Deciding Open Definability via Subisomorphisms

Carlos Areces[1,2], Miguel Campercholi[1,2], and Pablo Ventura[1,2]

[1] FaMAF, Universidad Nacional de Córdoba, Argentina
[2] CONICET, Argentina

Abstract. Given a logic $\mathcal{L}$, the $\mathcal{L}$-Definability Problem for finite structures takes as input a finite structure $\boldsymbol{A}$ and a target relation T over the domain of $\boldsymbol{A}$, and determines whether there is a formula of $\mathcal{L}$ whose interpretation in $\boldsymbol{A}$ coincides with T. In this note we present an algorithm that solves the definability problem for quantifier-free first order formulas. Our algorithm takes advantage of a semantic characterization of open definability via subisomorphisms, which is sound and complete. We also provide an empirical evaluation of its performance.

1 Introduction

Providing a logic $\mathcal{L}$ with a formal semantics usually starts with the definition of a function that, given a suitable structure $\boldsymbol{A}$ for $\mathcal{L}$ and a formula φ in $\mathcal{L}$, returns the *extension* of φ in $\boldsymbol{A}$. Often, this extension is a set of tuples built from elements in $\boldsymbol{A}$. Extensions, also called *definable sets,* are the elements that will be referred by the formulas of $\mathcal{L}$ in a given structure, and in that sense, define the expressivity of $\mathcal{L}$. Definable sets are one of the central objects studied by Model Theory, and it is usually an interesting question to investigate, given a logic $\mathcal{L}$, which are the definable sets of $\mathcal{L}$ over a given structure $\boldsymbol{A}$, or, more concretely, whether a particular set of tuples is a definable set of $\mathcal{L}$ over $\boldsymbol{A}$. This is what we call the *Definability Problem for $\mathcal{L}$ over $\boldsymbol{A}$*.

One of the goals of Computational Logic is to understand the computational complexity of different problems for different logics and define efficient algorithms to solve them. Among the most investigated inference problems is *Satisfiability*: given a formula φ from a given logic $\mathcal{L}$ decide whether there exists a structure that makes φ true. In recent years, and motivated by concrete applications, other reasoning problems have sparkled interest. A well known example is the *Model Checking Problem* used in software verification to check that a given property P (expressed as a formula in the verification language) holds in a given formal representation S of the system (see, e.g., [1,2]). From a more general perspective, model checking can be defined as follows: given a structure $\boldsymbol{A}$, and a formula φ decide which is the extension T of φ in $\boldsymbol{A}$. From that perspective, the definability problem can be understood as the *inverse* problem of model checking: given a structure $\boldsymbol{A}$ and a target set T it asks whether there is a formula φ whose extension is T. A further example of a reasoning task related to definability comes from computational linguistics. The *Generation of Referring Expressions* (GRE) problem can be defined as follows: given a context C and an target object t in C, generate a grammatically correct description (in some natural language) that represents t, differentiating it from other objects in C (see [3] for a survey). Most of the work in this area is focused on the content determination problem (i.e., finding the properties that single out the target object) and leaves the actual realization (i.e., expressing this content as a grammatically correct expression) to standard techniques. As it is discussed in [4,5] the content realization part of the GRE problem can be understood as the task that, given a structure $\boldsymbol{A}$ that represents the context C, and an object t in the domain of $\boldsymbol{A}$ returns a formula φ in a suitable logic $\boldsymbol{\mathcal{L}}$ whose extension in $\boldsymbol{A}$ coincides with t. Of course, this will be possible only if t is definable for $\boldsymbol{\mathcal{L}}$ over $\boldsymbol{A}$.

The complexity of the definability problem for a number of logics has already been investigated. Let **FO** be first-order logic with equality in a vocabulary without constant symbols. The computational complexity of **FO**-definability was already discussed in 1978 [6,7], when a semantic characterization of the problem, based on automorphisms, placed **FO**-definability within coNP. Much more recently, in [8], a polynomial-time algorithm for **FO**-definability was given, which uses calls to a graph-isomorphism subroutine as an oracle. As a consequence, **FO**-definability is shown to be inside GI (defined as the set of all languages that are polynomial-time Turing reducible to the graph isomorphism problem). The authors also show that the problem is GI-hard and, hence, GI-complete. Interestingly, Willard showed in [9], that the complexity of the definability problem for the fragment of **FO** restricted to conjunctive queries (**CQ**) –i.e., formulas of the form $\exists \bar{x} \bigwedge_i C_i$, where each conjunct C_i is atomic– was coNEXPTIME-complete. The complexity upper bound followed from a semantic characterization of **CQ**-definability in terms of polymorphisms given in [10], while the lower bound is proved by an encoding of a suitable tiling problem. The complexity of definability has been investigated also for some modal languages: [4] shows that for the basic modal logic K, the definability problem is in P; in [5] the result is extended to some fragments of K where the use of negation is limited.

In [11] we prove that the definability problem for open formulas is coNP-complete. We also show that if the size and the arity of the target relation are taken as parameters the definability problem for open formulas is coW[1]-complete for every vocabulary with at least one, at least binary, relation. Given its complexity, algorithms to decide open decidability should be designed with care. In this article we introduce an algorithm that takes advantage of a semantic characterization of open definability via subisomorphisms presented in [12], which is sound and complete. We present an implementation of this algorithm and provide a preliminary empirical evaluation of its performance.

2 Preliminaries

In this section we provide some basic definitions and fix notation. We assume basic knowledge of first-order logic. For a detailed account see, e.g., [13].

We focus on definability by open first-order formulas in a purely relational first-order vocabulary, i.e., without function or constant symbols. For a relation symbol R in a vocabulary τ, let $\mathrm{ar}(R)$ denote the arity of R. In what follows, all vocabularies are assumed to be finite and purely relational. We assume that the language contains variables from a countable, infinite set $\mathsf{VAR} = \{x_1, x_2, \dots, x_n, \dots\}$. Variables from VAR are the only *terms* in the language. *Atomic formulas* are either of the form $v_i = v_j$ or $R(\bar{v})$, where $v_i, v_j \in \mathsf{VAR}$, $\bar{v}$ is a sequence of variables in VAR of length k and R is a relation symbol of arity k. *Open formulas* are Boolean combinations of atomic formulas (or, equivalently, quantifier-free first order formulas). We shall often write just formula instead of open formula. We write $\varphi(v_1, \dots, v_k)$ for an open formula φ whose variables are included in $\{v_1, \dots, v_k\}$.

Let τ be a vocabulary. A *τ-structure* (or *model*) is a pair $\boldsymbol{A} = \langle A, \cdot^{\boldsymbol{A}} \rangle$ where A is a non-empty set (the *domain* or *universe*), and $\cdot^{\boldsymbol{A}}$ is an *interpretation function* that assigns to each k-ary relation symbol R in τ a subset $R^{\boldsymbol{A}}$ of A^k. If $\boldsymbol{A}$ is a structure we write A for its domain and $\cdot^{\boldsymbol{A}}$ for its interpretation function. Given a formula $\varphi(v_1, \dots, v_k)$, and a sequence of elements $\bar{a} = \langle a_1, \dots, a_k \rangle \in A^k$ we write $\boldsymbol{A} \models \varphi[\bar{a}]$ if φ is true in $\boldsymbol{A}$ under an assignment that maps v_i to a_i.

We say that a subset $T \subseteq A^k$ is *open-definable* in $\boldsymbol{A}$ if there is an open first-order formula $\varphi(x_1, \dots, x_k)$ in the vocabulary of $\boldsymbol{A}$ such that $T = \{\bar{a} \in A^k : \boldsymbol{A} \models \varphi[\bar{a}]\}$.

In this article we study the following computational decision problem:

OpenDef

Instance: A finite relational structure $\boldsymbol{A}$ and a relation T over the domain of $\boldsymbol{A}$.
Question: Is T open-definable in $\boldsymbol{A}$?

We shall refer to the relations $R^{\boldsymbol{A}}$ for $R \in \boldsymbol{\mathcal{L}}$ as *base relations* and to T as the *target relation.* Let $f : \operatorname{dom} f \subseteq A \to A$ be a function. Given $S \subseteq A^m$, we say that *f preserves S* if for all $\langle s_1, \dots, s_m \rangle \in S \cap (\operatorname{dom} f)^m$ we have $\langle f s_1, \dots, f s_m \rangle \in S$. The function f is a *subisomorphism* (subiso for short) of $\boldsymbol{A}$ provided that f is injective, and both f and f^{-1} preserve $R^{\boldsymbol{A}}$ for each $R \in \tau$. (Note that a subiso of $\boldsymbol{A}$ is exactly an isomorphism between two substructures of $\boldsymbol{A}$.) We denote the set of all subisomorphisms of $\boldsymbol{A}$ by $\operatorname{subIso} \boldsymbol{A}$.

The following characterization of open-definability is key to our study.

Theorem 1 ([12, Thm 3.1]). *Let $\boldsymbol{A}$ be a finite relational structure and $T \subseteq A^m$. The following are equivalent:*

1. *T is open-definable in $\boldsymbol{A}$.*
2. *T is preserved by all subisomorphisms γ of $\boldsymbol{A}$.*
3. *T is preserved by all subisomorphisms γ of $\boldsymbol{A}$ with $|\operatorname{dom} \gamma| \leq m$.*

Proof. The equivalence of (1) and (2) is proved in [12, Thm 3.1]. Certainly (2) implies (3), so we show that (3) implies (2). Let γ be a subiso of $\mathbf{A}$ and let $\langle a_1, \dots, a_m \rangle \in T \cap (\operatorname{dom} \gamma)^m$. Note that the restriction $\gamma|_{\{a_1, \dots, a_m\}}$ is a subiso of $\mathbf{A}$. Thus, $\gamma(\bar{a}) = \gamma|_{\{a_1, \dots, a_m\}}(\bar{a}) \in T$.

3 Computing OpenDef efficiently

In view of Theorem 1 a possible strategy to decide $\mathrm{OpenDef}(\boldsymbol{A}, T)$ is to check that every subiso of $\boldsymbol{A}$ preserves T. The following results show that in many cases it is not necessary to check every member of $\mathrm{subIso}\,\boldsymbol{A}$. Let $\mathbb{S}(\boldsymbol{A})$ denote the set of substructures of $\boldsymbol{A}$.

Theorem 2. *Let $\boldsymbol{A}$ be a finite relational structure. Suppose $\mathcal{S} \subseteq \mathbb{S}(\boldsymbol{A})$ and $\mathcal{F} \subseteq \mathrm{subIso}\,\boldsymbol{A}$ are such that:*

i) *no two members of $\mathcal{S}$ are isomorphic,*
ii) $\mathrm{aut}\,\boldsymbol{B} \subseteq \mathcal{F}$ *for all $\boldsymbol{B} \in \mathcal{S}$, and*
iii) *for every $\boldsymbol{C} \in \mathbb{S}(\boldsymbol{A}) \setminus \mathcal{S}$ there are $\boldsymbol{C}' \in \mathcal{S}$ and $\rho, \rho^{-1} \in \mathcal{F}$ such that $\rho : \boldsymbol{C} \to \boldsymbol{C}'$ is an isomorphism.*

Then every $\gamma \in \mathrm{subIso}\,\boldsymbol{A}$ is a composition of functions in $\mathcal{F}$.

Proof. Let $\gamma : \boldsymbol{C} \to \boldsymbol{C}'$ be an isomorphism with $\boldsymbol{C}, \boldsymbol{C}' \in \mathbb{S}(\boldsymbol{A})$. If both $\boldsymbol{C}$ and $\boldsymbol{C}'$ are in $\mathcal{S}$, then $\boldsymbol{C} = \boldsymbol{C}'$ and $\gamma \in \mathrm{aut}\,\boldsymbol{C} \subseteq \mathcal{F}$. Suppose next that $\boldsymbol{C} \notin \mathcal{S}$ and $\boldsymbol{C}' \in \mathcal{S}$. Then there are $\rho, \rho^{-1} \in \mathcal{F}$ such that $\rho : \boldsymbol{C} \to \boldsymbol{C}'$ is an isomorphism. Let $\alpha := \gamma\rho^{-1}$, and observe that $\alpha \in \mathrm{aut}\,\boldsymbol{C}' \subseteq \mathcal{F}$. So $\gamma = \alpha\rho$ is a composition of functions in $\mathcal{F}$. To conclude assume that neither $\boldsymbol{C}$ nor $\boldsymbol{C}'$ are in $\mathcal{S}$. There are $\boldsymbol{B} \in \mathcal{S}$ and $\rho, \rho^{-1}, \delta, \delta^{-1} \in \mathcal{F}$ such that $\rho : \boldsymbol{C} \to \boldsymbol{B}$ and $\delta : \boldsymbol{C}' \to \boldsymbol{B}$ are isomorphisms. Note that $\alpha := \delta\gamma\rho^{-1}$ is an automorphism of $\boldsymbol{B}$, and thus $\alpha \in \mathcal{F}$. So $\gamma = \delta^{-1}\alpha\rho$ is a composition of functions in $\mathcal{F}$.

Let T be a set of tuples, we define the *spectrum* of T as

$$\mathrm{spec}\,T := \{|\{a_1, \dots, a_n\}| : \langle a_1, \dots, a_n \rangle \in T\}.$$

Combining Theorems 1 and 2 we obtain the following characterization of open-definability which is essential to our proposed algorithm.

Corollary 1. *Let $\boldsymbol{A}$ be a finite relational structure and $T \subseteq A^n$. Suppose $\mathcal{S}$ and $\mathcal{F}$ are as in Theorem 2, and let $\mathcal{F}' := \{\gamma \in \mathcal{F} : |\mathrm{dom}\,\gamma| \in \mathrm{spec}(T)\}$. Then T is open-definable in $\boldsymbol{A}$ if and only if T is preserved by all subisos in $\mathcal{F}'$.*

Proof. If T is open-definable in $\boldsymbol{A}$ then it must be preserved by every subiso of $\boldsymbol{A}$; so the right-to-left direction is clear. Suppose T is preserved by all $\gamma \in \mathcal{F}'$. Fix $\delta \in \mathrm{subIso}\,\boldsymbol{A}$ and let $\langle a_1, \dots, a_n \rangle \in T$ such that $\{a_1, \dots, a_n\} \subseteq \mathrm{dom}\,\delta$. Note that $|\{a_1, \dots, a_n\}| \in \mathrm{spec}\,T$. Set $\delta' := \delta|_{\{a_1, \dots, a_n\}}$; by Theorem 2 we know that there are $\delta_1, \dots, \delta_m \in \mathcal{F}$ such that $\delta' = \delta_1 \circ \cdots \circ \delta_m$. Since each δ_j is a bijection we have $|\mathrm{dom}\,\delta_j| = |\mathrm{dom}\,\delta| \in \mathrm{spec}\,T$, and hence $\delta_j \in \mathcal{F}'$ for $j \in \{1, \dots, m\}$. So, as δ' is a composition of functions that preserve T, we have that δ' preserves T. Finally observe that $\langle \delta a_1, \dots, \delta a_n \rangle = \langle \delta' a_1, \dots, \delta' a_n \rangle \in T$.

Next, we show that it is possible to delete redundancies in the relations of the input structure without affecting definability.

Let $\bar{a} = \langle a_1, \dots, a_n \rangle$ be a tuple. We define the *pattern* of $\bar{a}$, denoted by $\operatorname{pattern} \bar{a}$, to be the partition of $\{1, \dots, n\}$ such that i, j are in the same block if and only if $a_i = a_j$. For example, $\operatorname{pattern} \langle a, a, b, c, b, c \rangle = \{\{1, 2\}, \{3, 5\}, \{4, 6\}\}$. We write $\lfloor \bar{a} \rfloor$ to denote the tuple obtained from $\bar{a}$ by deleting every entry equal to a prior entry. E.g., $\lfloor \langle a, a, b, c, b, c \rangle \rfloor = \langle a, b, c \rangle$. Given a set of tuples S and a pattern θ let $\lfloor S \rfloor := \{\lfloor \bar{s} \rfloor : \bar{s} \in S\}$ and $S_\theta := \{\bar{s} \in S : \operatorname{pattern} \bar{s} = \theta\}$. For a structure $\mathbf{A}$ and an integer $k \geq 1$

- let $\mathbf{A}_{\leq k}$ be the structure obtained from $\mathbf{A}$ by deleting every relation of arity greater than k, and
- let $\lfloor \mathbf{A} \rfloor$ be the structure obtained from $\mathbf{A}$ by replacing each relation R of $\mathbf{A}$ by the relations $\{\lfloor R_{\operatorname{pattern} \bar{a}} \rfloor : \bar{a} \in R\}$.

Lemma 1. *Let $\mathbf{A}$ be a structure $T \subseteq A^m$, and take $k := \max(\operatorname{spec}(T))$. Then, T is open-definable in $\mathbf{A}$ if and only if T is open-definable in $\lfloor \mathbf{A} \rfloor_{\leq k}$.*

Proof. First observe that if $\gamma : D \subseteq A \to A$ is injective, then for each relation R on A we have that: γ preserves R if and only if γ preserves $\lfloor R_{\operatorname{pattern} \bar{a}} \rfloor$ for every $\bar{a} \in R$. It follows that $\operatorname{subIso} \mathbf{A} = \operatorname{subIso} \lfloor \mathbf{A} \rfloor$. Hence, by Theorem 1, we have that T is open-definable in $\mathbf{A}$ iff T is open-definable in $\lfloor \mathbf{A} \rfloor$. Next, notice that Corollary 1 implies that T is open-definable in $\lfloor \mathbf{A} \rfloor$ iff T is preserved by every subisomorphism with domain of size at most $k = \max(\operatorname{spec}(T))$. Now observe that if S is a relation of $\lfloor \mathbf{A} \rfloor$ with arity greater than k and $D \subseteq A$ has size at most k, then the restriction of S to D is empty (because tuples in S have no repetitions). In consequence, the relations of $\lfloor \mathbf{A} \rfloor$ with arity at most k determine the subisos of $\lfloor \mathbf{A} \rfloor$ that have to preserve T for T to be open-definable.

4 Implementation

In this section we present an algorithm for deciding definability, a pseudo-code of the implementation is shown in Algorithm 1.

Algorithm 1 decides whether T is open-definable in model $\boldsymbol{A}$. We assume implemented the following functions:

modelThinning($\boldsymbol{A}$,k): Where $\boldsymbol{A}$ is a model and $k > 0$, returns $\lfloor \boldsymbol{A} \rfloor_{\leq k}$ as in Lemma 1.

computeSpectrum(T): Where T is a set of tuples, returns a reverse sorted list containing $\operatorname{spec} T$.

cardinalityCheck($\boldsymbol{A}$,$\boldsymbol{S}$): Where $\boldsymbol{A}$ and $\boldsymbol{S}$ are models, returns True if for each $R \in \boldsymbol{\mathcal{L}}$, $|R^{\boldsymbol{A}}| = |R^{\boldsymbol{S}}|$ (see Lemma 2 below).

submodels($\boldsymbol{A}$, n): Where $\boldsymbol{A}$ is a model and n is a natural number, generates all submodels of $\boldsymbol{A}$ with cardinality n.

To better describe the way our algorithm works let us define a tree $\mathrm{Tr}^{\boldsymbol{A}}$. The root of $\mathrm{Tr}^{\boldsymbol{A}}$ is $\boldsymbol{A}$. Given a node $\boldsymbol{B}$ of $\mathrm{Tr}^{\boldsymbol{A}}$, its children are the submodels of $\boldsymbol{B}$ with l elements, where l is the greatest number in $\operatorname{spec} T$ strictly less than $|B|$. So, if the numbers in $\mathit{spec}(T)$ are $l_1 > l_2 > \cdots > l_r$, the level j of $Tr^{\boldsymbol{A}}$ (for

Algorithm 1 Open Definability Algorithm

```
 1: function isOpenDef($\boldsymbol{A}$, $T$)
 2:     $spectrum$ = computeSpectrum($T$)
 3:     $\boldsymbol{A}$ = modelThinning($\boldsymbol{A}$, max($spectrum$))
        {* $spectrum$ is a reverse sorted list of natural numbers *}
 4:     global $\mathcal{S} = \emptyset$
 5:     if $spectrum$ = [ ] then
 6:         return True
 7:     for $\boldsymbol{B} \in$ submodels($\boldsymbol{A}$, head($spectrum$)) do
 8:         if not isOpenDefR($\boldsymbol{B}$, $T$, tail($spectrum$)) then
 9:             return False
10:     return True

11: function isOpenDefR($\boldsymbol{A}$, $T$, $spectrum$)
12:     for $\boldsymbol{S} \in \mathcal{S}$ do
13:         if cardinalityCheck($\boldsymbol{A}$,$\boldsymbol{S}$) then
14:             if there is an iso $\gamma : \boldsymbol{A} \to \boldsymbol{S}$ then
15:                 if $\gamma$ and $\gamma^{-1}$ preserve $T$ then
16:                     return True
17:                 else
18:                     return False
19:     for $\gamma \in$ aut($\boldsymbol{A}$) do
20:         if $\gamma$ does not preserve $T$ then
21:             return False
22:     $\mathcal{S} = \mathcal{S} \cup \{\boldsymbol{A}\}$
23:     if $spectrum$ = [ ] then
24:         return True
25:     for $\boldsymbol{B} \in$ submodels($\boldsymbol{A}$, head($spectrum$)) do
26:         if not isOpenDefR($\boldsymbol{B}$, tail($spectrum$)) then
27:             return False
28:     return True
```

$j \geq 1$) contains all submodels of $\boldsymbol{A}$ with l_j elements. The level 0 contains only $\boldsymbol{A}$. As Algorithm 1 runs it traverses $\mathrm{Tr}^{\boldsymbol{A}}$ depth-first. It starts from level 1, unless $|A| \in \operatorname{spec} T$ in which case it starts from level 0. For every node $\boldsymbol{B}$ it performs the following tasks:

- Check if there is an isomorphism $\gamma : \boldsymbol{B} \to \boldsymbol{S}$ for some $\boldsymbol{S} \in \mathcal{S}$;
    1. if there is no such isomorphism, inspect the automorphisms of $\boldsymbol{B}$. If any member of aut $\boldsymbol{B}$ fails to preserve T, return False. Otherwise, add $\boldsymbol{B}$ to $\mathcal{S}$ and proceed to process the children of $\boldsymbol{B}$.
    2. if there is such a γ, check if γ and γ^{-1} preserve T. If either preservation fails return False. Otherwise, move on to an unprocessed sibling of $\boldsymbol{B}$.

When the algorithm runs out of nodes to process it returns True. An example of an execution of the algorithm is shown in the Appendix.

In our implementation of Algorithm 1 isomorphisms are checked as constraint satisfaction problems which are solved using MINION (version 1.8) [14]. When checking for the existence of an isomorphism between structures $\boldsymbol{A}$ and $\boldsymbol{B}$ we first verify that their domains have the same size and that $|R^{\boldsymbol{A}}| = |R^{\boldsymbol{B}}|$ for each R in the vocabulary. After this, it is enough to verify if there is an injective homomorphism between the structures. This is done on Line 14.

Lemma 2. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be relational finite structures such that $|R^{\boldsymbol{A}}| = |R^{\boldsymbol{B}}|$. Then, any bijective R-homomorphism from $\boldsymbol{A}$ to $\boldsymbol{B}$ is an R-isomorphism.*

Proof. Suppose γ^{-1} is not a homomorphism. Then there is $\bar{b}$ such that $\bar{b} \in R^{\boldsymbol{B}}$ but $\gamma^{-1}(\bar{b}) \notin R^{\boldsymbol{A}}$, thus $|R^{\boldsymbol{A}}| \neq |R^{\boldsymbol{B}}|$.

Knowing that the constraint satisfaction problem is hard, it pays to try to limit the number of calls to MINION. In particular the cardinality tests described above are cheap ways of discovering non isomorphisms. Note that it is more likely for the tests to fail in large vocabularies. In this regard, observe that trading $\boldsymbol{A}$ for $\lfloor \boldsymbol{A} \rfloor$ (Lemma 1) increases the size of the underlying vocabulary without increasing the size of the domain.

5 Empirical tests

In this section we present a preliminary empirical evaluation of our implementation of Algorithm 1, written in Python 3 under GPL 2 license, source code available at https://github.com/pablogventura/OpenDef. All tests were performed using an Intel Xeon E5-2620v3 processor with 12 cores (however, our algorithm does not make use of parallelization) at 2.40GHz, 128 GiB DDR4 RAM 2133MHz. Memory was never an issue in the tests we ran. As there is no standard test set for definability problems, we tested on random instances. The main purpose of these experiments is to convey an idea of the times our tool requires to compute different kinds of instances, and how the parameters of the instances impact these running times. In view of the strategies our algorithm implements, we decided to consider the following parameters:

1. Size of the universe of the input model.
2. Number of relations in the input model.
3. Size of the relations in the input model (see the paragraph below).
4. Arity of the target relation.

Parameters (2) and (3) aims at testing our tool on models with different kinds of underlying "structure". When considering how the number of tuples in base relations affects the structure, it is easy to see that this depends on the size of the universe (adding an edge to a graph with 3 nodes has a much greater impact than adding and edge to a graph with 50 nodes). Thus, our measure for a base relation R of $\boldsymbol{A}$ is its *density*, defined as follows:

$$\text{density}(R) = \frac{|R|}{|A|^{\text{ar}(R)}}.$$

Since exchanging a base relation for its complement does not affect open-definability, it makes sense to consider only base relations with density at most 0.5.

For each test we compute median wall-time (in seconds), over 300 problem instances in each configuration.

5.1 On the number of isomorphism types of submodels

A key feature of our algorithm is that it takes advantage of models with a small number of isomorphism types among its substructures (because in that case the set $\mathcal{S}$ in Algorithm 1 remains small throughout the computation). One can think of a model with few isomorphism types among its substructures as having a *regular* or *symmetrical* structure. On the other hand, if there is a high diversity of isomorphism types of submodels the model has less regularities our strategy can exploit. Define the *k-diversity* of a structure $\boldsymbol{A}$ as the number of isomorphism types among its substructures of size k. At this point it is important to note that the k-diversity that is prone to have the greatest impact on the running times is for $k = \max(\operatorname{spec} T)$. This is due to the fact that submodels of size greater than k are not processed and the number of submodels of size less than k is substantially smaller (assuming $k \leq |A|/2$), and these are much easier to test for isomorphisms. It is worth pointing out here that when thinking of the k-diversity as great or small, this should be done by comparing it to the total number of subsets of size k of A.

To be able to evaluate the impact of the diversity in the running times we need to generate instances of varying diversities; this is accomplished by noting two things. First, models with sparse (low density) relations have low diversity since most submodels do not contain any edges at all. Second, a small number of base relations of small arity entails a low number of possible isomorphism types (e.g., there are only 10 directed graphs with two elements). Thus, by varying the density and number of base relations in our test configurations we are able to obtain a range of diversities. Since diversity grows rapidly as a function of the density of the base relations we use a logarithmic increments for densities.

In all but one of our test configurations (Figure 2) we use only binary base relations. This is due to a couple of reasons:

- All target relations have arity at most three and thus the function **modelThinning** will erase base relations of arity greater than 3.
- The point of varying base relations is to generate a range of diversities (as explained above) and this can be accomplished adding either ternary or binary base relations. However, using binary relations we have finer control over the increments in diversity.

5.2 Tests with definable target

When the target relation is definable, our algorithm has to find and check the greatest possible number of subisos of $\boldsymbol{A}$. Hence, to get a sense of the worst case times we ran most of the tests for the definable case. The instances are generated by first fixing the universe, number and density of base relations, and then each base relation is built by randomly picking its tuples. In all instances the target relation contains all tuples of the given length, thus being obviously definable.

In the tests shown in the first line of Figure 1 the target is binary. The size of the domains are 30, 40 and 50, and the number of base relations are 1, 5 and 10. All base relations have arity 2, and their densities go through $\frac{0.5}{2^4}$, $\frac{0.5}{2^3}$, $\frac{0.5}{2^2}$, $\frac{0.5}{2^1}$, $\frac{0.5}{2^0}$. For the tests displayed in the second line the arity of the target is 3.

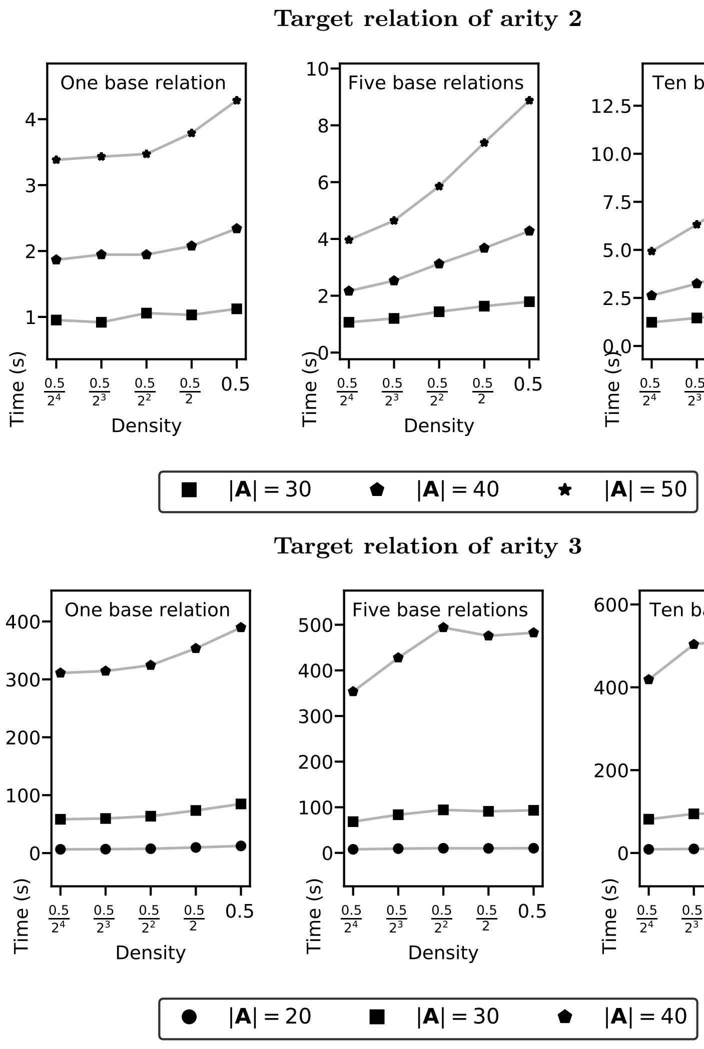


Fig. 1: Multiple base relations of arity 2 and target relation of arity 2 and 3.

The first thing that we notice is the considerable difference in times between the instances with binary and ternary targets. This is easily understood when considering that every one of the $\binom{|A|}{k}$ submodels of size $k = \max \operatorname{spec}(T)$ has to be processed, and in these examples k coincides with the arity of the target. Regarding the impact of universe sizes, we can clearly observe a polynomial behavior, quadratic for the binary target and cubic for the ternary target. Again, this is in direct correspondence to the number of submodels processed in each case. Finally, let us analyze the effect of increasing the number and density of the base relations. Note that these parameters do not affect the number of submodels to be processed but rather the amount of work processing each submodel takes (as discussed in section 5.1). Increasing the density and the number of base relations makes the problem harder, since both represent an increase in diversity. However, it is interesting to note that the overhead is rather small. For example, comparing the times in the second line of Figure 1 between instances with universe of 40 elements, we see in the leftmost graph that for one base relation of the lowest density the time is around 300 seconds, and, as shown in the rightmost graph, the time for the same universe size and ten base relations of arity 0.5 the time is roughly 550 seconds. A modest time increase considering the difference in the *size* of the compared instances (here size means number of bits encoding the instance). This behavior is explained by noting that, even though there is a large gap in the diversity of the two instances, most isomorphism tests done in the higher diversity case are rapidly answered by the function **cardinalityCheck**. Although many more isomorphism tests have to be carried out, the actual calls to the subroutine testing isomorphisms are avoided in the higher diversity case (recall that testing relational structures for isomorphism is GI-complete [15]). It should be noted that this behavior may not carry over to situations where the base relations are not randomly generated.

Effect of model thinning. **modelThinning** can have a great impact in running times. Figure 2 charts the times with and without applying the function **modelThinning** to the input model (line 3 in Algorithm 1). All instances have 20-element domains, and one ternary base relation whose density varies as in the above tests. The target is the set of all 3-tuples, and thus definable.

The time difference is explained by noting that **modelThinning** is likely to break the ternary base relation into one ternary and several binary and unary relations. Thus, the chances of the function **cardinalityCheck** to detect non-isomorphic submodels greatly increases, because for submodels $\boldsymbol{B}$ and $\boldsymbol{C}$ to be isomorphic we must have $\left|R^{\boldsymbol{B}}\right| = \left|R^{\boldsymbol{C}}\right|$ for *each* R in the vocabulary. Once again, this analysis may not apply to base relations that are not randomly generated.

5.3 Tests with non-definable target

The cases where the target relation T is *not* definable are potentially much easier for our algorithm, as it can stop computation the moment it finds a subisomorphism that does not preserve T. On the left of Figure 3 we can see the times of the non-definable case, where the input structures have the same parameters

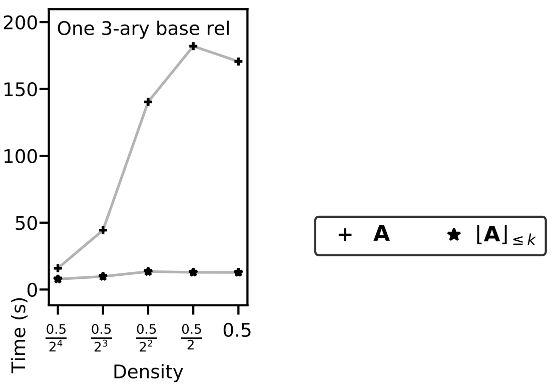


Fig. 2: $|A| = 20$, one base ternary relation and $T = A^3$.

as in the rightmost graph in the second line of Figure 1 (which we repeat on the right of Figure 3). To generate a non-definable targets for each instance, we randomly generated ternary relations of density 0.5 until a non-definable one was encountered.

6 Conclusions and future work

We designed and implemented an algorithm that decides the definability problem for open first-order formulas with equality over a purely relational vocabulary. The algorithm is based on the semantic characterization of open definability given in [12]. It is proved in [11] that this problem is coNP-complete and that the shortest defining formulas can grow exponentially for sequences of instances that grow polynomially in size. Thus, the direct approach to solve open definability by a search for a defining formula does not seem like an enticing alternative. (Also note that any kind of polynomially bound witness of definability would imply coNP $\subseteq$ NP.) Instead, our algorithm carries out a search over the submodels of size at most $k = \max(\text{spec}(T))$ of the input structure, and has to exhaust the search space to give a positive answer. Hence, the running time of our algorithm depends exponentially on the parameter k (for a detailed analysis of the parameterized complexity of OpenDef see [11]). We also gather from these facts that the refinement of the original semantic characterization of open definability [12] into Corollary 1 can have a meaningful impact for suitable instances. The empirical tests confirm the exponential dependency on k, and give us an insight into the impact of the other parameters considered. Notably, we learned from the tests that, for randomly generated input structures, varying the number and density of the base relations does not have a great impact in running times. As noted in the discussion in section 5.2, this is due to the fact that, as diversity grows and more isomorphism tests have to be calculated, the

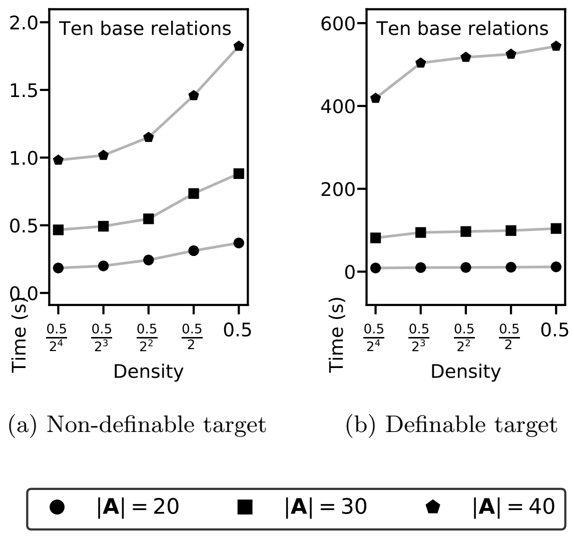


Fig. 3: Comparing definable and non-definable cases.

number of these tests that are easily decided by comparing cardinalities also grows. Another interesting conclusion we can draw from the experiments is that replacing an input structure $\boldsymbol{A}$ for the equivalent $\lfloor \boldsymbol{A} \rfloor_{\leq k}$ can entail significant improvements in computation times. Finally, the tests show that our strategy is better suited for negative instances of the problem (for obvious reasons).

There are many directions for further research. To start with, the empirical testing we presented is very preliminary, and further tests are necessary. There are no standard test sets for definability, and there are too many variables that can, a priori, impact in the empirical performance of any algorithm testing definability. We found particularly difficult to define challenging test sets that would let us explore the transition from definability to non-definability smoothly. It would also be interesting to attempt to extend our results (and our tool) to other fragments of first-order logic. We have already carried out preliminary work investigating the definability problem for open positive formulas. Considering signatures with functional symbols would be particularly interesting for applications of definability in universal algebra. It would also be natural to investigate definability over *finite classes of models* instead of over a single model. In the particular case of open definability the two problems coincide as it would be enough to consider the disjoint union of all the models in the class as input, but this is not the case for other logics. Finally, it would be useful to find classes of modes in which the problem is well conditioned, and where polynomial algorithms exist.

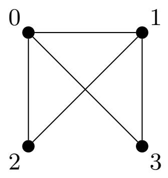
Acknowledgments. This work was partially supported by grant ANPCyT-PICT-2013-2011, STIC-AmSud “Foundations of Graph Structured Data (FoG)”, and the Laboratoire International Associé “INFINIS”. We thank also the reviewers for useful comments and minor corrections.

References

1. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
2. Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification: Model-Checking Techniques and Tools. Springer (2010)
3. Krahmer, E., van Deemter, K.: Computational generation of referring expressions: A survey. Computational Linguistics **38**(1) (2012) 173–218
4. Areces, C., Koller, A., Striegnitz, K.: Referring expressions as formulas of description logic. In: Proceedings of the Fifth International Natural Language Generation Conference (INLG’08), Association for Computational Linguistics (2008) 42–49
5. Areces, C., Figueira, S., Gorín, D.: Using logic in the generation of referring expressions. In Pogodalla, S., Prost, J., eds.: Proceedings of the 6th International Conference on Logical Aspects of Computational Linguistics (LACL 2011). Volume 6736 of Lecture Notes in Computer Science. Springer (2011) 17–32
6. Paredaens, J.: On the expressive power of the relational algebra. Information Processing Letters **7**(2) (1978) 107–111
7. Bancilhon, F.: On the completeness of query languages for relational data bases. In Winkowski, J., ed.: Proceedings of the 7th Symposium of Mathematical Foundations of Computer Science. Volume 64., Springer (1978) 112–123
8. Arenas, M., Diaz, G.: The exact complexity of the first-order logic definability problem. ACM Transactions on Database Systems **41**(2) (2016) 13:1–13:14
9. Willard, R.: Testing expressibility is hard. In: Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP10). (2010) 9–23
10. Jeavons, P., Cohen, D., Gyssens, M.: How to determine the expressive power of constraints. Constraints **4**(2) (1999) 113131
11. Areces, C., Campercholi, M., Penazzi, D., Ventura, P.: The complexity of definability by open first-order formulas. Logic Journal of the IGPL (2017) Submitted.
12. Campercholi, M., Vaggione, D.: Semantical conditions for the definability of functions and relations. Algebra universalis **76**(1) (Sep 2016) 71–98
13. Ebbinghaus, H., Flum, J., Thomas, W.: Mathematical Logic. Springer-Verlag (1994)
14. Gent, I., Jefferson, C., Miguel, I.: MINION: a fast, scalable, constraint solver. In: Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006), IOS Press (2006) 98–102
15. Zemlyachenko, V.N., Korneenko, N.M., Tyshkevich, R.I.: Graph isomorphism problem. Journal of Soviet Mathematics **29**(4) (May 1985) 1426–1481

A An execution of OpenDef

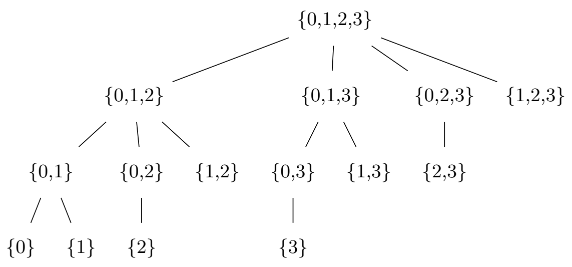
To illustrate how Algorithm 1 works, we run it for the following instance. The input structure is the graph $\boldsymbol{G} = (\{0, 1, 2, 3\}, E)$



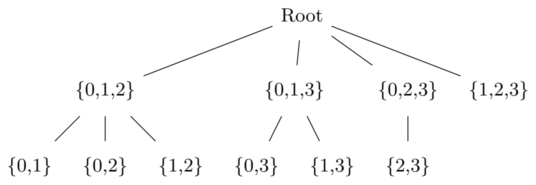
and the target relation is

$$T = \{(a, b, c, d) : E(b, c) \wedge (a = b \vee c = d)\}.$$

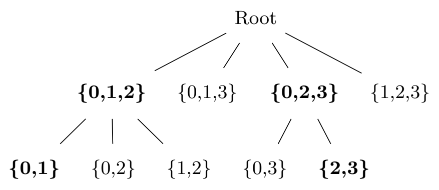
A brute force algorithm based on Theorem 1 would have to check that every sub-isomorphisms of $\boldsymbol{G}$ preserves T. That is, for each subuniverse S of $\boldsymbol{G}$ compute all sub-isomorphisms with domain S, and check them for preservation. This means going trough every subuniverse listed in the tree below, where the automorphisms of all subuniverses will be calculated and checked for preservation.



The first improvement of Algorithm 1 allows us to prune tree levels whose nodes do not have cardinality in spec(T)(this is correct due to Lemma 1). In the current example, it only processes subuniverses with size in spec(T) $= \{2, 3\}$. This amounts to process only the following subuniverses. (The node *Root* is not a subuniverse to be processed; we add it to better visualize the tree traversed by the DFS.)



The second improvement is due to the order in which Algorithm 1 computes the subisomorphisms with a given domain S. It first tries to find an isomorphism from S to an already processed subuniverse. In the case where such an isomorphism exists, no other isomorphisms from S are computed, and the subuniverses contained in S are pruned (this is correct by Corollary 1). The subuniverses of $\boldsymbol{G}$ actually processed by Algorithm 1 are shown below. The bold nodes are in $\mathcal{S}$, and therefore are the only ones that are checked for automorphisms and subuniverses.



In view of the strategies Algorithm 1 employs we get a good idea of what makes for a well conditioned instances $\boldsymbol{A}, T$:

- Target relations with a small spectrum compared to the power set of A.
- Structures with few isomorphisms types for substructures with cardinality in the spectrum.