, pages 243–258. Springer, 2000. LNCS 1877.
- 17 Marisa Navarro, Fernando Orejas, Elvira Pino, and Leen Lambers. A navigational logic for reasoning about graph properties. *Journal of Logical and Algebraic Methods in Programming*, 118:100616, 2021.
- 18 Karl-Heinz Pennemann. An algorithm for approximating the satisfiability problem of high-level conditions. In *GT-VC@CONCUR*, volume 213.1 of *Electronic Notes in Theoretical Computer Science*, pages 75–94. Elsevier, 2007.
- 19 Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Universität Oldenburg, May 2009.
- 20 Damien Pous. Complete lattices and up-to techniques. In *Proc. of APLAS '07*, pages 351–366. Springer, 2007. LNCS 4807.
- 21 Damien Pous. *Techniques modulo pour les bisimulations*. PhD thesis, ENS Lyon, February 2008. URL: <https://hal.archives-ouvertes.fr/tel-01441480>.
- 22 Damien Pous and Davide Sangiorgi. Enhancements of the coinductive proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2011.
- 23 Arend Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.
- 24 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.
- 25 Vladimiro Sassone and Paweł Sobociński. Reactive systems over cospans. In *Proc. of LICS '05*, pages 311–320. IEEE, 2005.
- 26 Sven Schneider, Leen Lambers, and Fernando Orejas. Symbolic model generation for graph properties. In *Proc. of FASE '17*, pages 226–243. Springer, 2017. LNCS 10202.
- 27 Paweł Sobociński. *Deriving process congruences from reaction rules*. PhD thesis, Department of Computer Science, University of Aarhus, 2004.

- 28 Lara Stoltenow, Barbara König, Sven Schneider, Andrea Corradini, Leen Lambers, and Fernando Orejas. Coinductive techniques for checking satisfiability of generalized nested conditions, 2024. arXiv:2407.06864. URL: <https://arxiv.org/abs/2407.06864>.
- 29 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

A Additional Material for §2 (Preliminaries)

Graphs and graph morphisms

We will define in more detail which graphs and graph morphisms we are using: in particular, a graph is a tuple $G = (V, E, s, t, \ell)$, where V, E are sets of nodes respectively edges, $s, t: E \rightarrow V$ are the source and target functions and $\ell: V \rightarrow \Lambda$ (where Λ is a set of labels) is the node labelling function. In the examples we will always omit node labels by assuming that there is only a single label.

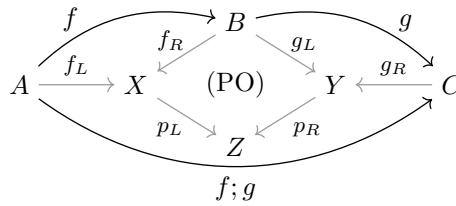
A graph G is finite if both V and E are finite.

Furthermore, given two graphs $G_i = (V_i, E_i, s_i, t_i, \ell_i)$, $i \in \{1, 2\}$, a graph morphism $\varphi: G_1 \rightarrow G_2$ consists of two maps $\varphi_V: V_1 \rightarrow V_2$, $\varphi_E: E_1 \rightarrow E_2$ such that $\varphi_V \circ s_1 = s_2 \circ \varphi_E$, $\varphi_V \circ t_1 = t_2 \circ \varphi_E$ and $\ell_1 = \ell_2 \circ \varphi_V$.

In the examples, the mapping of a morphism is given implicitly by the node identifiers: for instance, $\textcircled{1} \textcircled{2} \rightarrow \textcircled{1} \textcircled{3} \textcircled{2}$ adds the node identified by 3 and adds two edges from the existing nodes identified by 1 and 2.

Cospans and cospan composition

Two cospans $f: A \xrightarrow{f_L} X \xleftarrow{f_R} B$, $g: B \xrightarrow{g_L} Y \xleftarrow{g_R} C$ are composed by taking the pushout (p_L, p_R) of (f_R, g_L) as shown in Figure 1. The result is the cospan $f; g: A \xrightarrow{f_L; p_L} Z \xleftarrow{g_R; p_R} C$, where Z is the pushout object of f_R, g_L . We see an arrow $f: A \rightarrow C$ of **Cospan(D)** as an object B of **D** equipped with two interfaces A, C and corresponding arrows f_L, f_R to relate the interfaces to B , and composition glues the inner objects of two cospans via their common interface.



■ **Figure 1** Composition of cospans f and g is done via pushouts.

In order to make sure that arrow composition in **Cospan(D)** is associative on the nose, we quotient cospans up to isomorphism. In more detail: two cospans $f: A \xrightarrow{f_L} X \xleftarrow{f_R} B$, $g: A \xrightarrow{g_L} Y \xleftarrow{g_R} B$ are equivalent whenever there exists an iso $\iota: X \rightarrow Y$ such that $f_L; \iota = g_L$, $f_R; \iota = g_R$. Then, arrows are equivalence classes of cospans.

Equivalence laws for conditions

We rely on the results given in the following two propositions that were shown in [1]. They were originally stated for satisfaction with single arrows, but it is easy to see they are valid for possibly infinite sequences as well: conditions have finite depth and satisfaction only refers to finite prefixes of the sequence.

► **Proposition 35** (Adjunction). *Let \mathcal{A}, \mathcal{B} be two conditions with root object A , let \mathcal{C}, \mathcal{D} be two conditions with root object B and let $\varphi: A \rightarrow B$. Then it holds that:*

1. $\mathcal{A} \models \mathcal{B}$ implies $\mathcal{A}_{\downarrow\varphi} \models \mathcal{B}_{\downarrow\varphi}$.
2. $\mathcal{C} \models \mathcal{D}$ implies $\mathcal{Q}\varphi.\mathcal{C} \models \mathcal{Q}\varphi.\mathcal{D}$ for $\mathcal{Q} \in \{\exists, \forall\}$.
3. $\exists\varphi.(\mathcal{A}_{\downarrow\varphi}) \models \mathcal{A}$ and for every \mathcal{C} with $\exists\varphi.\mathcal{C} \models \mathcal{A}$ we have that $\mathcal{C} \models \mathcal{A}_{\downarrow\varphi}$.
4. $\mathcal{A} \models \forall\varphi.(\mathcal{A}_{\downarrow\varphi})$ and for every \mathcal{C} with $\mathcal{A} \models \forall\varphi.\mathcal{C}$ we have that $\mathcal{A}_{\downarrow\varphi} \models \mathcal{C}$.

► **Proposition 36** (Laws for conditions). *One easily obtains the following laws for shift and quantification, conjunction and disjunction:*

$$\begin{array}{ll}
 \mathcal{A}_{\downarrow\text{id}} \equiv \mathcal{A} & \mathcal{A}_{\downarrow\varphi; \psi} \equiv (\mathcal{A}_{\downarrow\varphi})_{\downarrow\psi} \\
 \forall\text{id}.\mathcal{A} \equiv \mathcal{A} & \forall(\varphi; \psi).\mathcal{A} \equiv \forall\varphi.\forall\psi.\mathcal{A} \\
 \exists\text{id}.\mathcal{A} \equiv \mathcal{A} & \exists(\varphi; \psi).\mathcal{A} \equiv \exists\varphi.\exists\psi.\mathcal{A} \\
 (\mathcal{A} \wedge \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \wedge \mathcal{B}_{\downarrow\varphi} & (\mathcal{A} \vee \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \vee \mathcal{B}_{\downarrow\varphi} \\
 \forall\varphi.(\mathcal{A} \wedge \mathcal{B}) \equiv \forall\varphi.\mathcal{A} \wedge \forall\varphi.\mathcal{B} & \exists\varphi.(\mathcal{A} \vee \mathcal{B}) \equiv \exists\varphi.\mathcal{A} \vee \exists\varphi.\mathcal{B}
 \end{array}$$

Borrowed context diagrams

For cospan categories over adhesive categories (such as $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}})$), borrowed context diagrams – initially introduced as an extension of DPO rewriting [4] – can be used as representative squares. Before we can introduce such diagrams, we first need the notion of jointly epi.

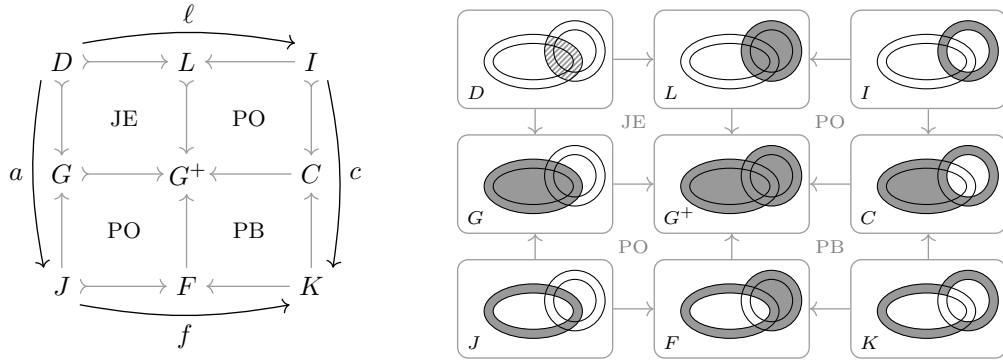
► **Definition 37** (Jointly epi). *A pair of arrows $f: B \rightarrow D$, $g: C \rightarrow D$ is jointly epi (JE) if for each pair of arrows $d_1, d_2: D \rightarrow E$ the following holds: if $f; d_1 = f; d_2$ and $g; d_1 = g; d_2$, then $d_1 = d_2$.*

In $\mathbf{Graph}_{\text{fin}}$ jointly epi equals jointly surjective, meaning that each node or edge of D is required to have a preimage under f or g or both (it contains only images of B or C).

This criterion is similar to, but weaker than a pushout: For jointly epi graph morphisms $d_1: B \rightarrow D$, $d_2: C \rightarrow D$, there are no restrictions on which elements of B, C can be merged in D . However, in a pushout constructed from morphisms $a_1: A \rightarrow B$, $a_2: A \rightarrow C$, elements in D can (and must) only be merged if they have a common preimage in A . (Hence every pushout generates a pair of jointly epi arrows, but not vice versa.)

► **Definition 38** (Borrowed context diagram [10]). *A commuting diagram in the category $\mathbf{ILC}(\mathbf{C})$, where \mathbf{C} is adhesive, is a borrowed context diagram whenever it has the form of the diagram shown in Figure 2a, and the four squares in the base category \mathbf{C} are pushout (PO), pullback (PB) or jointly epi (JE) as indicated. Arrows depicted as \rightsquigarrow are mono. In particular $L \rightsquigarrow G^+$, $G \rightsquigarrow G^+$ must be jointly epi.*

Figure 2b shows a more concrete version of Figure 2a, where graphs and their overlaps are depicted by Venn diagrams (assuming that all morphisms are injective). Because of the two pushout squares, this diagram can be interpreted as composition of cospans $a; f = \ell; c = D \rightarrow G^+ \leftarrow K$ with extra conditions on the top left and the bottom right square. The top left square fixes an overlap G^+ of L and G , while D is contained in the intersection of L and G (shown as a hatched area). Being jointly epi ensures that it really is an overlap and does not contain unrelated elements. The top right pushout corresponds to the left pushout of a DPO rewriting diagram. It contains a total match of L in G^+ . Then, the bottom left pushout gives us the minimal borrowed context F such that applying the rule becomes possible. The top left and the bottom left squares together ensure that the contexts to be considered are not larger than necessary. The bottom right pullback ensures that the interface K is as large as possible.



(a) Structure of a borrowed context diagram. The inner, lighter arrows are morphisms of the base category \mathbf{C} , while the outer arrows are morphisms of $\mathbf{ILC}(\mathbf{C})$.

(b) Borrowed context diagrams represented as Venn diagrams. The outer circles represent graphs L, G , and the area between the inner and outer circles represents their interfaces I, J .

■ **Figure 2** Borrowed context diagrams.

For more concrete examples of borrowed context diagrams, we refer to [4, 11].

For cospan categories over adhesive categories, borrowed context diagrams form a representative class of squares [1]. Furthermore, for some categories (such as $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$), there are – up to isomorphism – only finitely many jointly epi squares for a given span of monos and hence only finitely many borrowed context diagrams given a, ℓ (since pushout complements along monos in adhesive categories are unique up to isomorphism).

Whenever the two cospans ℓ, a are in $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, it is easy to see that f, c are in $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, i.e., they consist only of monos, i.e., injective morphisms.

Note also that representative squares in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ are simply jointly epi squares and they can be straightforwardly extended to squares of $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$.

Visualization of shifts

Given a condition \mathcal{A} and an arrow $c: A = \text{RO}(\mathcal{A}) \rightarrow B$, we will visualize shifts in diagrams as follows:

$$\begin{array}{c} \mathcal{A} \qquad \mathcal{A}_{\downarrow c} \\ \begin{array}{ccc} \swarrow & & \swarrow \\ A & \xrightarrow{c} & B \xrightarrow{d} X \\ \searrow & & \searrow \end{array} \end{array}$$

Remember that for an arrow $d: B \rightarrow X$ it holds that $d \models \mathcal{A}_{\downarrow c} \iff c; d \models \mathcal{A}$.

B Additional Material for §4 (Witnesses for infinite models)

■ **Table 1** Steps for the condition of Example 27, showing that a repeating infinite model exists.

\mathcal{C}_0	\mathcal{C}_1	$(\mathcal{C}_1)_{\downarrow m}$
$\forall [\textcircled{C} \rightarrow \textcircled{D}]. \text{false}$	$\forall [\textcircled{C} \rightarrow \textcircled{1} \rightarrow \textcircled{2}]. \text{false}$	
$\wedge \forall [\textcircled{1} \oplus]. (\exists [\textcircled{1} \oplus \rightarrow \textcircled{O}]. \text{true} \vee \exists [\textcircled{1} \leftarrow \oplus]. \text{true})$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2}]. \text{false}$ $\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \oplus]. (\exists [\textcircled{1} \rightarrow \textcircled{2} \oplus \rightarrow \textcircled{O}]. \text{true} \vee \exists [\textcircled{1} \rightarrow \textcircled{2} \leftarrow \oplus]. \text{true} \vee \exists [\oplus \rightarrow \textcircled{1} \rightarrow \textcircled{2}]. \text{true})$	$\forall [\textcircled{2} \oplus]. (\exists [\textcircled{2} \oplus \rightarrow \textcircled{O}]. \text{true} \vee \exists [\textcircled{2} \leftarrow \oplus]. \text{true})$
$\wedge \forall [\textcircled{1}]. \exists [\textcircled{1} \rightarrow \textcircled{2}]. \text{true}$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2}]. (\exists [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3}]. \text{true} \vee \exists [\textcircled{1} \leftarrow \rightarrow \textcircled{2}]. \text{true})$	$\wedge \forall [\textcircled{2}]. \exists [\textcircled{2} \rightarrow \textcircled{3}]. \text{true}$
$\wedge \forall [\textcircled{1} \textcircled{A} \rightarrow \textcircled{X} \leftarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \textcircled{A} \rightarrow \textcircled{X} \leftarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{2} \textcircled{A} \rightarrow \textcircled{X} \leftarrow \textcircled{B}]. \text{false}$
$\wedge \forall [\textcircled{1} \rightarrow \textcircled{X} \leftarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{X} \leftarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{2} \rightarrow \textcircled{X} \leftarrow \textcircled{B}]. \text{false}$
$\wedge \dots$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \leftarrow \textcircled{B}]. \text{false}$ $\wedge \dots$	$\wedge \forall [\textcircled{2} \leftarrow \textcircled{B}]. \text{false}$ $\wedge \dots$