





Specification and Automatic Verification of Computational Reductions

Julien Grange  

LACL, Université Paris-Est Créteil, France

Fabian Vehlken  

Ruhr University Bochum, Germany

Nils Vortmeier  

Ruhr University Bochum, Germany

Thomas Zeume  

Ruhr University Bochum, Germany

Abstract

We are interested in the following validation problem for computational reductions: for algorithmic problems P and P^* , is a given candidate reduction indeed a reduction from P to P^* ? Unsurprisingly, this problem is undecidable even for very restricted classes of reductions. This leads to the question: Is there a natural, expressive class of reductions for which the validation problem can be attacked algorithmically? We answer this question positively by introducing an easy-to-use graphical specification mechanism for computational reductions, called cookbook reductions. We show that cookbook reductions are sufficiently expressive to cover many classical graph reductions and expressive enough so that SAT remains NP-complete (in the presence of a linear order). Surprisingly, the validation problem is decidable for natural and expressive subclasses of cookbook reductions.

2012 ACM Subject Classification Theory of computation \rightarrow Logic; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Computational reductions, automatic verification, decidability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.56

Related Version *Full Version*: <https://arxiv.org/abs/2407.04037>

Funding *Fabian Vehlken*: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.

Thomas Zeume: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.

1 Introduction

Computational reductions are one of the most powerful concepts in theoretical computer science. They are used, among others, to establish undecidability in computability theory and hardness of algorithmic problems in computational complexity theory. In practical applications, reductions help to harness the power of modern SAT solvers for other problems.

For teaching reductions in introductory courses, instructors often design learning tasks for (i) understanding the computational problems involved, (ii) exploring existing reductions via examples, and (iii) designing reductions between computational problems. Technological teaching support so far is only provided for (i) and (ii), likely because these tasks are typically easy to illustrate and checking student solutions is algorithmically straightforward.

Providing teaching support for (iii) requires to address the foundational question: Is there a language for specifying reductions that can express a variety of reductions, but is also algorithmically accessible? In particular, it should be possible to test whether a candidate for a reduction provided by a student is indeed a valid reduction, preferably also providing a counterexample in case a submitted answer is incorrect.



© Julien Grange, Fabian Vehlken, Nils Vortmeier, and Thomas Zeume;
licensed under Creative Commons License CC-BY 4.0

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).

Editors: Rastislav Kráľovič and Antonín Kučera; Article No. 56; pp. 56:1–56:14



Leibniz International Proceedings in Informatics

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

In this paper, we propose such a specification language for reductions and study variants of the following algorithmic problem, parameterized by a class \mathcal{R} of reductions and complexity classes \mathcal{C} and \mathcal{C}^* :

Problem: REDUCTION? $(\mathcal{C}, \mathcal{C}^*, \mathcal{R})$

Input: Algorithmic problems $P \in \mathcal{C}$, $P^* \in \mathcal{C}^*$, and a reduction $\rho \in \mathcal{R}$.

Question: Is ρ a reduction from P to P^* ?

More precisely, our contributions are twofold:

- We propose a graphical and modular specification language for reductions, which we call *cookbook reductions* (Section 3). Its design is inspired by “building blocks” such as local replacement of nodes, edges, . . . [6] that are used in the context of many standard reductions. Cookbook reductions allow these building blocks to be combined in a simple, stepwise fashion. We compare the expressive power of cookbook reductions with standard methods of specifying reductions. Specifically, we relate cookbook reductions to quantifier-free first-order interpretations (Section 4.2) and observe that SAT remains NP-hard under cookbook reductions, assuming the presence of a linear order (Corollary 3).
- We study variants of the decision problem REDUCTION?, obtained by choosing different classes of reduction candidates and by either fixing the algorithmic problems P, P^* or by fixing complexity classes $\mathcal{C}, \mathcal{C}^*$ and letting $P \in \mathcal{C}, P^* \in \mathcal{C}^*$ be part of the input (Section 5). Not surprisingly, REDUCTION? is undecidable for many restricted variants (Theorem 4). To our surprise, several interesting variants remain decidable: for example, REDUCTION? is decidable for an arbitrary fixed problem P and fixed P^* expressible in monadic second-order logic¹, if reduction candidates are from the subclass of cookbook reductions that allows local replacements of edges by a gadget graph (Theorem 10). Also, for some concrete choices of problems P, P^* , we characterize valid reductions; the characterizations can be used to generate counterexamples for invalid candidates, which is particularly relevant in teaching contexts.

Related work. Restricted specification languages have also been used in [3, 9] in the context of learning reductions algorithmically. Reductions that are similar in spirit to cookbook reductions due to their stepwise fashion are pp-constructions and gadget reductions in the realm of (finite) constraint satisfaction problems [1, 5, 2].

2 Preliminaries

We assume familiarity with basic notions from finite model theory [10].

A (*purely relational*) *schema* $\sigma = \{R_1, \dots, R_m\}$ is a set of relation symbols R_i with associated *arities* $\text{Ar}(R_i)$. A (finite) σ -structure $\mathcal{S} = (U, R_1^{\mathcal{S}}, \dots, R_m^{\mathcal{S}})$ consists of a finite set U , called the *universe* or the *domain* of \mathcal{S} , and relations $R_i^{\mathcal{S}} \subseteq U^{\text{Ar}(R_i)}$. If clear from the context, we sometimes omit the superscript \mathcal{S} . We also refer to the domain of \mathcal{S} as $\text{dom}(\mathcal{S})$. We write FO_k for the set of all first-order formulas with quantifier depth at most k . The FO_k -*type* of a σ -structure \mathcal{S} is the set of all FO_k formulas over schema σ that \mathcal{S} satisfies. Two structures $\mathcal{S}_1, \mathcal{S}_2$ are *FO-similar* up to quantifier depth k , written $\mathcal{S}_1 \equiv_k^{\text{FO}} \mathcal{S}_2$, if they have the same FO_k -type.

An *isomorphism type* of σ -structures is an equivalence class of the equivalence relation “is isomorphic to”. We represent an isomorphism type by an arbitrarily fixed σ -structure \mathfrak{t} with universe $\{1, \dots, k\}$, for the appropriate number k , from that equivalence class. The

¹ This logic extends first-order logic with quantification over sets and can express for example the NP-complete problem 3-COLORABILITY.

<p><i>Problem:</i> CLIQUE <i>Input:</i> Undirected graph $G = (V, E)$, $k \in \mathbb{N}$ <i>Question:</i> Is there a set $U \subseteq V$ of size k with $(u, v) \in E$ for all $u, v \in U$?</p>	<p><i>Problem:</i> INDEPENDENTSET <i>Input:</i> Undirected graph $G = (V, E)$, $k \in \mathbb{N}$ <i>Question:</i> Is there a set $U \subseteq V$ of size k with $(u, v) \notin E$ for all $u, v \in U$?</p>
<p><i>Problem:</i> VERTEXCOVER <i>Input:</i> Undirected graph $G = (V, E)$, $k \in \mathbb{N}$ <i>Question:</i> Is there a set $U \subseteq V$ of size at most k such that $u \in U$ or $v \in U$ for all $(u, v) \in E$?</p>	<p><i>Problem:</i> FEEDBACKVERTEXSET <i>Input:</i> Undirected graph $G = (V, E)$, $k \in \mathbb{N}$ <i>Question:</i> Is there a set $U \subseteq V$ of size at most k such that removing U from G yields a cycle-free graph?</p>
<p><i>Problem:</i> HAMCYCLE_U <i>Input:</i> Undirected graph $G = (V, E)$ <i>Question:</i> Is there an undirected cycle in G that passes each node exactly once?</p>	<p><i>Problem:</i> HAMCYCLE_D <i>Input:</i> Directed graph $G = (V, E)$ <i>Question:</i> Is there a directed cycle in G that passes each node exactly once?</p>

■ **Figure 1** Collection of algorithmic problems considered in the paper.

arity of an isomorphism type is the universe size of its representative. Often, we identify an isomorphism type with its representative \mathfrak{t} . Given a structure \mathcal{S} and a subset A of its universe, we write $\mathfrak{tp}_{\mathcal{S}}(A)$ for the isomorphism type of $\mathcal{S}[A]$, so, the isomorphism type of the substructure of \mathcal{S} that is induced by A . We write $\mathfrak{tp}(A)$ if \mathcal{S} is clear from the context and call $\mathfrak{tp}(A)$ the isomorphism type of A .

An *embedding* π of a structure \mathcal{S} into a structure \mathcal{S}^* is an injective mapping from the domain of \mathcal{S} into the domain of \mathcal{S}^* that is an isomorphism between \mathcal{S} and the substructure of \mathcal{S}^* that is induced by the image of π . So, an embedding π witnesses that \mathcal{S}^* contains an isomorphic copy of \mathcal{S} as an induced substructure.

An (*algorithmic*) *problem* P is an isomorphism-closed set of σ -structures, for some schema σ . A *reduction* ρ from a problem P over schema σ to a problem P^* over schema σ^* is a mapping from σ -structures to σ^* -structures such that $\mathcal{S} \in P \Leftrightarrow \rho(\mathcal{S}) \in P^*$, for every σ -structure \mathcal{S} . A *d-dimensional first-order interpretation* from σ -structures to σ^* -structures is a tuple $\Psi = (\varphi_U(\bar{x}), \varphi_{\sim}(\bar{x}_1, \bar{x}_2), (\varphi_R(\bar{x}_1, \dots, \bar{x}_{\text{Ar}(R)}))_{R \in \sigma^*})$ of first-order formulas over schema σ , where each tuple $\bar{x} = (x_1, \dots, x_d)$, $\bar{x}_i = (x_{i,1}, \dots, x_{i,d})$ consists of d variables. For a given σ -structure \mathcal{S} with universe U , let $\hat{\Psi}(\mathcal{S})$ be the σ^* -structure with universe $\hat{U} = \{\bar{a} \in U^d \mid \mathcal{S} \models \varphi_U(\bar{a})\}$ and relations $R^{\hat{\Psi}(\mathcal{S})} = \{(\bar{a}_1, \dots, \bar{a}_{\text{Ar}(R)}) \in \hat{U}^{\text{Ar}(R)} \mid \mathcal{S} \models \varphi_R(\bar{a}_1, \dots, \bar{a}_{\text{Ar}(R)})\}$ for each $R \in \sigma^*$. We demand that for every σ -structure \mathcal{S} , the binary relation $\sim^{\hat{\Psi}(\mathcal{S})} = \{(\bar{a}_1, \bar{a}_2) \in \hat{U}^2 \mid \mathcal{S} \models \varphi_{\sim}(\bar{a}_1, \bar{a}_2)\}$ is a congruence relation on $\hat{\Psi}(\mathcal{S})$, that is, an equivalence relation on the universe that is compatible with the relations of the structure. For a given σ -structure \mathcal{S} , the interpretation Ψ defines the σ^* -structure $\Psi(\mathcal{S})$ that is the quotient structure of $\hat{\Psi}(\mathcal{S})$ with respect to $\sim^{\hat{\Psi}(\mathcal{S})}$, that is, the structure that results from $\hat{\Psi}(\mathcal{S})$ by restricting the universe to only one element for every equivalence class of $\sim^{\hat{\Psi}(\mathcal{S})}$.

Most of our examples will be drawn from the algorithmic problems from Figure 1. We also consider variants of some of these problems where k is a fixed parameter, e.g. k -CLIQUE asks, given a graph G , whether there is a k -clique in G .

For a natural number n , we sometimes write $[n]$ for the set $\{1, \dots, n\}$.

3 Cookbook reductions: A specification language for reductions


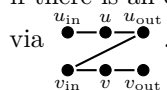
When looking for a reduction, one approach by typical experts is to subsequently try building blocks that they have encountered in the context of other reductions before. For example, Garey and Johnson [6, Section 3.2] discuss common proof techniques like local replacements

that occur in many standard reductions. An example is the standard reduction from the problem of finding a directed Hamiltonian cycle to finding an undirected Hamiltonian cycle that transforms a directed graph into an undirected graph by mapping each node $\begin{matrix} \rhd \\ \bullet \\ \lhd \end{matrix}$ to a small gadget $\begin{matrix} \rhd & \bullet & \bullet & \bullet & \lhd \\ & v_{in} & v & v_{out} & \end{matrix}$. Constructing such node gadgets is one of the typical building blocks when designing reductions.

Our approach towards constructing a specification language for reductions is to (1) identify common building blocks used in computational reductions between graph problems, and to (2) abstract these building blocks into a more general specification language. The resulting language is reasonably broad and, due to its modular and graphical nature, easy to use.

3.1 Building blocks and recipes

Many computational reductions can be crafted from a small set of common building blocks. For reductions between graph problems, some such building blocks are the following:

- *Edge gadgets* replace each edge (u, v) of the source instance uniformly by a graph. For example, in the standard reduction from VERTEXCOVER to FEEDBACKVERTEXSET, every edge $\bullet_u - \bullet_v$ in the source instance is replaced by a triangle .
- *Node gadgets* replace each node of the source instance uniformly by a graph and specify how these graphs are connected. For example, in the standard reduction from HAMCYCLE_d to HAMCYCLE_u, every node \bullet_v in the source instance is replaced by a path $\bullet_{v_{in}} - \bullet_v - \bullet_{v_{out}}$ and if there is an edge (u, v) in the source instance, then the paths for u and v are connected via .
- *Global gadgets* introduce a (global) graph and specify how each node of this graph is connected to the nodes of the source instance. For example, in the simple reduction from 3-CLIQUE to 4-CLIQUE, a single node \bullet_g is introduced as global graph and each node v of the source instance is connected to g via an edge $\bullet_g - \bullet_v$.

These building blocks have in common that target instances of reductions are obtained from source instances by following simple, recipe-like steps of the form “for every occurrence of a substructure t in the source instance, create a copy of the substructure t^* in the target structure”. For example, the recipes for the above reductions are as follows:

- Reducing k -VERTEXCOVER to k -FEEDBACKVERTEXSET: For every node v in the source instance, create a node v^* in the target instance. For every edge (u, v) in the source instance, create a node w_{uv}^* and edges $(u^*, v^*), (v^*, w_{uv}^*), (w_{uv}^*, u^*)$ in the target instance.
- Reducing HAMCYCLE_d to HAMCYCLE_u: For every node v in the source instance, create nodes v_{in}^*, v^*, v_{out}^* in the target instance and connect them as a path. For every directed edge (u, v) in the source instance, create the undirected edge (u_{out}^*, v_{in}^*) in the target instance.
- Reducing 3-CLIQUE to 4-CLIQUE: Create a node g^* in the target instance. For every node v of the source instance, create a node v^* in the target instance and add the edge (v^*, g^*) . Copy all edges (u, v) of the source instance as edges (u^*, v^*) to the target instance.

Other reductions can also be phrased in this form, for instance:

- Reducing k -CLIQUE to k -INDEPENDENTSET: First, for every node v of the source instance, create a node v^* in the target instance. Then, for every pair u, v of nodes that are not connected by an edge in the source instance, create an edge (u^*, v^*) in the target instance.

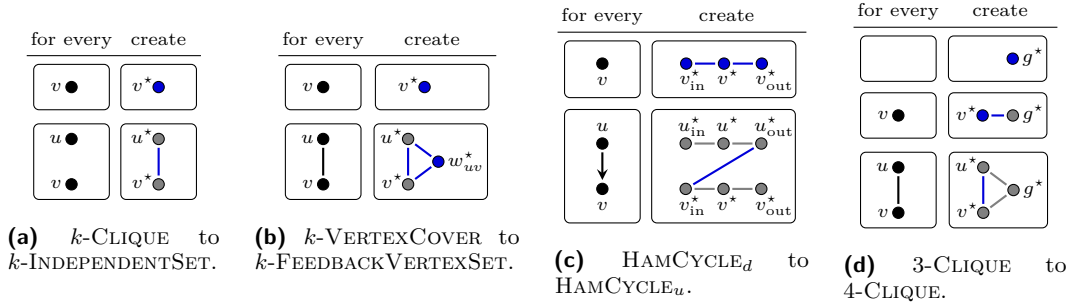


Figure 2 Graphical representations of four reductions. The reductions are applied stepwise, from the top-most step to the bottom-most step. Nodes and edges coloured blue are created in this step, grey nodes and edges were created in a previous step.

Reductions specified this way capture building blocks such as the ones from [6] and are usually easy to understand, often much more than their presentation as algorithms or as logical interpretations. Such reductions can also easily be specified graphically, see Figure 2.

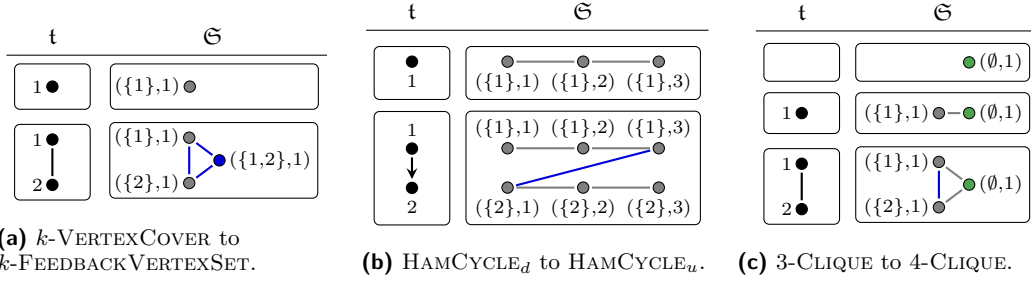
3.2 Cookbook reductions: Formalization

We now formalize cookbook reductions as such recipe-style descriptions of computational reductions. In general, graphical representations as in Figure 2 can be used to specify a cookbook reduction. In this section, we discuss the formal syntax and semantics.

Intuitively, a reduction specified in our formalism builds, based on a source structure, the target structure in a sequence of stages, starting from an empty structure. At first, independent of the source structure, some global elements and tuples over these elements may be introduced to the target structure. Then, for every element of the source structure, a set of elements may be added, together with tuples that may also incorporate the elements that were introduced in the step before. The added elements and tuples depend on the (atomic) type of the respective element of the source structure. In further stages, elements are analogously introduced for every set of two, three, \dots , elements of the source structure, depending on the type of these sets.

Syntactically, a *cookbook reduction* ρ from σ -structures to σ^* -structures is a finite set $\rho = \{(t_1, \mathfrak{S}_1), \dots, (t_m, \mathfrak{S}_m)\}$ of pairs which we call *instructions*. The structures t_i are σ -structures with universe $\{1, \dots, k_i\}$, for some natural number $k_i \geq 0$, that represent pairwise distinct isomorphism types of σ -structures. The set $\{t_1, \dots, t_m\}$ is the *support* of ρ . The *arity* of ρ is the maximal arity of an isomorphism type in the support of ρ . The structures \mathfrak{S}_i are over the schema σ^* . For $(t_i, \mathfrak{S}_i) \in \rho$, we also refer to \mathfrak{S}_i as $\mathfrak{S}(t_i)$. Each instruction (t, \mathfrak{S}) , where t has the universe $[k] = \{1, \dots, k\}$, satisfies the following properties:

- (P1) The universe $\text{dom}(\mathfrak{S})$ of \mathfrak{S} consists of elements (A, j) , where $A \subseteq [k]$ and $j \geq 1$. If $(A, j) \in \text{dom}(\mathfrak{S})$ with $j > 1$, then also $(A, 1), \dots, (A, j - 1)$ are in $\text{dom}(\mathfrak{S})$.
- (P2) For any $(A, j) \in \text{dom}(\mathfrak{S})$ with $A \subsetneq [k]$, the isomorphism type $t' = \mathfrak{tp}_t(A)$ is in the support of ρ and $(\{1, \dots, |A|\}, j)$ is in $\text{dom}(\mathfrak{S}(t'))$.
- (P3) For any tuple $((A_1, j_1), \dots, (A_\ell, j_\ell))$ in any relation of \mathfrak{S} with $\bigcup_{i \leq \ell} A_i \subsetneq [k]$, the isomorphism type $\mathfrak{tp}_t(\bigcup_{i \leq \ell} A_i)$ is in the support of ρ .
- (P4) For any $(t', \mathfrak{S}') \in \rho$ and any $A \subsetneq [k]$ with $\mathfrak{tp}_t(A) = t'$, there is an isomorphism π from t' to $t[A]$ such that the injective mapping $\hat{\pi}$ with $\hat{\pi}((A', j')) = (\pi(A'), j')$, for all (A', j') in $\text{dom}(\mathfrak{S}')$, is an embedding from \mathfrak{S}' into \mathfrak{S} .



■ **Figure 3** Three reductions formalized as cookbook reductions. Nodes introduced for type \mathbf{t}_\emptyset are coloured green, nodes and edges introduced for type \mathbf{t}_\bullet are coloured grey, and nodes and edges introduced for types $\mathbf{t}_{\bullet-\bullet}$ and $\mathbf{t}_{\bullet\rightarrow\bullet}$ are coloured blue. Compare to Figure 2(b), (c), and (d).

A cookbook reduction has to satisfy a further, semantic property, which we state after defining the semantics.

See Figure 3 for examples of cookbook reductions.

We give some more explanations for the conditions (P1)–(P4). Intuitively, an instruction $(\mathbf{t}, \mathfrak{S}) \in \rho$ means that for every occurrence of the type \mathbf{t} in the source structure, a copy of the structure \mathfrak{S} is included in the target structure. The conditions (P1) and (P2) are concerned with the universe $\text{dom}(\mathfrak{S})$ of \mathfrak{S} . If \mathbf{t} is an isomorphism type of k elements, the universe of \mathfrak{S} partly consists of elements $([k], 1), \dots, ([k], m)$, for some number m . These elements are added to the target structure for every occurrence of the type \mathbf{t} . We also call these m elements *fresh* and write $\#\text{fresh}(\mathbf{t}) = m$ (and $\#\text{fresh}(\mathbf{t}) = 0$ if no such element exists). The universe of \mathfrak{S} also contains further elements of the form (A, j) with $A \subsetneq [k]$. These represent elements that are added for sets of elements with size $k' < k$ (in the intuitive explanation: in previous stages). If such an element (A, j) occurs in the universe of \mathfrak{S} , there has to be a corresponding instruction to add this element, that is, the type \mathbf{t}' of the set A in \mathbf{t} has to be in the support of ρ and the element $([k'], j)$ has to be a fresh element in $\mathfrak{S}(\mathbf{t}')$.

The conditions (P3) and (P4) concern the relations of \mathfrak{S} . A tuple $((A_1, j_1), \dots, (A_\ell, j_\ell))$ with $\bigcup_{i \leq \ell} A_i = [k]$ in a relation of \mathfrak{S} says that this tuple is to be added to the target structure for every set of elements of type \mathbf{t} . No further conditions on these tuples are imposed by (P3) and (P4). If $A' \stackrel{\text{def}}{=} \bigcup_{i \leq \ell} A_i$ is a proper subset of $[k]$, this tuple is added for the subset A' of elements (intuitively: in a previous stage). Again, there needs to be another instruction that adds this tuple, that is, the isomorphism type \mathbf{t}' of A' needs to be in the support of ρ .

If a subtype \mathbf{t}' of \mathbf{t} is in the support of ρ then the corresponding instruction $(\mathbf{t}', \mathfrak{S}')$ needs to be respected: for every occurrence of \mathbf{t}' in \mathbf{t} , a copy of the structure \mathfrak{S}' needs to be present in \mathfrak{S} . Formally, if a set $A \subsetneq [k]$ with $|A| = k'$ has type \mathbf{t}' in \mathbf{t} , as witnessed by some isomorphism π from \mathbf{t}' to $\mathbf{t}[A]$, the substructure of \mathfrak{S} that is induced by the set $\{(A_i, j_i) \mid A_i \subseteq \pi([k'])\}$ is isomorphic to \mathfrak{S}' .

We now define the semantics of cookbook reductions. A cookbook reduction $\rho = \{(\mathbf{t}_1, \mathfrak{S}_1), \dots, (\mathbf{t}_m, \mathfrak{S}_m)\}$ maps a σ -structure \mathcal{S} to a set $\rho(\mathcal{S})$ of σ^* -structures, where σ is the schema of the isomorphism types \mathbf{t}_i and σ^* is the schema of the structures \mathfrak{S}_i . For some σ -structure \mathcal{S} , the σ^* -structure \mathcal{S}^* is in $\rho(\mathcal{S})$ if the following conditions hold:

- (S1) The universe $\text{dom}(\mathcal{S}^*)$ of \mathcal{S}^* consists of exactly those elements (A, j) with $A \subseteq \text{dom}(\mathcal{S})$ such that
- the isomorphism type $\mathbf{t} = \mathbf{tp}_{\mathcal{S}}(A)$ is in the support of ρ , and
 - the structure \mathfrak{S} with $(\mathbf{t}, \mathfrak{S}) \in \rho$ has the element $(\{1, \dots, |A|\}, j)$ in its universe.
- (S2) If a tuple $((A_1, j_1), \dots, (A_\ell, j_\ell))$ is in some relation $R^{\mathcal{S}^*}$ of \mathcal{S}^* , for any $R \in \sigma^*$, then the isomorphism type $\mathbf{tp}_{\mathcal{S}}(\bigcup_{i \leq \ell} A_i)$ is in the support of ρ .

(S3) For any $(\mathfrak{t}, \mathfrak{G}) \in \rho$ and any $A \subseteq \text{dom}(\mathcal{S})$ with $\text{tp}_{\mathcal{S}}(A) = \mathfrak{t}$, there is an isomorphism π from \mathfrak{t} to $\mathcal{S}[A]$ such that the injective mapping $\hat{\pi}$ with $\hat{\pi}((A', j')) = (\pi(A'), j')$, for all (A', j') in the universe of \mathfrak{G} , is an embedding from \mathfrak{G} into \mathcal{S}^* .

Intuitively, these conditions state that the elements (S1) and tuples (S3) of \mathcal{S}^* can be obtained by transforming occurrences of an isomorphism type \mathfrak{t} in \mathcal{S} into \mathfrak{G} , for any $(\mathfrak{t}, \mathfrak{G}) \in \rho$, and that no other tuples are present (S2).

A cookbook reduction ρ needs to satisfy the following semantic property².

(P5) For every σ -structure \mathcal{S} , the set $\rho(\mathcal{S})$ is a non-empty set of isomorphic structures. Abusing notation, we usually write $\rho(\mathcal{S})$ to denote some arbitrary structure $\mathcal{S}^* \in \rho(\mathcal{S})$.

4 The expressive power of cookbook reductions

In this section we study the expressive power of cookbook reductions. First, we explain how the building blocks from Section 3 are captured by restricted cookbook reductions. Afterwards, we discuss the expressive power of general cookbook reductions and relate them to quantifier-free first-order interpretations.

4.1 From building blocks to cookbook reductions

Cookbook reductions are a versatile reduction concept and as we have seen in the examples depicted in Figure 2 and Figure 3, many reductions have a small and easily understandable representation as cookbook reductions that have only few isomorphism types in their support.

In fact, the building blocks for graph problems that we discussed as motivation for cookbook reductions can be recovered as restricted variants of cookbook reductions. For undirected graphs with only the binary edge relation E and no self-loops, only four isomorphism types of arity at most 2 are relevant: the type \mathfrak{t}_\emptyset of the graph with 0 nodes, the type \mathfrak{t}_\bullet of a single node, the type $\mathfrak{t}_{\bullet-\bullet}$ of an undirected edge, and the type $\mathfrak{t}_{\bullet\bullet}$ of non-edges.

We obtain the following characterization:

- For a *global gadget reduction*, the inserted global graph $\mathfrak{G}(\mathfrak{t}_\emptyset)$ is arbitrary. Nodes of the source instance are copied, so we fix $\#\text{fresh}(\mathfrak{t}_\bullet) = 1$, but allow $\mathfrak{G}(\mathfrak{t}_\bullet)$ to arbitrarily select nodes from the global graph that are connected to every source node. Edges of the source are copied, so $\#\text{fresh}(\mathfrak{t}_{\bullet-\bullet}) = 0$ and $\mathfrak{G}(\mathfrak{t}_{\bullet-\bullet})$ just adds the edge.
- A *node gadget reduction* replaces every node by some gadget, so $\mathfrak{G}(\mathfrak{t}_\bullet)$ is arbitrary. The reduction can define how these gadgets are connected in case there is an edge between the corresponding nodes in the source instance, resulting in $\#\text{fresh}(\mathfrak{t}_{\bullet-\bullet}) = 0$ and $\mathfrak{G}(\mathfrak{t}_{\bullet-\bullet})$ being arbitrary apart from that.
- An *edge gadget reduction* replaces edges by some gadget. As every node from the source is copied to the target, $\mathfrak{G}(\mathfrak{t}_\bullet)$ is a single node. We allow any symmetric $\mathfrak{G}(\mathfrak{t}_{\bullet-\bullet})$.

Only the mentioned isomorphism types are in the support of the cookbook reduction.

A similar characterization holds if the source graph is directed.

Global, node or edge gadget reductions constitute expressive subclasses of cookbook reductions that are relatively easy to comprehend. More fragments can be defined by, e.g., setting an upper bound for $\#\text{fresh}(\mathfrak{t}_\bullet)$ in a node gadget reduction, or selecting a different set

² For global and node gadget reductions as introduced in Section 3.1, this property is trivially satisfied, for edge gadget reductions it is satisfied if the gadget graph is symmetric. In general, the following syntactic restriction is necessary: For every $(\mathfrak{t}, \mathfrak{G}) \in \rho$ and any automorphism π of \mathfrak{t} there is an automorphism $\hat{\pi}$ of \mathfrak{G} with $\hat{\pi}((A, j)) = (\pi(A), j)$, for any (A, j) in the universe of \mathfrak{G} .

of isomorphism types \mathfrak{t} for which $\mathfrak{S}(\mathfrak{t})$ needs to be provided. This modularity of cookbook reductions helps finding decidable cases of the REDUCTION? problem. In a teaching context, instructors can select the degree of freedom students have.

4.2 Relating cookbook reductions to quantifier-free interpretations

Quantifier-free first-order (FO) interpretations constitute a widely-used class of reductions with very low complexity, see, e.g., [7]. They are still expressive enough to show hardness of problems: SAT, the satisfiability problem for propositional formulas, is NP-hard even under quantifier-free FO interpretations [4].

In this section, we show that cookbook reductions can be expressed as quantifier-free FO interpretations. If we assume a linear order on the input structures, mildly restricted quantifier-free FO interpretations can be expressed as cookbook reductions. It follows that if input structures are linearly ordered, SAT is NP-hard under cookbook reductions.

We say that two reductions ρ_1 and ρ_2 are *equivalent* for a source structure \mathcal{S} over the appropriate schema, if the target structures $\rho_1(\mathcal{S})$ and $\rho_2(\mathcal{S})$ are isomorphic.

► **Theorem 1.** *For every cookbook reduction ρ there is a d -dimensional quantifier-free first-order interpretation Ψ , for some number d , such that ρ and Ψ are equivalent for every structure with at least 2 elements.*

Proof idea. Suppose that for a cookbook reduction $\rho = \{(\mathfrak{t}_1, \mathfrak{S}_1), \dots, (\mathfrak{t}_m, \mathfrak{S}_m)\}$ the maximal arity of an isomorphism type \mathfrak{t}_i is k and ℓ is the maximal size of the universe of a structure \mathfrak{S}_i . The interpretation Ψ intuitively creates for each set of elements of type \mathfrak{t}_i a copy of the structure \mathfrak{S}_i , so, defines a universe of elements of the form (A, i) , where $|A| \leq k$ and $i \leq \ell$. Such elements can be encoded by tuples of length $d \stackrel{\text{def}}{=} k + \ell + 1$. Quantifier-free formulas can determine the isomorphism type of a set of elements and, by the properties of a cookbook reduction, whether a tuple $((A, i_1), \dots, (A, i_r))$ exists in the interpreted structure only depends on the isomorphism type of A . ◀

We call a first-order interpretation *set-respecting* if, for the equivalence relation defined by the formula $\varphi_{\sim}(\bar{x}_1, \bar{x}_2)$, two tuples \bar{a}_1, \bar{a}_2 are only in the same equivalence class if \bar{a}_1 and \bar{a}_2 contain the same set of elements.

► **Theorem 2.** *For every set-respecting quantifier-free first-order interpretation Ψ there is a cookbook reduction ρ such that ρ and Ψ are equivalent for every structure with a linearly ordered universe.*

Proof idea. Let d be the dimension of Ψ . For every isomorphism type \mathfrak{t} of $k \leq d$ elements, the number ℓ of elements $([k], 1), \dots, ([k], \ell)$ in the universe of $\mathfrak{S}(\mathfrak{t})$, so, the number of elements added to the target structure because of a set of elements with isomorphism type \mathfrak{t} , is equal to the number of equivalence classes of the congruence defined by φ_{\sim} on the set of d -tuples that contain exactly the k elements of \mathfrak{t} and satisfy the formula φ_U of Ψ . We identify each of the ℓ elements with a particular d -tuple over the set $[k]$, which is possible as $[k]$ is linearly ordered. The structure $\mathfrak{S}(\mathfrak{t})$ is then defined as dictated by Ψ . ◀

As SAT is NP-hard under set-respecting quantifier-free FO interpretations [4], we obtain:

► **Corollary 3.** *Assuming that input structures are linearly ordered, SAT is NP-hard under cookbook reductions.*

Note that in descriptive complexity theory one often studies relational input structures that are not linearly ordered (although Immerman usually assumes a linear order to be present [7, Proviso 1.14]). However, when considering Turing machines as models of computation in complexity theory, inputs are binary string encodings and therefore linearly ordered.

5 Towards automated correctness tests and feedback

We now turn to the problem of checking whether a given reduction candidate is a valid reduction between two computational problems P and P^* . In a first variation of this problem, a corresponding algorithm gets as input the reduction candidate $\rho \in \mathcal{R}$ as well as the two problems $P \in \mathcal{C}$ and $P^* \in \mathcal{C}^*$, for a fixed class \mathcal{R} of reductions and fixed complexity classes \mathcal{C} and \mathcal{C}^* . Formally, this corresponds to solving the following algorithmic problem $\text{REDUCTION?}(\mathcal{C}, \mathcal{C}^*, \mathcal{R})$, parameterized by \mathcal{C} , \mathcal{C}^* , and \mathcal{R} . Also fixing the problems P and P^* yields the special case $\text{REDUCTION?}(P, P^*, \mathcal{R})$.

Problem: $\text{REDUCTION?}(\mathcal{C}, \mathcal{C}^*, \mathcal{R})$

Input: Algorithmic problems $P \in \mathcal{C}$,
 $P^* \in \mathcal{C}^*$, and a reduction $\rho \in \mathcal{R}$.

Question: Is ρ a reduction from P to P^* ?

Problem: $\text{REDUCTION?}(P, P^*, \mathcal{R})$

Input: A reduction $\rho \in \mathcal{R}$.

Question: Is ρ a reduction from P to P^* ?

We are slightly vague here, as for the moment we leave open how algorithmic problems and reductions are represented. It will be clear how these are represented for all classes \mathcal{C} , \mathcal{C}^* and \mathcal{R} we will consider. For standard classes of reductions, – including reductions computable in polynomial time or logarithmic space, as well as first-order definable reductions – already the second, more restricted problem is clearly undecidable for all non-trivial P and P^* . Already testing whether a quantifier-free interpretation or even an edge gadget reduction reduces from some problem P to another problem P^* is undecidable, for simple P and P^* . As soon as P or P^* are part of the input, the REDUCTION? problem is undecidable in most cases in which one of the classes \mathcal{C} or \mathcal{C}^* is defined by an undecidable fragment of second-order logic, even for very simple classes of reductions.

► Theorem 4.

1. $\text{REDUCTION?}(P, P^*, \mathcal{R})$ is undecidable for the following parameters:
 - a. The class \mathcal{R} of first-order interpretations, $P = \emptyset$ and arbitrary P^* (or vice versa, i.e. arbitrary P and $P^* = \emptyset$).
 - b. The class \mathcal{R} of edge gadget reductions, $P = \emptyset$ and some graph problem P^* definable in first-order logic with arithmetic.
 - c. The class \mathcal{R} of quantifier-free interpretations, $P = \emptyset$ and the graph problem P^* defined by the first-order formula $\varphi^* \stackrel{\text{def}}{=} \forall x \exists y E(x, y)$.
2. $\text{REDUCTION?}(\mathcal{C}, \mathcal{C}^*, \mathcal{R})$ is undecidable for the following parameters:
 - a. A class \mathcal{R} containing the identity mapping, a class \mathcal{C} containing the empty problem, and a class \mathcal{C}^* defined by a fragment of second-order logic with undecidable finite satisfiability problem.
 - b. A class \mathcal{R} containing the identity mapping, a class \mathcal{C} defined by a fragment of second-order logic with undecidable finite satisfiability problem, and a class \mathcal{C}^* containing the empty problem.

In the rest of this section, we explore how to overcome the undecidability barriers. That is, we explore for which parameters one can obtain algorithms for solving $\text{REDUCTION?}(P, P^*, \mathcal{R})$ and $\text{REDUCTION?}(\mathcal{C}, \mathcal{C}^*, \mathcal{R})$. Our focus is on (restrictions of) cookbook reductions.

We start by exhibiting toy examples for algorithms for $\text{REDUCTION?}(P, P^*, \mathcal{R})$ for concrete algorithmic problems P and P^* in Section 5.1. For these examples, counterexamples can be provided if the input is not a correct reduction. A generalized view is taken in Section 5.2, where we exhibit algorithm templates for $\text{REDUCTION?}(P, P^*, \mathcal{R})$ for algorithmic problems P and P^* selected from classes of problems. Then, in Section 5.3, we consider algorithmic problems as part of the input by studying $\text{REDUCTION?}(\mathcal{C}, \mathcal{C}^*, \mathcal{R})$.

5.1 Warm-up: Reductions between explicit algorithmic problems

In this section we provide toy examples of how $\text{REDUCTION?}(P, P^*, \mathcal{R})$ can be decided for very restricted classes \mathcal{R} : (1) for reducing k -CLIQUE to ℓ -CLIQUE via global gadgets, for $k < \ell$, (2) for reducing k -VERTEXCOVER to k -FEEDBACKVERTEXSET via edge gadgets, and (3) for reducing HAMCYCLE_d to HAMCYCLE_u via restricted node gadgets. In all cases, the decision procedures are obtained by characterizing the class of correct gadgets.

While not deep, these characterizations and the algorithms resulting from them are a first step towards more general results.

We start by characterizing those global gadgets that reduce k -CLIQUE to ℓ -CLIQUE. For simplicity, we represent global gadget reductions ρ by a global gadget \mathfrak{g}_ρ and a distinguished subset A of its nodes. When applying ρ to a graph $G = (V, E)$, the gadget \mathfrak{g}_ρ is disjointly added to G and edges (u, v) are introduced for all $u \in A$ and all $v \in V$.

► **Proposition 5.** *Let ρ be a global gadget reduction with global gadget \mathfrak{g}_ρ and a distinguished subset A of its nodes. Let $k, \ell \in \mathbb{N}$ with $k < \ell$. Then the following are equivalent:*

1. ρ is a reduction from k -CLIQUE to ℓ -CLIQUE
2. \mathfrak{g}_ρ and A satisfy the following conditions:
 - a. \mathfrak{g}_ρ has no ℓ -clique
 - b. \mathfrak{g}_ρ has an $(\ell - k)$ -clique contained in A
 - c. \mathfrak{g}_ρ has no $(\ell - k + 1)$ -clique contained in A



Furthermore, if ρ is not a reduction from k -CLIQUE to ℓ -CLIQUE, then a counterexample can be computed efficiently.

We next characterize those edge gadgets that constitute a reduction from k -VERTEXCOVER to k -FEEDBACKVERTEXSET. We represent edge gadget reductions ρ by an edge gadget \mathfrak{g}_ρ with two distinguished nodes c and d . When applying ρ to a graph $G = (V, E)$, all edges $(u, v) \in E$ are replaced by disjoint copies of \mathfrak{g}_ρ , where u, v are identified with c, d , respectively.

► **Proposition 6.** *Let ρ be an edge gadget reduction based on the edge gadget \mathfrak{g}_ρ with distinguished nodes c and d . Then the following are equivalent:*

1. ρ is a reduction from k -VERTEXCOVER to k -FEEDBACKVERTEXSET
2. \mathfrak{g}_ρ satisfies the following conditions:
 - a. $\{c\}$ and $\{d\}$ are feedback vertex sets of \mathfrak{g}_ρ
 - b. \emptyset is not a feedback vertex set of \mathfrak{g}_ρ .

Furthermore, if ρ is not a reduction from k -VERTEXCOVER to k -FEEDBACKVERTEXSET, then a counterexample can be computed efficiently.

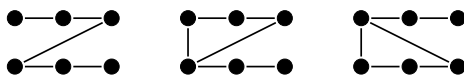
Lastly, we characterize restricted node gadget reductions from the directed Hamiltonian cycle problem HAMCYCLE_d to the undirected variant HAMCYCLE_u . For simplicity, we represent node gadget reductions ρ by node gadgets \mathfrak{g}_ρ . A node gadget \mathfrak{g}_ρ consists of two copies of a *node graph* $\mathfrak{S}(\mathfrak{t}_\bullet)$ and a set of additional edges between these copies. As an example, the standard reduction from HAMCYCLE_d to HAMCYCLE_u is represented by the node gadget  consisting of two copies of the node graph  with one additional edge

between them (cf. Figures 2(c) and 3(b)). When applying \mathfrak{g}_ρ to a graph $G = (V, E)$, all nodes in V are replaced by a copy of the node graph and two such copies for nodes u, v are connected accordingly by the additional set of edges, if $(u, v) \in E$.

As a first step towards characterizing node gadget reductions between HAMCYCLE_d and HAMCYCLE_u , we characterize all correct node gadget reductions whose node graph has at most three nodes.

► **Proposition 7.** *Let ρ be a node gadget reduction with node gadget \mathfrak{g}_ρ whose node graph has at most three nodes. Then the following are equivalent:*

1. ρ is a reduction from HAMCYCLE_d to HAMCYCLE_u
2. \mathfrak{g}_ρ is either of the following node gadgets (with the two copies of the node graphs depicted at top and bottom), up to symmetries:



Furthermore, if ρ is not a reduction from HAMCYCLE_d to HAMCYCLE_u , a counterexample can be computed efficiently.

5.2 Decidable cases for classes of (fixed) algorithmic problems

In this section, we study the question whether there are classes \mathcal{C} and \mathcal{C}^* of algorithmic problems as well as classes \mathcal{R} of reductions, such that after fixing $P \in \mathcal{C}$ and $P^* \in \mathcal{C}^*$ there is an algorithm that tests correctness of inputs $\rho \in \mathcal{R}$.

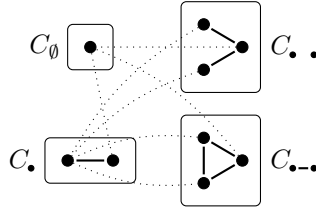
We first give an example that decidability results are possible for non-trivial classes of reductions and problems. Afterwards, we sketch how the technique employed in the proof can be generalized.

► **Theorem 8.** *$\text{REDUCTION?}(P, P^*, \mathcal{R})$ is decidable for the class \mathcal{R} of cookbook reductions with arity bounded by some $r > 0$, arbitrary P , and P^* definable in first-order logic.*

The proof idea is to represent cookbook reductions ρ by “recipe structures” $\text{recipe}(\rho)$ such that $\rho(\mathcal{A})$ can be constructed from the disjoint union $\mathcal{A} \uplus \text{recipe}(\rho)$ of \mathcal{A} and $\text{recipe}(\rho)$ via an FO-interpretation which depends on the arity and schema of ρ , but is independent of ρ itself. Then we prove that correctness of reductions in the setting of Theorem 8 only depends on the FO-similarity type of their recipe.

Intuitively, the recipe of a cookbook reduction ρ is the disjoint union of the structures $\mathfrak{S}(\mathfrak{t})$ for all relevant isomorphism types \mathfrak{t} , where additional unary relations indicate the source structure and an additional binary relation identifies inherited elements (those (A, j) where A is a strict subset of the domain of \mathfrak{t}) with their origin. Formally, fix two schemas σ and σ^* , an arity $r \in \mathbb{N}$, and define $\mathfrak{T}_{\leq r}$ to be the finite set of all isomorphism types \mathfrak{t} over the schema σ of arity at most r . The *recipe* $\text{recipe}(\rho)$ of a cookbook reduction ρ of arity at most r from σ to σ^* is a structure over the schema $\sigma^* \cup \{\approx\} \cup \{C_{\mathfrak{t}} \mid \mathfrak{t} \in \mathfrak{T}_{\leq r}\}$, where \approx is binary and all $C_{\mathfrak{t}}$ are unary. The restriction of $\text{recipe}(\rho)$ to the schema $\sigma^* \cup \{C_{\mathfrak{t}} \mid \mathfrak{t} \in \mathfrak{T}_{\leq r}\}$ is the disjoint union $\biguplus_{\mathfrak{t} \in \mathfrak{T}_{\leq r}} \mathfrak{S}(\mathfrak{t})$, where we set $\mathfrak{S}(\mathfrak{t}) = \rho(\mathfrak{t})$ if \mathfrak{t} is not in the support of ρ , and each $C_{\mathfrak{t}}$ is interpreted as the universe of $\mathfrak{S}(\mathfrak{t})$. The relation \approx “identifies” inherited elements and their original version: for every $\mathfrak{t}, \mathfrak{t}' \in \mathfrak{T}_{\leq r}$ such that \mathfrak{t} is the type of a strict subset of the elements of \mathfrak{t}' , if a' is an element of $\mathfrak{S}(\mathfrak{t}')$ inherited from $\mathfrak{S}(\mathfrak{t})$'s element a , then $a' \approx a$ holds in $\text{recipe}(\rho)$.

The structure $\text{recipe}(\rho)$ representing the cookbook reduction ρ from 3-CLIQUE to 4-CLIQUE given in Figure 3 can be found in Figure 4.



■ **Figure 4** The recipe $\text{recipe}(\rho)$ for the cookbook reduction of arity 2 from 3-CLIQUE to 4-CLIQUE from Figure 3. There are four unary relations for the types \mathfrak{t}_\emptyset , \mathfrak{t}_\bullet , $\mathfrak{t}_{\bullet\bullet}$, and $\mathfrak{t}_{\bullet\bullet\bullet}$ of loopless undirected graphs. The dotted edges represent the binary inheritance relation \approx .

There is an FO-interpretation that applies a recipe $\text{recipe}(\rho)$ to a structure \mathcal{A} by interpreting $\mathcal{A} \uplus \text{recipe}(\rho)$.

► **Lemma 9.** *Fix $r > 0$ and two schemas σ, σ^* . There is an FO-interpretation $\mathcal{I}_{\sigma, \sigma^*}^r$ such that $\rho(\mathcal{A})$ and $\mathcal{I}_{\sigma, \sigma^*}^r(\mathcal{A} \uplus \text{recipe}(\rho))$ are isomorphic, for every cookbook reduction ρ from σ to σ^* of arity at most r and for every σ -structure \mathcal{A} .*

As FO-interpretations preserve FO-similarity, there is a function $f_{\sigma, \sigma^*}^r : \mathbb{N} \rightarrow \mathbb{N}$ such that for every $k \in \mathbb{N}$, $\mathcal{A} \equiv_{f_{\sigma, \sigma^*}^r(k)}^{\text{FO}} \mathcal{A}'$ entails $\mathcal{I}_{\sigma, \sigma^*}^r(\mathcal{A}) \equiv_k^{\text{FO}} \mathcal{I}_{\sigma, \sigma^*}^r(\mathcal{A}')$ (see, e.g., [8, Section 3.2]).

We now prove Theorem 8.

Proof of Theorem 8. We show that whether a cookbook reduction ρ is a reduction from P to P^* solely depends on the FO_m -type of $\text{recipe}(\rho)$, for some large enough m that depends only on r , P , and P^* . As there are only finitely many such FO_m -types and because the type of $\text{recipe}(\rho)$ can be determined, the statement follows.

Let k be the quantifier rank of a formula $\varphi^* \in \text{FO}$ defining P^* . If the recipes of two reductions ρ and ρ' of arity at most r are f_{σ, σ^*}^r -similar, then so are $\mathcal{A} \uplus \text{recipe}(\rho)$ and $\mathcal{A} \uplus \text{recipe}(\rho')$ for all σ -structures \mathcal{A} (due to a simple Ehrenfeucht–Fraïssé argument). But then $\mathcal{I}_{\sigma, \sigma^*}^r(\mathcal{A} \uplus \text{recipe}(\rho))$ and $\mathcal{I}_{\sigma, \sigma^*}^r(\mathcal{A} \uplus \text{recipe}(\rho'))$ – and therefore also $\rho(\mathcal{A})$ and $\rho'(\mathcal{A})$ –, are k -similar. In particular, the reductions ρ and ρ' behave in the same way for all σ -structures \mathcal{A} , that is $\rho(\mathcal{A}) \models \varphi^*$ if and only if $\rho'(\mathcal{A}) \models \varphi^*$.

We conclude that whether $\rho(\mathcal{A})$ satisfies φ^* only depends on the $\text{FO}_{f_{\sigma, \sigma^*}^r(k)}$ -type of $\text{recipe}(\rho)$ for all \mathcal{A} . Hence, the recipe of positive instances of $\text{REDUCTION?}(P, P^*, \mathcal{R})$ is a union of equivalence classes for $\equiv_{f_{\sigma, \sigma^*}^r(k)}^{\text{FO}}$. For a reduction ρ it can now be evaluated whether its recipe satisfies the type of one of these equivalence classes. ◀

In the rest of this section, we explore how the technique used in the proof above can be generalized to logics beyond FO. Our focus is on monadic-second order logic (MSO), which extends FO by quantifiers for sets of elements. One of the key ingredients, that FO-interpretations preserve FO-similarity, does not translate to MSO for interpretations of dimension greater than one (not even for quantifier-free interpretations). Yet, decidability is retained for problems $P^* \in \text{MSO}$ if we restrict ourselves to *edge gadget reductions* (on graphs), instead of general cookbook reductions. This generalizes Proposition 6.

► **Theorem 10.** *$\text{REDUCTION?}(P, P^*, \mathcal{R})$ is decidable for the class \mathcal{R} of edge gadget reductions, arbitrary P , and P^* definable in monadic second-order logic.*

The proof exploits compositionality of MSO and can be generalized to other subclasses of cookbook reductions. A discussion of such subclasses is postponed to the long version of this paper.

Proof sketch. An edge gadget reduction ρ is specified as a graph \mathfrak{g}_ρ , with two distinguished nodes. As in the proof of Theorem 8, the idea is to show that there is an integer m such that whether ρ is a reduction from P to P^* only depends on the MSO_m -type of \mathfrak{g}_ρ . More precisely, for all gadget graphs \mathfrak{g}_ρ and $\mathfrak{g}_{\rho'}$ with $\mathfrak{g}_\rho \equiv_m^{\text{MSO}} \mathfrak{g}_{\rho'}$, one proves that $\rho(G) \equiv_k^{\text{MSO}} \rho'(G)$ for all graphs G , where k is the quantifier rank of an MSO-sentence describing P^* .

For proving MSO_k -similarity of $\rho(G)$ and $\rho'(G)$, one can use Ehrenfeucht-Fraïssé games for MSO (see, e.g., [10, Section 7.2]). The graphs $\rho(G)$ and $\rho'(G)$ are a composition of G with the edge gadgets \mathfrak{g}_ρ and $\mathfrak{g}_{\rho'}$, respectively. Duplicator has a winning strategy for the MSO-game played on (G, G) as well as for the MSO-game played on $(\mathfrak{g}_\rho, \mathfrak{g}_{\rho'})$. Her strategy for the game on $\rho(G)$ and $\rho'(G)$ is to combine these two winning strategies. For instance, if Spoiler moves on $\rho(G)$ and part of his move is on the edge gadget inserted for an edge (u, v) of G , then Duplicator's response for this part of the move is derived from her strategy for the game on $(\mathfrak{g}_\rho, \mathfrak{g}_{\rho'})$. The partial answers for individual edges are then combined. ◀

For both FO and MSO, the proof uses that the respective classes of reductions can be finitely partitioned into similarity classes and that all reductions in one class are either correct or not correct. This provides a basis for characterizations akin to the ones in Section 5.1 for concrete, arbitrary problems P and concrete P^* definable in FO or MSO.

5.3 Algorithmic problems as input: decidable cases

We now explore decidability when source and/or target problems are part of the input. We consider classes \mathcal{C} and \mathcal{C}^* captured by logics \mathcal{L} and \mathcal{L}^* , respectively, and write, e.g., $\text{REDUCTION}^?(\mathcal{L}, \mathcal{L}^*, \mathcal{R})$ for the algorithmic problem where we ask, given $\varphi \in \mathcal{L}$, $\varphi^* \in \mathcal{L}^*$ and $\rho \in \mathcal{R}$, whether ρ is a reduction from the problem defined by φ to the one defined by φ^* .

One approach for obtaining decidability for the problem $\text{REDUCTION}^?(\mathcal{L}, \mathcal{L}^*, \mathcal{R})$ is by restating it as a satisfiability question for a decidable logic. For a quantifier-free interpretation \mathcal{I} from σ -structures to σ^* -structures, denote by $\mathcal{I}^{-1}(\varphi^*)$ the σ -formula obtained from a σ^* -formula φ^* by replacing atoms in φ^* according to their definition in \mathcal{I} . Whether a quantifier-free interpretation \mathcal{I} is a reduction from the algorithmic problem defined by $\varphi \in \mathcal{L}$ to the one defined by $\varphi^* \in \mathcal{L}^*$ is equivalent to whether $\mathcal{A} \models \varphi$ if and only if $\mathcal{I}(\mathcal{A}) \models \varphi^*$, for all structures \mathcal{A} . This in turn is equivalent to checking whether $\varphi \leftrightarrow \mathcal{I}^{-1}(\varphi^*)$ is a tautology.

These observations yield, for instance, the following decidable variants, some involving the class QF of quantifier-free first-order interpretations, a class that includes all cookbook reductions, see Theorem 1. See the full version for the proof.

► **Theorem 11.**

1. $\text{REDUCTION}^?(\exists^*\text{FO}, \exists^*\text{FO}, \text{QF})$ is decidable.
2. $\text{REDUCTION}^?(P, \exists^*\text{FO}, \text{QF})$ is decidable for every fixed algorithmic problem P .
3. $\text{REDUCTION}^?(\exists^*\text{FO}, P^*, \mathcal{R})$ is decidable for every fixed algorithmic problem P^* definable in MSO and the class \mathcal{R} of edge gadget reductions.

6 Summary and discussion

We studied variants of the algorithmic problem $\text{REDUCTION}^?$ which asks whether a given mapping is a computational reduction between two algorithmic problems. In addition to studying this problem for standard classes of reductions, we also proposed a graphical and compositional language for computational reductions, called cookbook reductions, and compared their expressive power to quantifier-free first-order interpretations. While $\text{REDUCTION}^?$ is undecidable in many restricted settings, we identified multiple decidable cases

involving (restricted) cookbook reductions and quantifier-free first-order interpretations. Due to its graphical and compositional nature, cookbook reductions are well-suited to be used in teaching support systems for learning tasks tackling the design of computational reductions.

A prototype³ of our formal framework has been integrated into the teaching support system *Iltis* [11]. Recently it has been used in introductory courses *Theoretical Computer Science* with > 300 students at Ruhr University Bochum and TU Dortmund in workflows covering (i) understanding computational problems, (ii) exploring reductions via examples, and (iii) designing reductions.

References

- 1 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223:363–398, 2018.
- 2 Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Lecture Notes in Logic. Cambridge University Press, 2021.
- 3 Michael S. Crouch, Neil Immerman, and J. Eliot B. Moss. Finding reductions automatically. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2010. doi:10.1007/978-3-642-15025-8_10.
- 4 Elias Dahlhaus. Reduction to NP-complete problems by interpretations. In Egon Börger, Gisbert Hasenjaeger, and Dieter Rödding, editors, *Logic and Machines: Decision Problems and Complexity, Proceedings of the Symposium “Rekursive Kombinatorik” held from May 23-28, 1983 at the Institut für Mathematische Logik und Grundlagenforschung der Universität Münster/Westfalen*, volume 171 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 1983. doi:10.1007/3-540-13331-3_51.
- 5 Victor Dalmau and Jakub Oprsal. Local consistency as a reduction between constraint satisfaction problems. *CoRR*, abs/2301.05084, 2023. doi:10.48550/arXiv.2301.05084.
- 6 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.
- 7 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987. doi:10.1137/0216051.
- 8 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 9 Charles Jordan and Lukasz Kaiser. Experiments with reduction finding. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2013. doi:10.1007/978-3-642-39071-5_15.
- 10 Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.
- 11 Marko Schmellenkamp, Fabian Vehlken, and Thomas Zeume. Teaching formal foundations of computer science with Iltis. *Educational Column of the Bulletin of EATCS*, 2024. URL: <http://bulletin.eatcs.org/index.php/beatcs/article/download/797/842>.

³ See <https://iltis.cs.tu-dortmund.de/computational-reductions>