Finding models through graph saturation

Sebastiaan J. C. Joosten

Formal Methods and Tools group, University of Twente, the Netherlands
Email: `Sebastiaan.Joosten@utwente.nl`

Abstract. We give a procedure that can be used to automatically satisfy invariants of a certain shape. These invariants may be written with the operations intersection, composition and converse over binary relations, and equality over these operations. We call these invariants sentences that we interpret over graphs. For questions stated through sets of these sentences, this paper gives a semi-decision procedure we call graph saturation. It decides entailment over these sentences, inspired on graph rewriting. We prove correctness of the procedure. Moreover, we show the corresponding decision problem to be undecidable. This confirms a conjecture previously stated by the author [7].

This is the preprint of `https://doi.org/10.1016/j.jlamp.2018.06.005`.

1 Introduction

The question 'what models does a set of formulas $\mathcal{T}$ have' has practical relevance, as it is an abstraction of an information system: We interpret the data set stored in an information system at a certain point in time as a model, and each invariant of the system corresponds to a formula in $\mathcal{T}$. This correspondence is the core idea behind languages such as Ampersand [8], that define an information system this way. Users of an information system try to change the data set continually. These changes might violate the constraints. While Ampersand responds to such violations by rejecting the change, it would be convenient to automatically add data items such that all constraints are satisfied. The question then becomes: what data items should be added? We solve this question partially by means of a graph saturation procedure.

The question 'does a set of formulas $\mathcal{T}$ have a model satisfying all formulas' essentially asks whether $\mathcal{T}$ is free of contradictions. So far, we did not discuss the language in which we can write the formulas in $\mathcal{T}$. Several interesting problems arise when restricting the language in which we can write formulas: the satisfiability problem is obtained by restricting to disjunctions of positive and negative literals. Restricting to linear integer equalities, we obtain the linear programming problem. In this paper, we restrict those formulas to equalities over terms, in which terms are expressions of relations combined through the allegorical operations[1]. We define *sentence* to be a formula over the restricted language considered in this paper (Definition 3).

Our interest in this language stems from experience in describing systems in Ampersand. All operations from relation-algebra are part of the Ampersand language. The

[1] These are $;$, $\sqcap$, $\breve{}$ and $\mathbb{1}$. See the book by Freyd and Scedrov for details on allegories [4].

operations considered here include only the most frequently used subset of those operations. Therefore, many of the formulas used in Ampersand will be sentences as considered in this work. We therefore consider this work a step towards an Ampersand system that helps the user find models.

1.1 Approach

We give a short summary of the basic algorithm presented here, so we can better relate our approach to other literature, describe our contributions, and give the outline of this paper. Italicised words in the next paragraph are defined later.

The algorithm aims to determine whether there is a particular *model* for a set of *sentences*, say $\mathcal{T}$, and is guaranteed to terminate if no such *model* exists. It proceeds to construct a (possibly infinite) *model* otherwise. The procedure has two phases: first, we translate the *sentences* in $\mathcal{T}$ into a set of *graph rules*. We then apply a saturation procedure on the *graph rules*. This procedure creates a *chain* of *graphs*, whose limit is a *least consequence graph*. A *graph* contains a *conflict* if it has an edge with the label $\bot$. If a *least consequence graph* contains a *conflict*, then there is no *model* for $\mathcal{T}$. Otherwise, the *least consequence graph* corresponds to a *model* of $\mathcal{T}$, if the *graph rules* correspond to $\mathcal{T}$ according to a straightforward *translation*. We abort the procedure as soon as a *conflict* arises, because we can be sure that no *models* for $\mathcal{T}$ exist in this case. A second question we can answer through the same algorithm is that of *entailment*: *entailment* is the question whether a *sentence* ϕ follows from a set of *sentences* $\mathcal{T}$.

In an information system, a least consequence graph is a well suited to determine which data items to add: If conflict free, it corresponds to a graph that maintains the invariants. At the same, it only contains necessary consequences: it will not cause data items to be added that have nothing to do with the change the user made.

1.2 Related Work

We compare the work in this paper to existing work in two ways: work it is similar to in motivation, and work it is similar to in implementation from an abstract perspective. In motivation, our research is closely related to the Alcoa tool, which we'll discuss first. In approach, our methods are related to description logics and to graph rewriting, which we'll discuss second.

The Alcoa Tool. Our search for a reasoner for Ampersand is related to Alcoa [6], which is the analyzer for Alloy [5], a language based on Z [13]. Like Ampersand, the languages Z and Alloy are based on relations. Alloy is a simplification of Z: it reduces the supported operations to a set that is small yet powerful. This paper differs from Alloy in the expressivity of its operations, however: Alloy allows writing full first order formula's plus the Kleene-star, making it a language that is even more expressive than Ampersand. We compare to Alcoa because this work is similar in purpose.

In Alloy, a user may write assertions, which are formulas that the user believes follow from the specification. Alcoa tries to find counterexamples to those assertions, as well as a finite model for the entire specification. Unfortunately, several properties of the Alcoa tool hinder our purposes in Ampersand: Alcoa requires an upper bound on the size of (or number of elements in) the model. It does not perform well if this bound is too large. In a typical information system, the amount of data is well above what can be considered 'too large'. As an additional complication, we cannot adequately predict the size of the model we might require. This is why we look at other methods for achieving similar goals.

Description Logics. We can regard our procedure as a way to derive facts from previously stated facts: this is what happens in terms of sentences between subsequent graphs in the chain we create. So called description logics are languages used in conjunction with an engine, that gives a procedure to learn new facts from previously learned facts, using declarative statements (or rules) in the corresponding description logic. For a good overview of description logics, see the book on that topic by Baader [1].

A set of derivation rules is consistent if it has a model. For a highly expressive description logic such as OWL DL, determining consistency is undecidable. Still, a rule engine for OWL DL will happily try to learn new facts until a model is found. Users of OWL DL typically need to ensure that the stated derivation rules together with the rule engine give a terminating procedure. For many description logics, termination of its rule engine is syntactically guaranteed, and these logics are consequentially decidable.

The description logic for which the language and implementation is closest to our language is the logic $\mathcal{EL}$ and its extensions proposed by Baader et al [2,3]. Instead of using tableau-based procedures, as most description logics, it uses a saturation-based reasoner. Syntax of the derivation rules is limited to ensure termination of any saturation procedure: $\mathcal{EL}$ allows statements about unary relations using top, bottom, individual elements called 'nominal', and conjunction. Statements about binary relations use a different syntax, that can be translated into sentences using composition, converse and the identity relation (but not necessarily vice-versa). By modeling $\mathcal{EL}$'s unary relations as binary relations that are a subset of the identity relation, all of $\mathcal{EL}$ and its extensions can be expressed through the sentences described in this paper. In particular, the syntax of $\mathcal{EL}$ does not have disjunctions, thus eliminating the need for backtracking. In fact, $\mathcal{EL}$ is designed such that its consistency can be decided deterministically in polynomial time. Its extensions have different complexity bounds, but preserve polynomial runtime for the fragment that falls within $\mathcal{EL}$.

In our work, we do not work under the assumption of termination: neither the user or the syntax guarantees it. This allows us to use a richer language than one that is syntactically guaranteed to terminate. Despite this lack of termination, we do ensure termination in case of conflicts: a conflict will be found if our sentences imply it. This allows the user to approach certain problems through any set of rules within the grammar, rather than just those sets for which the implementation is guaranteed to terminate. The implementation presented in our work applies graph rules 'fairly' to ensure this. Fair application of rules is typically not required in the implementation of description logic engines.

Graph Rewriting. A central concept in graph rewriting is that a pushout can be used to apply a *graph rule* on a graph, as described by Wolfram Kahl [9]. The usual idea of such a pushout is that it models execution by removing a portion of the graph, and replacing it with the result of the execution step. Graph rewriting might then terminate when no rules can be applied anymore. Our approach diverges on this point: rather than execution, a step models learning a deducible conclusion. Rather than terminating when no step is possible, we are interested in the limit of the sequence of graphs. For this reason, the notions of weak pushout step and weak pushout don't coincide exactly: we ensure that the sequence of graphs form a chain in order for the limit to exist.

The term saturation is borrowed from the saturation procedure in resolution procedures, introduced by Robinson in 1965 [11]. His procedure solves an entailment problem over a certain language. As in his procedure, our procedure adds derivable facts iteratively.

1.3 Contributions and Paper Outline

We mentioned how this paper contributes by comparing it to related work: Compared to the work on $\mathcal{EL}$, our approach allows sentences in a richer language, and we present a translation to graph rules to separate the semantics from the core of the implementation. Compared to the work on graph-rewriting, we present a new graph-based manipulation algorithm, and give an interpretation of those graphs as models for sets of sentences.

We also relate the contribution of this paper to a paper presenting Amperspiegel [7]. This earlier paper by the author conjectured that the problem whether no least consequence graph exists is undecidable. It also contains a procedure for finding such graphs, which it conjectures to be correct. We will show that the procedure in the paper is an instance of the variations of the procedure described here. To simplify the presentation of our results in this paper, the definition of a least consequence graph is slightly different here: A least consequence graphs always exist according to the definitions used in this paper. In the terminology of this paper, the conjecture just mentioned would be: the problem whether a least consequence graph contains a conflict is undecidable. This paper proves the stated result.

The procedure presented in this work is simpler than the one presented earlier. However, the latter can be obtained by applying optimizations to the former. We show correctness of the procedure, and show that the existence of a conflict free least consequence graph implies the existence of models for a set of sentences. Semi-decidability of consistency is not surprising in this setting: the logic we consider is less expressive than several logics for which semi-decidability is established. Our contribution lies in presenting an intuitive, flexible, graph-based algorithm that does not use backtracking.

The outline of this paper is as follows: we define the syntax and semantics of sentences in Section 2, and define the problems our procedure aims to solve: deciding consistency and entailment. Section 3 then introduces the heart of the procedure by defining least consequence graphs and indicating how to obtain them through graph rules. Section 4 connects these two, by giving a translation of sentences to graph rules. The procedure is given as an algorithm in Section 5, and we indicate how to use the procedure to decide consistency and entailment. Before going to the conclusion, we indicate why we cannot hope to do better than giving a possibly non-terminating procedure, by proving undecidability in Section 6. Conclusion and acknowledgements are in Section 7.

2 Background and Problem Statement

As this paper primarily deals with directed labeled graphs, we choose to use these graphs for the semantics of sentences as well. There is no fundamental difference between this presentation and the usual binary relation based semantics usually presented as the canonical allegory (or as the canonical model for relation algebra). However, using graphs now simplifies our proofs later on, and makes it that we do not have to define them later. Graphs are defined as follows:

Definition 1 (Graph, Empty, Finite). *A directed labeled graph $G = (\mathcal{L}, V, E)$ is given by a set of labels $\mathcal{L}$, a set of vertices V, and a set of edges $E \subseteq \mathcal{L} \times V \times V$. The set of all graphs with labels $\mathcal{L}$ is written as $\mathbb{G}_{\mathcal{L}}$. We write* graph *when we mean a directed labeled graph. We say that a graph is* finite *if both its set of vertices V and its set of edges E are finite. The cardinality of V is written $|G|$. A graph with no vertices (and therefore no edges) is called* empty*, written $\mathbb{0}_{\mathcal{L}}$.*

Terms are built inductively from relation symbols $\mathcal{L}$, combined with the operations $\_ \sqcap \_$, $\_ \mathbin{;} \_$, and $\_^{\smile}$. The operations stand for intersection, relational composition, and relational converse, respectively. The set of all terms over $\mathcal{L}$ is denoted as $\mathbb{E}_{\mathcal{L}}$. We use the same letter $\mathcal{L}$ to indicate labels in graphs, as well as relation symbols in terms. This notation is deliberately chosen because of the semantics given in Definition 2 below.

Definition 2 (Semantics). *For a graph $G = (\mathcal{L}, V, E)$, the* semantics *of a term $\mathbb{e} \in \mathbb{E}_{\mathcal{L}}$, written as $[\![\mathbb{e}]\!]_G \subseteq V \times V$, is as in representable relation algebra:*

$$\begin{aligned}
[\![l]\!]_G &= \{(x, y) \mid (l, x, y) \in E\} \\
[\![\mathbb{e}_1 \sqcap \mathbb{e}_2]\!]_G &= [\![\mathbb{e}_1]\!]_G \cap [\![\mathbb{e}_2]\!]_G \\
[\![\mathbb{e}^{\smile}]\!]_G &= \left\{(y, x) \mid (x, y) \in [\![\mathbb{e}]\!]_G\right\} \\
[\![\mathbb{e}_1 \mathbin{;} \mathbb{e}_2]\!]_G &= \left\{(x, y) \mid \exists z.\ (x, z) \in [\![\mathbb{e}_1]\!]_G \ \wedge\ (z, y) \in [\![\mathbb{e}_2]\!]_G\right\}
\end{aligned}$$

A sentence is the proposition stating that two terms are equal:

Definition 3 (Sentence, Holds). *Given the terms $\mathbb{e}_1, \mathbb{e}_2 \in \mathbb{E}_{\mathcal{L}}$, the pair $(\mathbb{e}_1, \mathbb{e}_2)$ is a* sentence*, written $\mathbb{e}_1 = \mathbb{e}_2$. We write $\mathbb{e}_L \sqsubseteq \mathbb{e}_R$ for a sentence of the shape $\mathbb{e}_L = \mathbb{e}_L \sqcap \mathbb{e}_R$. We say that a sentence* holds *in graph G if $[\![\mathbb{e}_1]\!]_G = [\![\mathbb{e}_2]\!]_G$, in which case we write: $G \vDash \mathbb{e}_1 = \mathbb{e}_2$. If $\mathcal{T}$ is a set of sentences, we say that it holds in G if each of the sentences holds in G, written $G \vDash \mathcal{T}$.*

Lemma 1. *Let $\mathbb{e}_1, \mathbb{e}_2 \in \mathbb{E}_{\mathcal{L}}$, and $G \in \mathbb{G}_{\mathcal{L}}$.*

$$\begin{aligned}
G \vDash \mathbb{e}_1 \sqsubseteq \mathbb{e}_2 \quad &\Leftrightarrow \quad [\![\mathbb{e}_1]\!]_G \subseteq [\![\mathbb{e}_2]\!]_G \\
G \vDash \mathbb{e}_1 = \mathbb{e}_2 \quad &\Leftrightarrow \quad G \vDash \mathbb{e}_1 \sqsubseteq \mathbb{e}_2 \ \wedge\ G \vDash \mathbb{e}_2 \sqsubseteq \mathbb{e}_1
\end{aligned}$$

We deviate slightly from allegories: First, we are working in an untyped setting, or put differently: in an allegory with only a single object. In ‘typed allegories’, allegories with more than one object, relational composition is a partial operation. This deviation is not fundamental: we are simply adding more terms to our language than would be there in the typed setting. A second deviation is that we have not introduced identity morphisms. We introduce the identity symbol $\mathbb{1}$ by treating it as a symbol in $\mathcal{L}$. Our approach generalizes to multiple identity symbols, as one would expect in allegories with multiple objects, but this is out of scope in favor of a simplified presentation.

Apart from the identity symbol $\mathbb{1}$, we also introduce bottom and top ($\bot$ and $\top$) as symbols in $\mathcal{L}$. In Definition 4 we give the interpretation of these designated relation symbols, defining a graph as standard if it adheres to this interpretation.

Definition 4 (Standard). *We say that a set of labels $\mathcal{L}$ is* standard *with the (possibly empty) set of constant elements C if $\bot, \top, \mathbb{1} \in \mathcal{L}$ and $C \subseteq \mathcal{L}$. We refer to elements in C simply as constants. Let $\mathcal{L}$ be a standard set of labels with the constants C. A graph $G = (\mathcal{L}, V, E)$ is called* standard *if $V \neq \{\}$, and:*

$$\begin{aligned} [\![\mathbb{1}]\!]_G &= \{(x, x) \mid x \in V\} \\ [\![\top]\!]_G &= \{(x, y) \mid x, y \in V\} \\ [\![\bot]\!]_G &= \{\} \\ \forall c \in C.\ [\![c]\!]_G &= \{(c, c)\} \end{aligned}$$

This work looks at models for $\mathcal{T}$, and investigates whether $\mathcal{T}$ entails ϕ. We can now give the definitions that necessary to make this precise.

Definition 5 (Model, Consistent). *Let $\mathcal{T}$ be a set of sentences over a standard set of labels $\mathcal{L}$ (with constants C). We say that the graph $G \in \mathbb{G}_{\mathcal{L}}$ is a* model *for $\mathcal{T}$ if every sentence in $\mathcal{T}$ holds in G and G is standard. We say that $\mathcal{T}$ is* consistent *if such a graph exists. We may refer to any set of sentences $\mathcal{T}$ as an instance of the consistency problem.*

Definition 6 (Entails). *Let $\mathcal{T}$ be a set of sentences over a standard set of labels $\mathcal{L}$, and let ϕ be a sentence over $\mathcal{L}$. We say that $(\mathcal{T}, \phi)$ is an instance of the* entailment problem*. We say that $\mathcal{T}$* entails *ϕ if for all standard graphs G, $G \vDash \mathcal{T}$ implies $G \vDash \phi$.*

Our use of 'standard' in these definitions is not a restriction: given a graph G over a language $\mathcal{L}$ with $\bot, \top, \mathbb{1} \notin \mathcal{L}$, we can make it into a standard graph G' over $\mathcal{L} \cup \{\bot, \top, \mathbb{1}\}$, choosing the constants $C = \{\}$, and adding the edges according to Definition 4. Then $G \vDash \phi$ if and only if $G' \vDash \phi$ for $\phi \in \mathbb{E}_{\mathcal{L}}$, as ϕ cannot talk about $\bot, \top$ or $\mathbb{1}$.

We prove a straightforward correspondence between the consistency problem and the entailment problem:

Lemma 2. *There is a standard graph G such that $G \vDash \mathcal{T}$ if and only if $\mathcal{T}$ does not entail $\bot = \top$.*

Proof. We first prove that if $\mathcal{T}$ entails $\bot = \top$, then there is no standard graph with $G \vDash \mathcal{T}$: A standard graph must have at least one vertex, say v. Then $(v, v) \in [\![\top]\!]_G$, and $(v, v) \notin [\![\bot]\!]_G$, so $[\![\bot]\!]_G \neq [\![\top]\!]_G$. For the other direction: Suppose there is no standard graph with $G \vDash \mathcal{T}$, then entailment of any formula follows by definition. $\square$

We proceed with a small example of sentences, an entailment and a consistency problem. As an example, we make an administration of people and rooms. We use the label $\mathtt{i}$ to denote which room a person Inhabits, and $\mathtt{r}$ to denote which people are Roommates. We think of the labels in terms of their semantics: as binary relations. We show how these relations are connected by the sentence expressing: Two people are roommates if and only if they share a room: $\mathtt{r} = \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}$. This gives a one-sentence theory $\mathcal{T} = \{\mathtt{r} = \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}\}$ on a standard set of labels that contains $\mathtt{i}$ and $\mathtt{r}$.

We ask ourselves if being a roommate is a transitive relation. That is, does $\mathcal{T}$ entail $\mathtt{r} \mathbin{;} \mathtt{r} \sqsubseteq \mathtt{r}$ or not? The answer is negative. A possible counter-example our procedure may produce is a graph G with:

$$\begin{aligned}\llbracket \mathtt{i} \rrbracket_G &= \{(0,3),(1,4),(2,3),(2,4)\}\\ \llbracket \mathtt{r} \rrbracket_G &= \{(0,0),(0,2),(1,1),(1,2),(2,0),(2,1),(2,2)\}\end{aligned}$$

In this example, $0, 1, 2$ are people, and $3, 4$ are their rooms. While 0 and 1 are roommates of 2, 0 is not a roommate of 1. Note that person 2 has two rooms in this example. We may wish to forbid this: the sentence $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$ expresses that $\mathtt{i}$ is univalent (if two rooms are inhabited by the same person, those two must be the same room). Now $\mathcal{T} = \{\mathtt{r} = \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}, \mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}\}$ entails $\mathtt{r} \mathbin{;} \mathtt{r} \sqsubseteq \mathtt{r}$, and our procedure shows this, as we will demonstrate in Section 5.

We elaborate on the same example for checking consistency, and add some constants to $\mathcal{L}$. Let $C = \{\text{‘}\mathtt{Liz}\text{’}, \text{‘}\mathtt{Jon}\text{’}, \text{‘}\mathtt{Batcave}\text{’}, \text{‘}\mathtt{Room\ 11}\text{’}\}$. Let's say we want Liz and Jon to be roommates, and ask ourselves if that's possible. That is, we wish to solve the consistency problem for:

$$\begin{aligned}\mathcal{T} = \{\ & \mathtt{r} = \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}\\ ,\ & \mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}\\ ,\ & \text{‘}\mathtt{Liz}\text{’} \mathbin{;} \top \mathbin{;} \text{‘}\mathtt{Jon}\text{’} \sqsubseteq \mathtt{r}\}\end{aligned}$$

Our procedure then produces a graph like G with:

$$\begin{aligned}\llbracket \mathtt{i} \rrbracket_G &= \{(\text{‘}\mathtt{Liz}\text{’}, 0), (\text{‘}\mathtt{Jon}\text{’}, 0)\}\\ \llbracket \mathtt{r} \rrbracket_G &= \{(\text{‘}\mathtt{Liz}\text{’}, \text{‘}\mathtt{Jon}\text{’})\}\end{aligned}$$

Without going into details on why, we remark that our procedure comes up with a new room, here called 0, even with the Batcave and Room 11 available. Finally, if we require Liz and Jon to be in their rooms of their choice, the Batcave and Room 11 respectively, our procedure detects that the requirements are no longer consistent. That is, the following theory is not consistent:

$$\begin{aligned}\mathcal{T} = \{\ & \mathtt{r} = \mathtt{i} \mathbin{;} \mathtt{i}^{\smile},\ \mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}\\ ,\ & \text{‘}\mathtt{Liz}\text{’} \mathbin{;} \top \mathbin{;} \text{‘}\mathtt{Jon}\text{’} \sqsubseteq \mathtt{r}\\ ,\ & \text{‘}\mathtt{Liz}\text{’} \mathbin{;} \top \mathbin{;} \text{‘}\mathtt{Batcave}\text{’} \sqsubseteq \mathtt{i}\\ ,\ & \text{‘}\mathtt{Jon}\text{’} \mathbin{;} \top \mathbin{;} \text{‘}\mathtt{Room\ 11}\text{’} \sqsubseteq \mathtt{i}\}\end{aligned}$$

3 Graph Rules and Consequence Graphs

This section defines a least consequence graph, and gives conditions on a chain of graphs that ensure that its limit is a least consequence graph. When a graph is a least consequence graph, we can use it to answer both the entailment problem and the consistency problem. The conditions on a chain of graphs tell us how graph rules should be applied by possible implementations. Basically 'least consequence graph' characterises that all graph rules are applied correctly and sufficiently. We define graph rules in Definition 9, least consequence graphs in Definition 11, and conclude the section with the conditions that give us a least consequence graph, proven in Lemma 3 and 4.

We introduce special notation for two basic operations on graphs: relabeling of vertices, and taking the union of two graphs. Suppose we have a function $f : V_1 \to V_2$, where V_1 is the set of vertices of some graph. We can apply the function on the corresponding graph, written $\hat{f}$:

$$\hat{f}((\mathcal{L}, V_1, E)) = \left(\mathcal{L}, \{f(v) \mid v \in V_1\}, \{(l, f(x), f(y)) \mid (l, x, y) \in E\}\right)$$

For taking the union of two graphs, we simply write $\cup$, defined as follows:

$$(\mathcal{L}_1, V_1, E_1) \cup (\mathcal{L}_2, V_2, E_2) = (\mathcal{L}_1 \cup \mathcal{L}_2, V_1 \cup V_2, E_1 \cup E_2)$$

This leads to a natural definition of subgraph:

Definition 7 (Subgraph). *We say that G_1* is a subgraph of *G_2 if $G_1 \cup G_2 = G_2$. It follows that a subgraph of a finite graph is again finite. If G_1 is a subgraph of G_2 and $G_1, G_2 \in \mathbb{G}_{\mathcal{L}}$ for some $\mathcal{L}$, we write $G_1 \xrightarrow{\subseteq} G_2$.*

In this article, we consider the set of labels $\mathcal{L}$ to be arbitrary but fixed. The relation 'subgraph' forms a complete lattice over $\mathbb{G}_{\mathcal{L}}$, which justifies the following definition:

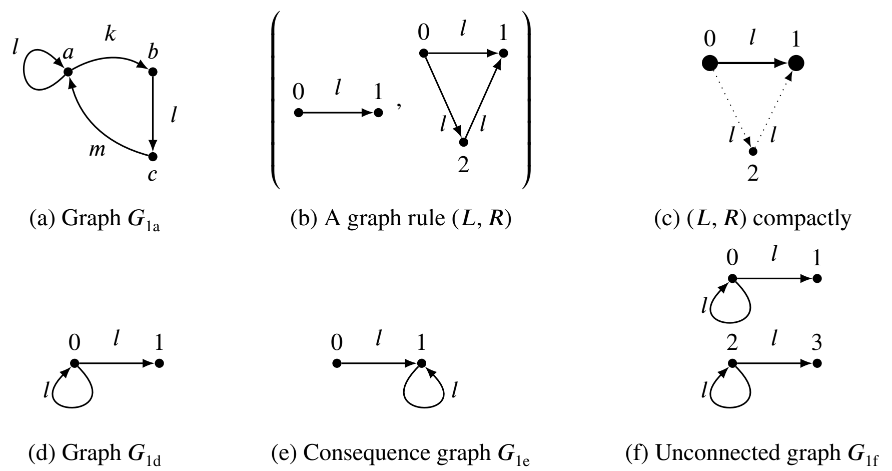
Definition 8 (Chain, Supremum). *Given a set of labels $\mathcal{L}$. We say that $S : \mathbb{N} \to \mathbb{G}_{\mathcal{L}}$ is a* chain *if for all $i \in \mathbb{N}$, $S(i)$ is a subgraph of $S(i+1)$. The union of all graphs in a chain, written $S(\infty)$, is called the* supremum, *defined as $S(\infty) = (\mathcal{L}, \bigcup_i E_i, \bigcup_i V_i)$ with $S(i) = (\mathcal{L}, E_i, V_i)$.*

The way we use graph rewriting is most closely related to the single-pushout rewriting found in the literature (e.g. [9]). In this approach, graph rules are related through a morphism that is, for instance, a partial function. Vertices in the left hand side of the rule not related to the right hand side get removed upon application of the rule. Similarly, vertices on the right hand side get inserted. In our setting, we need the application of a rule to form a chain. To make sure that we can do this, we use 'subgraph' as a condition on graph rules.

Definition 9 (Graph Rule). *A pair of graphs (L, R) is called a* graph rule *if L is a subgraph of R, and R is finite. We say that a set $\mathcal{R}$ is a* set of graph rules with labels $\mathcal{L}$ *if each $(L, R) \in \mathcal{R}$ is a graph rule, and $L, R \in \mathbb{G}_{\mathcal{L}}$.*

We proceed by giving an example of a graph rule, and do so visually. A graph can be drawn in the usual way. Figure 1a is an example of a graph with $k, l, m \in \mathcal{L}$. A picture does not specify the set of labels $\mathcal{L}$, only the set of edges and the set of vertices. An example of a graph rule is given in Figure 1b and 1c. Using the subgraph condition allows us to draw a graph rule (L, R) in a single figure, using small dots for nodes in R but not in L, and big dots and solid edges for what is in L, and therefore in R.

We present a saturation procedure, so we need to capture when a graph is 'saturated'. For this purpose, we define 'maintained', which indicates that a rule is applied sufficiently in a graph. For defining 'maintained', we first define graph embeddings:



(a) Graph G_{1a} (b) A graph rule (L, R) (c) (L, R) compactly

(d) Graph G_{1d} (e) Consequence graph G_{1e} (f) Unconnected graph G_{1f}

Fig. 1: Graphs and graph rules

Definition 10 (Embedding). *Let $G_1, G_2 \in \mathbb{G}_{\mathcal{L}}$. If $\hat{f}(G_1) \xrightarrow{\subseteq} G_2$, then (f, G_1, G_2) is an* embedding *of G_1 in G_2. In such case, we write $G_1 \xrightarrow{f} G_2$. We say that G_1 is* embedded *in G_2 if such an f exists, written $G_1 \rightarrow G_2$. It follows immediately that $G \xrightarrow{f} \hat{f}(G)$.*

We briefly explain our notations with the observation that embeddings form a category: its objects are graphs with labels $\mathcal{L}$, and its arrows are embeddings. Although $\_ \xrightarrow{\subseteq} \_ = \_ \xrightarrow{\lambda x.x} \_$, note that $G_1 \xrightarrow{\subseteq} G_2$ is only the identity arrow if $G_1 = G_2$, which is why we avoid writing $\_ \xrightarrow{id} \_$.

Definition 11 (Maintained, (Least) Consequence Graph). *A graph rule (L, R) with $L = (\mathcal{L}, V_L, E_L)$ is* maintained *in G if for every embedding $L \xrightarrow{f} G$, there is an embedding $R \xrightarrow{g} G$ such that $f(v) = g(v)$ for all $v \in V_L$. If for a set of graph rules $\mathcal{R}$, each graph rule in $\mathcal{R}$ is maintained in G, we say that G is a* consequence graph *maintaining $\mathcal{R}$. If furthermore S is a subgraph of G, and (S, G) is maintained in each consequence graph maintaining $\mathcal{R}$, then G is a* least consequence graph *of S maintaining $\mathcal{R}$.*

We use chains to find least consequence graphs. We look at two properties: 'fairness' and 'weak pushout', that help establish graphs to be a consequence graph and least, respectively. To get some intuition, and hopefully help dispel some overly optimistic conjectures, we look at some examples before defining these two properties.

We begin with an example of an embedding. Let $L = (\{k, l, m\}, \{0, 1\}, \{(l, 0, 1)\})$ and $R = (\{k, l, m\}, \{0, 1, 2\}, \{(l, 0, 1), (l, 0, 2), (l, 2, 1)\})$ be graphs. Note that (L, R) is the graph rule drawn in Figure 1b. We can embed L into the graph G_{1a} as shown in Figure 1a. A corresponding embedding is (f, L, G_{1a}) with $f(i) = a$ for $i \in \{0, 1\}$. There is also an embedding for R: (g, R, G_{1a}) with $g(i) = a$ for $i \in \{0, 1, 2\}$, which satisfies $g(i) = f(i)$ for $i \in \{0, 1\}$. However, the graph rule is not maintained, as for the embedding (f', L, G_{1a}) with $f'(0) = b$, $f'(1) = c$, there is no such g.

As an example of a consequence graph, let $\mathcal{R} = \{(L, R)\}$ with (L, R) as defined above, and let $G_{1d} = (\{k, l, m\}, \{0, 1\}, \{(l, 0, 1), (l, 0, 0)\})$, as drawn in Figure 1d. Then G_{1d} is a consequence graph maintaining $\mathcal{R}$. It is, however, not a least consequence graph of L maintaining $\mathcal{R}$, since Figure 1e gives a consequence graph maintaining $\mathcal{R}$ in which (L, G) is not maintained. We believe every least consequence graph of L maintaining $\mathcal{R}$ is infinite and even infinitely branching: loops in such consequence graphs would make that they are no longer 'least', and to every edge with label l there need to be two more of such edges in order to maintain $\mathcal{R}$.

The graph G_{1d} defined above is an example of a least consequence graph of G_{1d} maintaining $\mathcal{R}$. Graph $G_{1f} = (\{k, l, m\}, \{0, 1, 2, 3\}, \{(l, 0, 1), (l, 0, 0), (l, 2, 3), (l, 2, 2)\})$, consisting of two disjunctive copies of G_{1d}, is a least consequence graph too, see Figure 1f. If a least consequence graph is unique, it must be the empty graph.

From the definition of maintained it follows that if $L \xrightarrow{\subseteq} M \xrightarrow{\subseteq} R$ and (L, R) is maintained in G, then (L, M) is maintained in G too. Consequently, if (L, R) is maintained in a least consequence graph of L maintaining $\mathcal{R}$, then (L, R) is maintained in every consequence graph maintaining $\mathcal{R}$.

The following definition gives a sufficient condition to reach a consequence graph:

Definition 12 (Fair Chain). *Given a set of graph rules $\mathcal{R}$ and a chain S. We say that S is a* fair chain *for $\mathcal{R}$ if for each graph rule $(L, R) \in \mathcal{R}$ and for each embedding $L \xrightarrow{f} S(i)$ there exists a $j \in \mathbb{N}$ and an embedding $R \xrightarrow{g} S(j)$ with $f(v) = g(v)$ for all v in the set of vertices of L.*

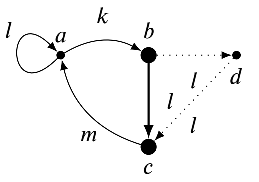
Lemma 3. *If S is a fair chain for $\mathcal{R}$, $S(\infty)$ is a consequence graph maintaining $\mathcal{R}$.*

Proof. By definition, $S(\infty)$ is a consequence graph if we can show that R is embedded in $S(\infty)$ for every L that is embedded in it. Take such an embedding $L \xrightarrow{f} S(\infty)$. Then for each edge (l, u, v) of L there is an i such that $(l, f(u), f(v))$ is an edge in $S(i)$. Take the largest such i, then f embeds L in $S(i)$, and therefore g embeds R in $S(j)$ for some j with $f(v) = g(v)$, so g also embeds R in $S(\infty)$. □

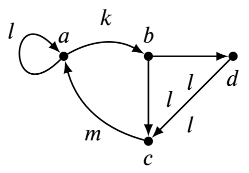
We define weak pushout step as an upper limit to each step, to ensure that a consequence graph found as a supremum of a chain built out of these steps is also a least consequence graph.

Definition 13 (Weak Pushout Step). *Let G_1 and G_2 be graphs in $\mathbb{G}_{\mathcal{L}}$, and let (L, R) be a graph rule. We say that (G_1, G_2) is a* weak pushout step *for (L, R) if the following hold:*

- *G_1 is a subgraph of G_2.*
- *There are embeddings $L \xrightarrow{f} G_1$ and $R \xrightarrow{g} G_2$ such that $f(v) = g(v)$ for all vertices in L.*
- *If there are embeddings $G_1 \xrightarrow{f'} G$ and $R \xrightarrow{g'} G$ such that $f'(f(v)) = g'(v)$ for all vertices in L, then there is an embedding $G_2 \xrightarrow{h} G$ such that $f'(v) = h(v)$ for all vertices in G_1.*



(a) A weak pushout step, compactly



(b) Result of the step

Fig. 2: On weak pushout steps

Like in our variation of graph rules, we use a weak pushout step as a variation of the categorical pushout[2] that is typically used in graph rewriting, to ensure that chains are formed. In such a (weak) pushout, the requirement of subgraphs is missing, making the entire definition symmetrical (G_1 and R can be switched). A pushout, as compared to a weak pushout, additionally requires that the embeddings f' and g' at the end of our definition, is unique. These subtle differences arise out of our need to form chains, which aren't typical structures in graph rewriting.

Definition 14 ((Simple) Weak Pushout Chain). *Let S be a chain with $S(i) = (\mathcal{L}, E_i, V_i)$, and let $\mathcal{R}$ be a set of graph rules. If for each i, either $S(i) = S(i+1)$ or there exists an $r \in \mathcal{R}$ such that $(S(i), S(i+1))$ is a weak pushout step for r, then S is a* simple weak pushout chain *under $\mathcal{R}$.* Weak pushout chains *are inductively defined:*

1. *every simple weak pushout chain under $\mathcal{R}$ is a weak pushout chain under $\mathcal{R}$,*
2. *if for each i, there exists an s, which is a weak pushout chain under $\mathcal{R}$ with $s(0) = S(i)$ and $s(\infty) = S(i+1)$, then S is a weak pushout chain under $\mathcal{R}$,*
3. *nothing else is a weak pushout chain.*

For most of this paper, it suffices to consider simple weak pushout chains.

There is a way to draw weak pushout steps that is convenient in practice, although it can leave parts implicit. On a weak pushout step (G_1, G_2) for (L, R), as drawn in Figure 2a, large vertices indicate vertices in the image of f for $L \xrightarrow{f} G_1$. The applied graph rule is that of Figure 1c. Edges in $\hat{f}(L)$ are drawn slightly thicker. The corresponding $\hat{g}$ for $R \xrightarrow{g} G_2$ is drawn with dotted lines. Since the drawing is of a weak pushout step, small vertices connected to dotted lines are in G_2 but not in G_1. The graph G_1 is the graph of Figure 1a, and G_2 is the graph in Figure 2b.

A weak pushout chain does not necessarily have a consequence graph as its supremum: we can construct a weak pushout chain with $S(i) = G$ for any graph G. However, the following holds:

Lemma 4. *If S is a weak pushout chain under $\mathcal{R}$ and $S(\infty)$ is a consequence graph maintaining $\mathcal{R}$, then $S(\infty)$ is a least consequence graph of $S(0)$ maintaining $\mathcal{R}$.*

Proof. Let G be a consequence graph. We first consider the case in which S is a simple weak pushout chain. By induction on i, we prove that $(S(0), S(i))$ is maintained in G: For

[2] Pushouts in a category with embeddings as arrows

$i = 0$, $(S(0), S(0))$ is trivially maintained in any graph. For $S(i+1)$, assume $(S(0), S(i))$ is maintained in G by induction. If $S(i+1) = S(i)$, then $S(i+1)$ is trivially embedded in G. If $S(i) \neq S(i+1)$, then $(S(i), S(i+1))$ is a weak pushout step for some $(L, R) \in \mathcal{R}$. Given an embedding $S(0) \to G$, as L is embedded in $S(i)$, transitively L is embedded in G. Since G is a consequence graph, R is embedded in G such that, by definition, there exists an embedding of $S(i+1)$ into G. We conclude that for all i, $S(i)$ is embedded in G. To conclude that $S(\infty)$ is also embedded in G, note that the individual embeddings of $S(i)$ in G have a limit (each embedding function is contained in the next by $f'(v) = h(v)$). The case in which the weak pushout chain S is not simple follows inductively from composing embeddings. Therefore $S(\infty)$ is a least consequence graph. □

A chain that is both fair and a weak pushout chain is called a fair weak pushout chain. A fair weak pushout chain has a least consequence graph as its supremum. This gives a way to create least consequence graphs, which we'll come back to in Section 5.

4 Translation between Sentences and Graph Rules

This section shows how to turn sentences into graph rules. For every sentence, there is a corresponding graph rule that is maintained if and only if the sentence holds. This allows us to use graph rules in order to reason about sentences. We introduce a translate function $\mathcal{E}_{\mathcal{L}} : \mathbb{E}_{\mathcal{L}} \to \mathbb{G}_{\mathcal{L}}$ in Definition 15 to make precise which graph belongs to a term. Lemma 6 states how the two correspond in the case of sentences of the shape $\_ \sqsubseteq \_$. Using Lemma 1, this means we can encode a set of sentences as a set of graph rules.

Definition 15 (Translation). *Given a term* $\mathbb{e}$*, we say that* $\mathcal{E}_{\mathcal{L}}(\mathbb{e})$ *is the* translation *of* $\mathbb{e}$. *We define* $\mathcal{E}_{\mathcal{L}} : \mathbb{E}_{\mathcal{L}} \to \mathbb{G}_{\mathcal{L}}$ *as follows:*

$$\begin{aligned}
\mathcal{E}_{\mathcal{L}}(l) &= (\mathcal{L}, \{0,1\}, \{(l,0,1)\})\\
\mathcal{E}_{\mathcal{L}}(\mathbb{e}^{\smile}) &= \hat{f}(\mathcal{E}_{\mathcal{L}}(\mathbb{e})) \quad \textit{with } f(v) = 1 - v \textit{ for } v < 2 \textit{ and } f(v) = v \textit{ for } v \geq 2.\\
\mathcal{E}_{\mathcal{L}}(\mathbb{e}_1 \,;\, \mathbb{e}_2) &= \hat{f}_1(\mathcal{E}_{\mathcal{L}}(\mathbb{e}_1)) \cup \hat{f}_2(\mathcal{E}_{\mathcal{L}}(\mathbb{e}_2))\\
&\qquad \textit{with } f_1(0) = 0 \textit{ and } f_1(v) = v + \left|\mathcal{E}_{\mathcal{L}}(\mathbb{e}_2)\right| - 1 \textit{ for } v \neq 0,\\
&\qquad \textit{and } f_2(0) = \left|\mathcal{E}_{\mathcal{L}}(\mathbb{e}_2)\right| \textit{ and } f_2(v) = v \textit{ for } v \neq 0.\\
\mathcal{E}_{\mathcal{L}}(\mathbb{e}_1 \sqcap \mathbb{e}_2) &= \mathcal{E}_{\mathcal{L}}(\mathbb{e}_1) \cup \hat{f}(\mathcal{E}_{\mathcal{L}}(\mathbb{e}_2))\\
&\qquad \textit{with } f(v) = v \textit{ for } v < 2 \textit{ and } f(v) = v + \left|\mathcal{E}_{\mathcal{L}}(\mathbb{e}_1)\right| - 2 \textit{ for } v \geq 2.
\end{aligned}$$

For notational convenience, $\mathcal{E}_{\mathcal{L}}\left(\mathbb{e}_L \sqsubseteq \mathbb{e}_R\right) = \left(\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L), \mathcal{E}_{\mathcal{L}}(\mathbb{e}_L \sqcap \mathbb{e}_R)\right)$. *It follows that* $\mathcal{E}_{\mathcal{L}}\left(\mathbb{e}_L \sqsubseteq \mathbb{e}_R\right)$ *is a graph rule.*

As an example of how the translation works, the graphs in Figure 1b are $\mathcal{E}_{\mathcal{L}}(l)$ and $\mathcal{E}_{\mathcal{L}}(l \sqcap l \,;\, l)$ respectively. As a whole, the graph rule in Figure 1b is $\mathcal{E}_{\mathcal{L}}(l \sqsubseteq l \,;\, l)$.

The vertices 0 and 1 of $\mathcal{E}_{\mathcal{L}}(\mathbb{e})$ can intuitively be understood as the variables x and y as in Definition 2. Lemma 5 makes this precise:

Lemma 5. $(v_0, v_1) \in [\![\mathbb{e}]\!]_G$ *if and only if there is an* f *such that* $\mathcal{E}_{\mathcal{L}}(\mathbb{e}) \xrightarrow{f} G$ *with* $f(i) = v_i$ *for* $i < 2$.

Proof. The statement follows by induction on $\mathbb{e}$, using that the vertices in $\mathcal{E}_{\mathcal{L}}(\mathbb{e})$ are $\left\{i \mid i \in \mathbb{N} \wedge i < \left|\mathcal{E}_{\mathcal{L}}(\mathbb{e})\right|\right\}$. □

We can use Lemma 5 to show a connection between graph rules and sentences:

Lemma 6. *A sentence $\mathbb{e}_L \sqsubseteq \mathbb{e}_R$ holds in G if and only if $\left(\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L), \mathcal{E}_{\mathcal{L}}(\mathbb{e}_L \sqcap \mathbb{e}_R)\right)$ is maintained in G.*

Proof. Suppose the sentence holds in G, and $\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L) \xrightarrow{f} G$. It follows from Lemma 5 that $(f(0), f(1)) \in [\![\mathbb{e}_L]\!]_G$. As the sentence holds, $(f(0), f(1)) \in [\![\mathbb{e}_R]\!]_G$. Using Lemma 5, take g with $\mathcal{E}_{\mathcal{L}}(\mathbb{e}_R) \xrightarrow{g} G$ and $f(v) = g(v)$ for $v < 2$. Following Definition 15, construct g' such that $\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L \sqcap \mathbb{e}_R) \xrightarrow{g'} G$ and $f(v) = g'(v)$ for v in the vertices of $\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L)$.

For the other direction, suppose $\left(\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L), \mathcal{E}_{\mathcal{L}}(\mathbb{e}_L \sqcap \mathbb{e}_R)\right)$ is maintained in G, and let $(x, y) \in [\![\mathbb{e}_L]\!]_G$. By Lemma 5, there is an f such that $\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L) \xrightarrow{f} G$ with $f(0) = x$ and $f(1) = y$. Since the graph rule is maintained, there is a g such that $\mathcal{E}_{\mathcal{L}}(\mathbb{e}_L \sqcap \mathbb{e}_R) \xrightarrow{g} G$ with $g(0) = f(0) = x$ and $g(1) = f(1) = y$. Again using Lemma 5, $(x, y) \in [\![\mathbb{e}_L \sqcap \mathbb{e}_R]\!]_G = [\![\mathbb{e}_L]\!]_G \cap [\![\mathbb{e}_R]\!]_G \subseteq [\![\mathbb{e}_R]\!]_G$, so the sentence holds in G. □

We use graph rules to deal with the requirements in Definition 4 to make a graph standard, in a way similar to Lemma 6. We give a set of graph rules that make checking if a standard graph exists easy: A standard graph exists provided that $[\![\bot]\!]_G = \{\}$, and that a set of additional graph rules, which we will call the standard-rules, is maintained. This motivates the following definitions:

Definition 16 (Conflict (Free)). *Let $\bot \in \mathcal{L}$. The relation symbol $\bot$ stands for an empty relation. A graph for which $[\![\bot]\!]_G = \{\}$ is* conflict free. *If $G = (\mathcal{L}, V, E)$ is conflict free, we have $\forall xy.(\bot, x, y) \notin E$, so we call any edge $(\bot, x, y)$ a* conflict.

Definition 17 (Top-rule). *Let $\top \in \mathcal{L}$. The relation symbol $\top$ stands for the full relation. We refer to the graph rule $((\mathcal{L}, \{0, 1\}, \{\}), (\mathcal{L}, \{0, 1\}, \{(\top, 0, 1)\}))$ as the* top-rule*, since any graph $G = (\mathcal{L}, V, E)$ satisfies $[\![\top]\!]_G = \{(x, y) \mid x, y \in V\}$ if and only if G maintains the top-rule.*

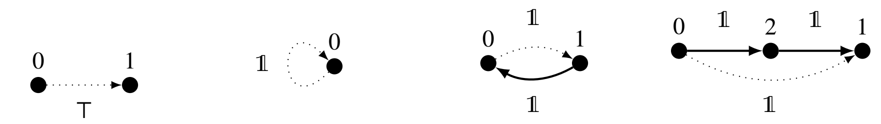
Definition 18 (Nonempty-rule). *Let $\bot, \top \in \mathcal{L}$. The graph rule $((\mathcal{L}, \{\}, \{\}), (\mathcal{L}, \{0\}, \{\}))$ is called the* nonempty-rule. *A graph $G = (\mathcal{L}, V, E)$ maintains the nonempty-rule if and only if $V \neq \{\}$.*

A conflict-free graph G that maintains the top-rule, satisfies $[\![\top]\!]_G \neq [\![\bot]\!]_G$ if and only if it maintains the nonempty-rule.

The relation symbol $\mathbb{1}$ models the identity relation $\{(x, x) \mid x \in V\}$. However, we do not let $[\![\mathbb{1}]\!]_G$ represent this relation directly. Instead, we let $\mathbb{1}$ stand for an equivalence relation and ensure that we can make a graph based on equivalence classes, in which $[\![\mathbb{1}]\!]_G = \{(x, x) \mid x \in V\}$ holds.

Definition 19 (Identity-rules). *Given a set of relation symbols $\mathcal{L}$, we say that the following set of graph rules are the* identity-rules *for $\mathcal{L}$:*

$$\left((\mathcal{L}, \{0\}, \{\}),\ (\mathcal{L}, \{0\}, \{(\mathbb{1}, 0, 0)\})\right) \tag{1}$$



(a) The top-rule (b) Identity-rule (1) (c) Identity-rule (2) (d) Identity-rule (3)

Fig. 3: Several standard-rules

$$\mathcal{E}_{\mathcal{L}}\big(\mathbb{1}^{\smile} \sqsubseteq \mathbb{1}\big) \tag{2}$$

$$\mathcal{E}_{\mathcal{L}}\big(\mathbb{1}\ ;\mathbb{1} \sqsubseteq \mathbb{1}\big) \tag{3}$$

$$\forall l \in \mathcal{L}.\ \mathcal{E}_{\mathcal{L}}\big(\mathbb{1}\ ; l\ ; \mathbb{1} \sqsubseteq l\big) \tag{4}$$

Identity-rules (1) to (4) can be understood as ensuring $\mathbb{1}$ is reflexive, symmetric, transitive, and a congruence respectively. The identity-rules hold under the standard semantics of $\mathbb{1}$, that is: if for some graph $G = (\mathcal{L}, E, V)$, we have $[\![\mathbb{1}]\!]_G = \{(x,x) \mid x \in V\}$ then the identity-rules are maintained in G. The following lemma speaks about the other direction:

Lemma 7. *Let $G = (\mathcal{L}, V, E)$ be a graph in which the identity-rules for $\mathcal{L}$ are maintained. There is an idempotent f such that $\hat{f}(G) \xrightarrow{\subseteq} G$, and $[\![\mathbb{1}]\!]_{\hat{f}(G)} = \{(f(x), f(x)) \mid x \in V\}$.*

Proof. Since the first three identity-rules for $\mathcal{L}$ are maintained in G, $[\![\mathbb{1}]\!]_G$ is an equivalence relation on V. Let f be some function that takes a canonical element from the equivalence class. It follows that $[\![\mathbb{1}]\!]_{\hat{f}(G)} = \{(f(x), f(x)) \mid x \in V\}$, and it remains to be shown that $\hat{f}(G) \xrightarrow{\subseteq} G$. For the vertices, this is immediate. For the edges: For all $(l, x, y) \in E$ we show that $(l, f(x), f(y)) \in E$. By our choice of f, $(\mathbb{1}, f(x), x) \in E$ and $(\mathbb{1}, y, f(y)) \in E$. Suppose $(l, x, y) \in E$. Since Identity-rule (4) for l is maintained in G, we get $(l, f(x), f(y)) \in E$. Therefore $\hat{f}(G) \xrightarrow{\subseteq} G$. □

Lemma 7 gives us exactly the desired semantics for $\mathbb{1}$: for $(\mathcal{L}, V', E') = \hat{f}(G)$, we have $[\![\mathbb{1}]\!]_{\hat{f}(G)} = \{(x,x) \mid x \in V'\}$. Furthermore, it states that $\hat{f}(G)$ and G are mutually embedded ($G \to \hat{f}(G)$ holds for all f).

We now proceed to introduce constants, through a set of sentences. This characterisation is similar to how points are characterized in relation algebra, see for instance work by Schmidt and Ströhlein [12]. If c is a constant, then $p = c\ ;\top$ is a point (sometimes called a right ideal). The corresponding constant can be retrieved from a point: $c = p\ ; p^{\smile}$. Our presentation here in terms of constants rather than points is a matter of personal preference. These rules state that the relation $[\![c]\!]_G$ should be nonempty, the cross-product of two sets, and a subset of the identity relation. Finally, we state that for two different constants, $[\![c_1]\!]_G$ and $[\![c_2]\!]_G$ should be non-overlapping.

Definition 20 (Constant-rules, Standard-rules). *Let $\mathcal{L}$ be a standard set of labels with constants $\mathcal{C}$, we say that the following set of graph rules are the* constant-rules *for $\mathcal{C}$:*

$$\forall c \in \mathcal{C}. \qquad \mathcal{E}_{\mathcal{L}}\big(\top \sqsubseteq \top\ ; c\ ;\top\big) \tag{5}$$

$$\forall c \in \mathcal{C}. \qquad \mathcal{E}_{\mathcal{L}}\big(c \,;\, \top \,;\, c \sqsubseteq c\big) \tag{6}$$

$$\forall c \in \mathcal{C}. \qquad \mathcal{E}_{\mathcal{L}}\big(c \sqsubseteq \mathbb{1}\big) \tag{7}$$

$$\forall c_1, c_2 \in \mathcal{C}.\ c_1 \neq c_2 \Rightarrow \qquad \mathcal{E}_{\mathcal{L}}\big(c_1 \,;\, c_2 \sqsubseteq \bot\big) \tag{8}$$

The top-, nonempty-, identity-, and constant-rules together are called the standard-rules *for C and $\mathcal{L}$, written $\mathcal{S}_{C,\mathcal{L}}$.*

Similar to our treatment of $\mathbb{1}$, we would like to find an f such that $[\![c]\!]_{\hat{f}(G)} = \{(c, c)\}$. The f of Lemma 7 gives us a graph that is isomorphic to one in which $\forall c \in C.\ [\![c]\!]_{\hat{f}(G)} = \{(c, c)\}$ holds, provided that G is conflict free and maintains the standard-rules. Lemma 8 says that, for finding a model with 'standard semantics', it suffices to find a conflict free graph that maintains the standard-rules.

Lemma 8. *Let $\mathcal{L}$ be a standard set of labels with constants C. Let $\mathcal{T}$ be a set of sentences over $\mathcal{L}$ of the shape $\_ \sqsubseteq \_$. We define $\mathcal{R} = \mathcal{S}_{C,\mathcal{L}} \cup \{\mathcal{E}_{\mathcal{L}}(t) \mid t \in \mathcal{T}\}$. Let G' be a conflict free consequence graph maintaining $\mathcal{R}$, then there is a graph $G = (\mathcal{L}, E, V)$, and functions f and g such that:*

1. *$G = \hat{f}(\hat{g}(G)) = \hat{f}(G')$, and $\hat{g}(G) \xrightarrow{\subseteq} G'$.*
2. *The graph G is standard.*
3. *Every sentence in $\mathcal{T}$ holds in G.*

Proof. We begin the proof by constructing G and f, based on $G' = (\mathcal{L}, E', V')$. By Lemma 7, there is an idempotent function h with $[\![\mathbb{1}]\!]_{\hat{h}(G')} = \{(h(x), h(x)) \mid x \in V'\}$. Top- and nonempty-rules are maintained in G', so by constant-rule (5), there are vertices $v_1, v_2 \in V'$ with $(c, v_1, v_2) \in E'$ for each $c \in C$. Let $m : C \to V'$ such that for each c, $\exists v \in V'.\ (c, m(c), v) \in E'$, therefore $\exists v \in V'.\ (c, h(m(c)), h(v)) \in E'$. Using constant-rule (7), it follows that $h(m(c)) = h(v)$, so $(c, h(m(c)), h(m(c))) \in E'$. From constant-rule (8) and that G' is conflict free, we get $(c_1, h(m(c_2)), h(m(c_2))) \notin E'$ iff $c_1 \neq c_2$. We conclude that $h \circ m$, the function that maps $c \in C$ to $h(m(c))$, is injective, so m is injective. Therefore, there is a V and an m' with $C \subseteq V$ such that $m' : V \to V'$ is bijective and $m(c) = m'(c)$ for $c \in C$. Let $f = h \circ m'$, defining $G = \hat{f}(G')$. Let g be the inverse of m', giving $\hat{f}(\hat{g}(G)) = \hat{h}(G) = G$ since h is idempotent. We have $[\![\mathbb{1}]\!]_G = \{(x, x) \mid x \in V\}$ by our choice of h. Also G is a consequence graph of $\mathcal{R}$ since $G = \hat{f}(G')$ and G' is a consequence graph of $\mathcal{R}$. From $(c, h(m(c)), h(m(c))) \in E'$ we get $(c, c) \in [\![c]\!]_G$. Using constant-rule (6) and constant-rule (7), now with $[\![\mathbb{1}]\!]_G = \{(x, x) \mid x \in V\}$, we get $\{(c, c)\} = [\![c]\!]_G$. All properties now follow. □

5 A Procedure to Find a Standard Graph

A set of sentences is satisfiable if and only if there is no i such that $S(i)$ contains a conflict in a corresponding fair weak pushout chain. This follows from the previous sections as follows: Given a set of sentences $\mathcal{T}'$ with relation symbols $\mathcal{L}$, Lemma 1 shows that we can find an equivalent set of sentences $\mathcal{T}$ such that each sentence is of the shape $\_ \sqsubseteq \_$. We derive a set of graph rules $\mathcal{R}$ that includes the standard-rules and the translation of the sentences in $\mathcal{T}$. By making a fair weak pushout chain S starting in the empty graph, we obtain a supremum that is a least consequence graph of $\mathbb{0}_{\mathcal{L}}$ maintaining $\mathcal{R}$. If this graph contains a conflict, then any graph maintaining $\mathcal{R}$ will, so $\mathcal{T}'$ is unsatisfiable. If not, we can apply Lemma 8 to find a model for $\mathcal{T}'$. In this section, we look at constructing fair weak pushout chains, based on a set of graph rules $\mathcal{R}$ that include the standard-rules.

5.1 An Algorithm for Fair Weak Pushout Chains

Assume that the set of sentences $\mathcal{T}$ is finite. Consequently, only finitely many relation symbols $\mathcal{L}$ are used in those sentences. We restrict $\mathcal{L}$ to those relation symbols that are actually used in $\mathcal{T}$. This makes the corresponding set of graph rules $\mathcal{R}$ (including the standard-rules) finite. Thus, we can construct a fair weak pushout chain for $\mathcal{R}$. Algorithm 1 gives a procedure for this.

```
1  function ProduceChain ($n \in \mathbb{N}, \mathcal{L}, E, \mathcal{R}$);
   Input: A set of edges $E$ such that $G = (\mathcal{L}, \{i \mid i \in \mathbb{N},\ i < n\}, E)$ is a graph. A finite set of
          finite graph rules $\mathcal{R}$ with relation symbols $\mathcal{L}$.
   Effect: Produces an infinite list of graphs that are a fair weak pushout chain starting in $G$.
2  Let $G = (\mathcal{L}, \{i \mid i \in \mathbb{N},\ i < n\}, E)$, produce $G$;
3  Let $W = \{\}$ be our worklist;
4  for $(L, R) \in \mathcal{R}$ do
5      Take $V$ such that $(\_, V, \_) \in L$;
6      for $f$ such that $L \xrightarrow{f} G$ do
7          if There is no $g$ such that $R \xrightarrow{g} G$ with $\forall v \in V.\ f(v) = g(v)$ then
8              Let $N \in \mathbb{N}$ be the maximum of $f(v)$;
9              Add $(N, L, R, f)$ to $W$;
10         end
11     end
12 end
13 if $W$ is empty then
14     ProduceChain($n, \mathcal{L}, E, \{\}$);
15 else
16     Take $(N, L, R, f) \in W$ such that $N$ is minimal;
17     Take $V, V'$ such that $(\_, V, \_) = L$ and $(\_, V', \_) = R$;
18     Let $\Delta = V' - V$;
19     Let $V'' = \{i \mid i \in \mathbb{N},\ i < n + |\Delta|\}$;
20     Take $g : V' \to V''$ such that $g(v) = f(v)$ for $v \in V$ and $g(\delta) \geq n$ for $\delta \in \Delta$ such that
         $g(\delta_1) \neq g(\delta_2)$ if $\delta_1 \neq \delta_2$ for $\delta_1, \delta_2 \in \Delta$;
21     Take $E'$ such that $(\mathcal{L}, V'', E') = G \cup g(R)$;
22     ProduceChain($n + |\Delta|, \mathcal{L}, E', \mathcal{R}$);
23 end
```

Algorithm 1: Construct a fair weak pushout chain starting in its input

Lemma 9. *Algorithm 1 constructs a fair weak pushout chain starting in G under $\mathcal{R}$, the limit of which is a least consequence graph of G under $\mathcal{R}$.*

Proof. The algorithm constructs a weak pushout chain, because the graph constructed on Line 21 is part of a weak pushout step for a graph rule in $\mathcal{R}$. Let $S : \mathbb{N} \to \mathbb{G}_{\mathcal{L}}$ describe the weak pushout chain generated (with $S(0) = G$). Pick an arbitrary N. Since the set of graph rules is finite, also the number of functions f with $f(v) \leq N$ that embed left-hand sides of graph rules into $S(\infty)$ is finite. For some i, all such embeddings are in $S(i)$. If an embedding is picked on Line 16, there is a g such that $R \xrightarrow{g} S(\infty)$, since such a g is added on Line 21. Therefore, for each embedding f with $f(v) \leq N$ such that $L \xrightarrow{f} G$, there is a g such that $R \xrightarrow{g} S(\infty)$ with $f(v) = g(v)$. The domain for every such f is finite, so we can pick an N for every f such that $f(v) \leq N$. Therefore, the weak pushout chain is fair. Lemma 3 and 4 complete the proof. □

The algorithm can be changed into a semi-decision procedure to decide whether the limit contains a conflict: If G contains a conflict, then any limit in which G occurs will contain the conflict. Therefore, if we are only interested in whether the limit has a conflict, we can abort the algorithm as soon as $G \cup g(R)$ in Line 21 has a conflict. Vice versa, if the limit has a conflict, then there will be a graph G in some iteration of the algorithm that has that conflict. This gives a semi-decision procedure. We can use this to decide consistency, using $\mathbb{0}_{\mathcal{L}}$ as the initial graph.

The same procedure can be used to prove entailment. Say we wish to determine if $\mathcal{T}$ entails ϕ for a problem on a standard set of labels $\mathcal{L}$, for ϕ equal to $\mathbb{e}_1 \sqsubseteq \mathbb{e}_2$. Assume without loss of generality that $l \notin \mathcal{L}$. We introduce a new label l: $\mathcal{L}' = \mathcal{L} \cup \{l\}$. Let $\mathcal{T}' = \mathcal{T} \cup \{l \sqsubseteq \mathbb{e}_1, \mathbb{e}_2 \sqcap l \sqsubseteq \bot\}$. Let $\mathcal{R}$ be the standard rules plus the derived rules of $\mathcal{T}'$. This time, run the algorithm with $(\mathcal{L}', \{0, 1\}, \{(l, 0, 1)\})$ as the initial graph: we obtain a least consequence graph maintaining $\mathcal{R}$. If this graph does not contain a conflict, there is a standard graph G in which $\mathcal{R}$ is maintained, and therefore $\mathcal{T}$ holds in G, but ϕ does not hold as $[\![l]\!]_G \subseteq [\![\mathbb{e}_1]\!]_G$ but $[\![l]\!]_G \cap [\![\mathbb{e}_1]\!]_G = \{\}$ for $[\![l]\!]_G$ nonempty, since $(\mathcal{L}', \{0, 1\}, \{(l, 0, 1)\}) \to G$. If the obtained graph does contain a conflict, then all consequence graphs of $\mathcal{R}$ with nonempty l contain a conflict. Suppose G is standard, each of $\mathcal{T}$ holds, there is a pair (labeled l) in $[\![\mathbb{e}_1]\!]_G$, and that pair is not in $[\![\mathbb{e}_2]\!]_G$, then we get a contradiction to the statement that all consequence graphs of $\mathcal{R}$ with nonempty l contain a conflict. In other words: for each standard $G \vDash \mathcal{T}$, we have $[\![\mathbb{e}_1]\!]_G \subseteq [\![\mathbb{e}_2]\!]_G$, so $\mathcal{T}$ entails ϕ. This shows that a least consequence graph can be used to decide entailment. By terminating our procedure when a conflict is found, we can prove entailment if it holds (and do not terminate otherwise). This can be extended to ϕ of the shape $\mathbb{e}_1 = \mathbb{e}_2$, by applying this procedure to both $\mathbb{e}_1 \sqsubseteq \mathbb{e}_2$ and $\mathbb{e}_2 \sqsubseteq \mathbb{e}_1$.

There is another case in which we can abort: once the graph maintains all graph rules in $\mathcal{R}$, we hit Line 14, and G is equal to the limit. In such a case, we have found the limit of the chain given by Algorithm 1, and can immediately decide whether or not it is conflict free. Unfortunately, even if conflict free graphs that maintain all graph rules exist (so by definition of least consequence graph, the limit is conflict free), we do not necessarily hit this case. Section 6 shows that we cannot hope to find an algorithm that decides whether or not a conflict free consequence graph exists.

5.2 Optimizations for Implementations

We discuss some possible optimizations for the purpose of showing correctness of the algorithm described by the author in an earlier paper [7]. The earlier algorithm is not Algorithm 1, but an optimized version thereof. We only describe a few optimizations, that suffice to show that the algorithm presented earlier is correct as well.

As optimizations, we allow changing the outcome of the algorithm, but require that the proof of Lemma 9 remains valid. In particular, instead of the graph $G \cup g(R)$ constructed on Line 21, we can make a larger graph $S(\infty)$ if $G \cup g(R) \subseteq S(\infty)$ and $S(\infty)$ is the limit of a (not necessarily fair) weak pushout chain. Through this change, the algorithm no longer constructs simple weak pushout chains, but Lemma 9 still holds.

As an instance of this, observe that we can combine graph rules, as this is a form of combining weak pushout steps: suppose (L, R) and (L', R') are graph rules in $\mathcal{R}$, such that $L' \xrightarrow{f} R$. Then we can find an R'' such that (R, R'') is a weak pushout step of (L', R'). We can then safely replace the graph rule (L, R) for (L, R'') in $\mathcal{R}$, as a weak pushout step of (L, R'') is the limit of a chain that satisfies the aforementioned condition.

Apart from changing the set of graph rules $\mathcal{R}$, we can change the algorithm such that the standard-rules are always maintained after each step. Let G' be a graph constructed in that way. According to Lemma 8, we represent the graph G' by the graph $\hat{f}(G')$, making it such that we do not need to store the relation-symbols $\top$, $\bot$, $\mathbb{1}$, or the constants in C. We do need to keep track of which vertices originally belong to which equivalence classes, in order to be able to produce the underlying G' in each step. Since the function f possibly maps several vertices of G' to one vertex in $\hat{f}(G')$, the original graph $\hat{f}_i(G'_i)$ is not necessarily a subgraph of the newly generated graph $\hat{f}_{i+1}(G'_{i+1})$. On the other hand, if we are only interested in whether or not there is a conflict in the least consequence graph, then we only need to keep track of the least vertex of each class such that the N chosen on Line 16 corresponds to a minimal embedding of f. This is precisely the algorithm proposed in the earlier paper [7], showing it is a semi-decision procedure for deciding whether a least consequence graph contains a conflict.

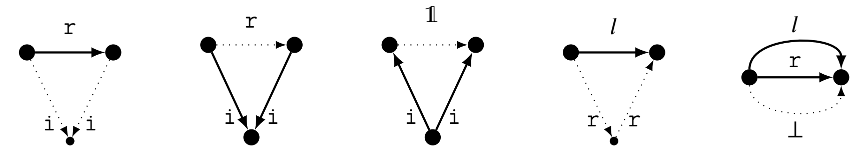
5.3 Example Run of the Optimized Algorithm

We return to one of the examples given in Section 2: the entailment problem that asks whether $\mathcal{T} = \{\mathtt{r} = \mathtt{i} \mathbin{;} \mathtt{i}^{\smile},\ \mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}\}$ entails $\mathtt{r} \mathbin{;} \mathtt{r} \sqsubseteq \mathtt{r}$. We construct a $\mathcal{T}'$ for the entailment problem as described in Section 5.1:

$$\mathcal{T}' = \{\, \mathtt{r} \sqsubseteq \mathtt{i} \mathbin{;} \mathtt{i}^{\smile} ,\ \mathtt{i} \mathbin{;} \mathtt{i}^{\smile} \sqsubseteq \mathtt{r} ,\ \mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1} ,\ l \sqsubseteq \mathtt{r} \mathbin{;} \mathtt{r} ,\ l \sqcap \mathtt{r} \sqsubseteq \bot \,\}$$

Using the translation of Section 4, Figure 4 gives the graph rules we work with. We use the optimizations just described, and do not restate the standard-rules.

We start the procedure with $n = 2$ and $E = \{(l, 0, 1)\}$. Note that per our optimizations, the self loops $(\mathbb{1}, 0, 0)$ and $(\mathbb{1}, 1, 1)$ are implicitly there, as well as all $\top$ edges. Only one rule does not hold: $l \sqsubseteq \mathtt{r} \mathbin{;} \mathtt{r}$, and consequently only one graph rule is not maintained. A pushout step for it gives $n = 3$ and $E = \{(l, 0, 1), (\mathtt{r}, 0, 2), (\mathtt{r}, 2, 1)\}$ as the next call to ProduceChain. Again only one rule is not maintained, the one for $\mathtt{r} \sqsubseteq \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}$. This time, our work-list contains two elements: one for each edge labeled $\mathtt{r}$. Both have a



(a) $\mathtt{r} \sqsubseteq \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}$ (b) $\mathtt{i} \mathbin{;} \mathtt{i}^{\smile} \sqsubseteq \mathtt{r}$ (c) $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$ (d) $\mathtt{l} \sqsubseteq \mathtt{r} \mathbin{;} \mathtt{r}$ (e) $l \sqcap \mathtt{r} \sqsubseteq \bot$

Fig. 4: Graph Rules for $\mathcal{T}'$

maximum node number of 2, so we can choose either. We pick f that maps to $(\mathtt{r}, 0, 2)$, and $n = 4$ and $E = \{(l, 0, 1), (\mathtt{r}, 0, 2), (\mathtt{r}, 2, 1), (\mathtt{i}, 0, 3), (\mathtt{i}, 2, 3)\}$. This time, $\mathtt{i} \mathbin{;} \mathtt{i}^{\smile} \sqsubseteq \mathtt{r}$ is also not maintained: $(\mathtt{i}, 0, 3)$ but $(\mathtt{r}, 0, 0)$ is missing. The highest node number assigned to N is 3 however, so we need to finish treating $\mathtt{r} \sqsubseteq \mathtt{i} \mathbin{;} \mathtt{i}^{\smile}$. Next iteration: $n = 5$ and $E = \{(l, 0, 1), (\mathtt{r}, 0, 2), (\mathtt{r}, 2, 1), (\mathtt{i}, 0, 3), (\mathtt{i}, 1, 4), (\mathtt{i}, 2, 3), (\mathtt{i}, 2, 4)\}$. Subsequently: $n = 5$ and $E = \{(l, 0, 1), (\mathtt{r}, 0, 2), (\mathtt{r}, 2, 1), (\mathtt{i}, 0, 3), (\mathtt{i}, 1, 4), (\mathtt{i}, 2, 3), (\mathtt{i}, 2, 4), (\mathtt{r}, 0, 0)\}$, then $(\mathtt{r}, 2, 2)$ is added. At this point, we have a choice again, between $\mathtt{i} \mathbin{;} \mathtt{i}^{\smile} \sqsubseteq \mathtt{r}$ and $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$. We apply the former first: after several iterations it gives us the graph that satisfies all rules except $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$:

$$
\begin{aligned}
[\![l]\!]_G &= \{(0, 1)\} \\
[\![\mathtt{i}]\!]_G &= \{(0, 3), (1, 4), (2, 3), (2, 4)\} \\
[\![\mathtt{r}]\!]_G &= \{(0, 0), (0, 2), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)\}
\end{aligned}
$$

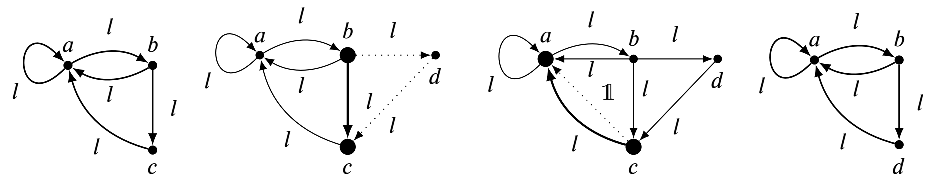
Since we did not use $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$ yet, and all other rules are satisfied up to this point, we are exactly in the place we would have been if $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$ wasn't present. This is (minus the l) the graph given in Section 2 as a possible graph our algorithm could give. If we would have handled $(\mathtt{r}, 2, 1)$ before $(\mathtt{r}, 0, 2)$ instead, we would have gotten a graph with a different numbering.

We now proceed by applying $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$. The pushout step adds $(\mathbb{1}, 3, 4)$. We have not described precisely how our optimizations proceed at this point, but we need to renumber the nodes such that 3 and 4 are identified. For preserving fairness, we renumber high to low: the node 4 is relabeled to 3. This can cause some pushout steps to get assigned a lower N, but never a higher one. We proceed with the graph G':

$$
\begin{aligned}
[\![l]\!]_{G'} &= \{(0, 1)\} \\
[\![\mathtt{i}]\!]_{G'} &= \{(0, 3), (1, 3), (2, 3)\} \\
[\![\mathtt{r}]\!]_{G'} &= \{(0, 0), (0, 2), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)\}
\end{aligned}
$$

At this point, $\mathtt{i} \mathbin{;} \mathtt{i}^{\smile} \sqsubseteq \mathtt{r}$ does not hold, and the resulting action is to insert $(\mathtt{r}, 0, 1)$. Subsequently, $l \sqcap \mathtt{r} \sqsubseteq \bot$ does not hold and we insert a conflict. We abort concluding that the entailment holds.

While we needed several iterations to conclude entailment, we saved many iterations by treating the standard rules separately. If we had applied $\mathtt{i}^{\smile} \mathbin{;} \mathtt{i} \sqsubseteq \mathbb{1}$ earlier, we would have derived the contradiction sooner.



(a) Starting graph (b) First step (c) Second step (d) Standardized

Fig. 5: Some weak pushout steps

5.4 Presentation of the Algorithm

We conclude this section with a note on the presentation in this paper. In the earlier paper, we presented the efficient implementation [7] as discussed in the previous paragraph. This does not allow us to talk about the limit of the procedure. Using the same presentation would have alleviated the need for Lemma 7. However, the simpler presentation used in this paper allows us to argue that the limit of a chain always exists. This simplifies many of the other proofs in this paper.

We give an example that shows why it is problematic to describe limits in the more involved presentation: Given the graph rules $\mathcal{E}\,(l \sqsubseteq l\,;l)$, $\mathcal{E}\left(l \sqsubseteq \mathbb{1}\right)$ and the identity-rules, Figure 5 shows a part of a weak pushout chain. Following the procedure for the given rules, we obtain the graphs in Figure 5b and 5c. After every step, we could decide to apply the identity-rules until they are maintained. If we construct a chain like this, and proceed in a similar manner as illustrated in Figure 5, we indeed construct a fair weak pushout chain. The limit of this chain is an infinite graph in which every two vertices are connected by an edge labeled l, as well as an edge labeled $\mathbb{1}$, which indeed maintains the graph rules. If we apply the mentioned optimizations and choose a representation of the graph as intended in Lemma 7 after each graph, defining the 'limit' becomes problematic: We do not need to draw edges with the label $\mathbb{1}$, as they are given by the drawn vertices, and the graph representation after the step in Figure 5c is drawn in Figure 5d. This graph is isomorphic to the one we started with, showing we end up in a sequence that alternates between two graphs. None of these graphs maintains any of the given graph rules, despite the 'underlying' chain being fair. Since a well defined limit is an important concept in many lemmas, we chose to use chains as described in this paper.

6 A Proof of Undecidability

Lemma 10. *The following decision problem is undecidable: given a set of sentences $\mathcal{T}$, is there a standard graph G in which every sentence in $\mathcal{T}$ holds?*

Proof. This proof closely follows a proof by Krisnadhi and Lutz [10] on 'conjunctive query answering'. We use a reduction from the undecidable problem whether two context free grammars have an empty intersection. This problem is given by two grammars with non-terminals N_1 and N_2, a common set of terminals T, and production rules $P_i \subseteq N_i \times (N_i \cup T)^*$ with $i \in \{1,2\}$. The sets N_1, N_2 and T are mutually disjoint. The question to be answered is whether there exists a sequence of terminals that is generated by the two starting nodes $s_i \in N_i$.

We make an encoding by choosing $\mathcal{C}, \mathcal{L}$ and $\mathcal{T}$ such that there is a standard graph in which the sentences in $\mathcal{T}$ hold if and only if the context free grammars have an empty intersection. We encode every terminal and nonterminal with corresponding relation symbols, and use the constant symbol ϵ for the empty word: $\mathcal{C} = \{\epsilon\}$ and $\mathcal{L} = \{\epsilon, \top, \bot, \mathbb{1}\} \cup N_1 \cup N_2 \cup T$. For $\mathcal{T}$ we use a sentence for each production rule, one for each terminal, and a final sentence that requires the two grammars to have an empty intersection:

$$\mathcal{T} = \left\{\sigma_0 \mathbin{;} \cdots \mathbin{;} \sigma_k \sqsubseteq n_j \mid (n_j, \sigma_0 \cdots \sigma_k) \in P_1 \cup P_2\right\} \cup \left\{\mathbb{1} \sqsubseteq t \mathbin{;} t^{\smile} \mid t \in T\right\} \cup \left\{\epsilon \mathbin{;} s_1 \mathbin{;} s_2^{\smile} \mathbin{;} \epsilon \sqsubseteq \bot\right\}$$

We show that there is a standard graph in which $\mathcal{T}$ holds if the grammars have an empty intersection, and there is no such graph if the grammars share a word. First suppose the grammars have an empty intersection. We construct a graph as follows: The vertices are words over T where ϵ is the empty word. There are edges (t, u, ut) for each non-terminal $t \in T$, edges (n, u, up) if the word p is a valid parse of $n \in N_1 \cup N_2$, and edges to make the graph standard:

$$E = \left\{(t, u, ut) \mid u \in T^*,\ t \in T\right\} \cup \{(\epsilon, \epsilon, \epsilon)\} \cup \{(\mathbb{1}, u, u) \mid u \in T^*\} \cup \{(\top, u, v) \mid u, v \in T^*\} \cup \left\{(n, u, up) \mid u, p \in T^*,\ n \in N_1 \cup N_2,\ n \text{ parses } p\right\}$$

It can be checked that $G = (\mathcal{L}, T^*, E)$ is standard and all sentences in $\mathcal{T}$ hold. In particular, $[\![s_1 \mathbin{;} s_2]\!]_G = \{\}$ as there is no word p such that s_1 parses p and s_2 parses p.

Now suppose for a proof by contradiction that $G = (\mathcal{L}, V, E)$ is a standard graph in which all sentences in $\mathcal{T}$ hold, and that $w = t_0 \cdots t_{n-1}$ is a word that is parsed by s_1 and s_2. Since $\mathcal{T}$ holds, there is a path in G with vertices $\epsilon = v_0, \ldots, v_n$ and edges $(t_0, v_0, v_1), \ldots, (t_{n-1}, v_{n-1}, v_n)$. By induction on the parse-tree of how s_1 parses w, there is an edge $(s_1, v_0, v_n) \in E$. Similarly, $(s_2, v_0, v_n) \in E$. Since $\epsilon = v_0$ and G is standard, $(\epsilon, \epsilon) \in \left[\!\left[\epsilon \mathbin{;} s_1 \mathbin{;} s_2^{\smile} \mathbin{;} \epsilon\right]\!\right]_G$. Since all sentences in $\mathcal{T}$ hold, $(\epsilon, \epsilon) \in [\![\bot]\!]_G$, contradicting that G is standard. □

Undecidability of entailment follows as a corollary: there are no standard graphs in which every sentence in $\mathcal{T}$ holds iff the sentence $\top = \bot$ is entailed.

We end with a final remark about the proof we presented. The relation symbol $\top$ and the operation $\sqcap$ were not used in the proof. Thus, this proof of undecidability holds if $\mathcal{T}$ is restricted to sentences of a simpler shape. By application of Lemma 8, we conclude that deciding whether a conflict free consequence graph under $\mathcal{R}$ exists is undecidable. Equivalently, given $\mathcal{R}$, it is undecidable to determine whether any least consequence graph under $\mathcal{R}$ contains a conflict.

7 Conclusion

In this paper, we have given a translation of sentences into graph rules, and have proven that for a graph G, a sentence is maintained in G if and only if the translated graph rule holds in G. Furthermore, when allowing the sentences to use extra relation-symbols with a dedicated meaning, we can add corresponding graph rules to ensure that this dedicated meaning is preserved. In addition, we showed that there exists a least consequence graph of a set of graph rules, which one may be able to find through a semi-decision procedure. This procedure also allows us to determine whether a set of sentences is consistent. Finally, we have shown that in a sense, we cannot do better than to give a semi-decision procedure: The problem of whether a set of sentences has a standard graph in which all graph rules hold is undecidable.

Our procedure can partially automate preserving invariants in information systems. Its implementation and evaluation is foreseen in Ampersand, but considered outside the scope of this paper.

Acknowledgements We thank Wolfram Kahl and Stef Joosten for their thoughts and comments on this paper and earlier versions. Part of the research presented in this paper was performed at the University of Innsbruck, Austria, supported by the Austrian Science Fund (FWF) project Y757. Supported by The Netherlands Organisation for Scientific Research (NWO) project 639.023.710.

References

1. Baader, F.: The description logic handbook: Theory, implementation and applications. Cambridge university press (2003)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence. pp. 364–369. IJCAI'05, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005), `http://dl.acm.org/citation.cfm?id=1642293.1642351`
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions. Washington, DC, USA (April 2008)
4. Freyd, P., Scedrov, A.: Categories, allegories. North-Holland Publishing Co. (1990)
5. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM) 11(2), 256–290 (2002)
6. Jackson, D., Schechter, I., Shlyakhter, I.: Alcoa: the alloy constraint analyzer. In: ICSE (2000)
7. Joosten, S.J.C.: Parsing and printing of and with triples. In: International Conference on Relational and Algebraic Methods in Computer Science. pp. 159–176. Springer (2017)
8. Joosten, S.: Software development in relation algebra with Ampersand. In: Pous, D., Struth, G., Höfner, P. (eds.) Relational and Algebraic Methods in Computer Science: 16th International Conference (RAMICS). Springer International Publishing (2017)
9. Kahl, W.: A relation-algebraic approach to graph structure transformation. In: International Conference on Relational Methods in Computer Science. pp. 1–14. Springer (2001)
10. Krisnadhi, A., Lutz, C.: Data complexity in the $\mathcal{EL}$ family of description logics. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 333–347. Springer (2007)
11. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM 12(1), 23–41 (Jan 1965), `http://doi.acm.org/10.1145/321250.321253`
12. Schmidt, G., Ströhlein, T.: Relation algebras: Concept of points and representability. Discrete Mathematics 54(1), 83 – 92 (1985), `http://www.sciencedirect.com/science/article/pii/0012365X85900640`
13. Spivey, J.: The Z Notation: A reference manual. International Series in Computer Science, Prentice Hall, New York, 2nd edn. (1992)