Logical Characterizations of Heap Abstractions

G. Yorsh[1⋆], T. Reps[2], M. Sagiv[1], and R. Wilhelm[3]

[1] School of Comp. Sci., Tel-Aviv University; {gretay, msagiv}@post.tau.ac.il
[2] Comp. Sci. Dept., University of Wisconsin; reps@cs.wisc.edu
[3] Informatik, Univ. des Saarlandes; wilhelm@cs.uni-sb.de

Abstract. Shape analysis concerns the problem of determining "shape invariants" for programs that perform destructive updating on dynamically allocated storage. In recent work, we have shown how shape analysis can be performed using an abstract interpretation based on 3-valued first-order logic. In that work, concrete stores are finite 2-valued logical structures, and the sets of stores that can possibly arise during execution are represented (conservatively) using a certain family of finite 3-valued logical structures. In this paper, we show how 3-valued structures that arise in shape analysis can be characterized using formulas in first-order logic with transitive closure. We also define a non-standard ("supervaluational") semantics for 3-valued first-order logic that is more precise than a conventional 3-valued semantics, and demonstrate that the supervaluational semantics can be implemented using existing theorem provers.

1 Introduction

Abstraction and abstract interpretation [9] are key tools for automatically verifying properties of systems, both for hardware systems [7, 10] and software systems [44]. In abstract interpretation, sets of concrete stores are represented in a conservative manner by abstract values (as explained below). Each transition of the system is given an interpretation over abstract values that is conservative with respect to its interpretation over corresponding sets of concrete stores; that is, the result of "executing" a transition must be an abstract value that describes a superset of the concrete stores that actually arise. This methodology guarantees that the results of abstract interpretation overapproximate the sets of concrete stores that actually arise at each point in the system.

One issue that arises when abstraction is employed concerns the *expressiveness* of the abstraction method: "What collections of concrete states can be expressed exactly using the given abstraction method?" A second issue that arises when abstraction is employed is how to *extract information* from an abstract value. For instance, this is a fundamental problem for clients of abstract interpretation, such as verification tools, program optimizers, program-understanding tools, etc., which need to be able to interpret what an abstract value means. An abstract value a represents a set of concrete stores X; ideally, a query φ should return an answer that summarizes the result of posing φ against each concrete store $S \in X$:

– If φ is true for each S, the summary answer should be "true".
– If φ is false for each S, the summary answer should be "false".
– If φ is true for some $S \in X$ but false for some $S' \in X$, the summary answer should be "unknown".

This paper presents results on both of these questions, for a class of abstractions that originally arose in work on the problem of shape analysis [28, 5, 51]. Shape analysis

⋆ This research was supported by THE ISRAEL SCIENCE FOUNDATION (grant No 304/03).

concerns the problem of finding "shape descriptors" that characterize the shapes of the data structures that a program's pointer variables point to. Shape analysis is one of the most challenging problems in abstract interpretation because it generally deals with programs written in languages like C, C++, and Java, which allow (i) dynamic allocation and deallocation of cells from the heap, (ii) destructive updating of structure fields, and, in the case of Java, (iii) dynamic creation and destruction of threads. This combination of features creates considerable difficulties for any abstract-interpretation method.

The motivation for the present paper was to understand the expressiveness of the shape abstractions defined in [51]. In that work, concrete stores are finite 2-valued logical structures, and the sets of stores that can possibly arise during execution are represented (conservatively) using a certain family of finite 3-valued logical structures. In this setting, an abstract value is a set of 3-valued logical structures.

Because the notion of abstraction used in [51] is based on logical structures, our results are actually more broadly applicable than shape-analysis problems. For example, it was applied to verification of sorting algorithms [37]; showing absence of concurrent modification exception [47]; correct usage of JDBC, I/O streams, Java collections and iterators [59]; correctness of concurrent queue algorithms [60]; modelling concurrency in Java programs, which contain dynamic creation of objects and threads [58]; analyzing processes in ambient calculus [45]; and reducing space consumption of Java programs via compile-time memory management, with application to JavaCard programs [52].

In fact, our results apply to any abstraction in which concrete states of a system are represented by finite 2-value logical structure and abstraction is performed via the mechanisms described in Sections 2 and 3. The approach taken in the paper should also be relevant for addressing expressibility issues for a number of other abstractions that are related to [51], including [40, 31, 17, 24, 7, 6], as well as for the *allocation-site* abstraction—often used in points-to analysis [1, 54, 53, 15, 55, 11, 18]—in which all objects allocated at a single statement are represented by a single "abstract memory object" [29, 5]. Throughout the paper, however, we use shape-analysis examples to illustrate the concepts discussed.

The paper investigates the expressiveness of finite 3-valued structures by giving a logical characterization of these structures; that is, we examine the question

> For a given 3-valued structure S, under what circumstances is it possible to create a formula $\widehat{\gamma}(S)$, such that $S^\natural$ satisfies $\widehat{\gamma}(S)$ exactly when $S^\natural$ is a 2-valued structure that S represents? I.e., $S^\natural \models \widehat{\gamma}(S)$ iff S represents $S^\natural$.

This paper presents two results concerning this question:

– It is not possible to give a formula $\widehat{\gamma}(S)$ written in first-order logic with transitive closure for an arbitrary structure S (unless $NL = NP$, see Section 3). However, it is always possible for a well-defined class of 3-valued structures. (This class includes all the 3-valued structures that have been shown to be useful for shape analysis [51].)
– Moreover, it is always possible to give a $\widehat{\gamma}(S)$ in general, using a more powerful formalism, namely, monadic second-order formulas.

The ability to write a formula $\widehat{\gamma}(S)$ that exactly captures what S represents provides a fundamental tool for improving TVLA [38] by the use of symbolic methods. The current TVLA system performs iterative fixed-point computations and yields at every program point a set of 3-valued structures, which represent a superset of all possible stores that can arise at this point in any execution. However, TVLA suffers from two limitations: (i) it is not always as precise as possible (as explained below); (ii) it does not scale to handle large programs, because the worst-case complexity of the algorithm is doubly-exponential in certain parameters (typically, the number of program variables).

The contributions of this paper lay the required groundwork for using symbolic techniques to address both of these limitations. The ability to characterize a 3-valued structure S by a formula $\widehat{\gamma}(S)$ is a key step toward harnessing a standard (2-valued) theorem prover to aid in abstract interpretation:

– Computing the effect of a program statement on an abstract value in the most-precise way possible for a given shape-analysis abstraction.
– Developing a modular shape-analysis by using *assume-guarantee* reasoning. The idea is to allow arbitrary first-order formulas to be used to express pre- and post-conditions, thereby enabling the code of each procedure to be analyzed once for all potential contexts. This allows to use shape analysis for applications in which not all the source code is available. This becomes specifically profitable for recursive procedures since it saves the need to iterate shape analysis.

These methods are the subject of [62, 34].

Another contribution of this paper directly addresses the first of the aforementioned limitations of TVLA's current technique. We give a procedure for extracting information from a 3-valued logical structure S in the most-precise way possible. That is, we give a nonstandard way to check if a formula φ holds in S:

– If $\widehat{\gamma}(S) \Rightarrow \varphi$ is valid, i.e., holds in all 2-valued structures, we know that φ evaluates to 1 in all the 2-valued structures represented by S.
– If $\widehat{\gamma}(S) \Rightarrow \neg\varphi$ is valid, we know that φ evaluates to 0 in all the 2-valued structures represented by S.
– Otherwise we know that there exists a 2-valued structure represented by S where φ evaluates to 1, and there exists another 2-valued structure represented by S where φ evaluates to 0.

This method represents the most-precise way of extracting information from a 3-valued logical structure; in particular, whenever this method returns $1/2$ (standing for "unknown"), any sound method for extracting information from S must also return $1/2$. This is in contrast with the techniques used in [51], which can return $1/2$ even when all the 2-valued structures represented by S have the value 1 (or all have the value 0).

For practical purposes, the success of using symbolic methods depends on having a terminating theorem prover. Although the validity question is undecidable for first-order logic with transitive closure, several theorem provers for first-order logic have been created. In this paper, we report on two experiments in which we used these tools to implement symbolic procedures for extracting information from a 3-valued structure in the most-precise way possible. We also performed several successful experiments with other symbolic operations [62, 12]. Although these experiments are rather preliminary, we believe that this approach can be made to work in practice. For example, there has been some progress recently in using SPASS, including the use of transitive closure [36]. Also, in [26], we have identified a decidable subset of first-order logic with transitive closure that is useful for shape analysis. We define conditions under which $\widehat{\gamma}$ can be expressed in that logic (Section 5.2). We are also investigating other decidable logics, as well.

The remainder of the paper is organized as follows. Section 2 defines our terminology, and explains the use of 3-valued structures as abstractions of 2-valued structures. Section 3 presents the results on the expressiveness of 3-valued structures, and gives an algorithm for generating $\widehat{\gamma}$ for certain families of 3-valued structures. Section 4 discusses the problem of reading out information from a 3-valued structure in the most-precise way possible. Section 5 discusses the applications of $\widehat{\gamma}$ to program analysis and some implementation issues. Section 6 discusses related work. Appendix A defines an alternative abstract domain for shape analysis, based on canonical abstraction, and the $\widehat{\gamma}$ operation for that domain. Appendix B shows how to characterize general 3-valued structures. Appendix C contains the details for one of the paper's examples. The proofs appear in Appendix D.

2 Preliminaries

Section 2.1 defines the syntax and standard Tarskian semantics of first-order logic with transitive closure and equality. Section 2.2 introduces *integrity formulas*, which exclude structures that do not represent a potential store. Section 2.3 introduces 3-valued logical structures, which extend ordinary logical structures with an extra value, $1/2$, which represents "unknown" values that arise when several concrete nodes are represented by a single abstract node. The powerset of 3-valued structures forms an abstract domain, which is related to the concrete domain consisting of the powerset of 2-valued structures via *embedding*, as described in Section 2.4.

Fig. 1(a) shows the declaration of a linked-list data type in C, and Fig. 1(b) shows a C program that searches a list and splices a new element into the list. This program will be used as a running example throughout this paper.

```
/* list.h */
typedef struct node {
  struct node *n;
  int data;
} *List;
```

(a)

```
/* insert.c */
#include "list.h"
void insert(List x, int d) {
  List y, t, e;
  assert(acyclic_list(x) && x != NULL);
  y = x;
  while (y->n != NULL && ...)
    y = y->n;
  t = malloc();
  t->data = d;
  e = y->n;
  t->n = e;
  y->n = t;
}
```

(b)

Fig. 1. (a) Declaration of a linked-list data type in C. (b) A C function that searches a list pointed to by parameter x, and splices in a new element.

2.1 Syntax and Semantics of First-Order Formulas with Transitive Closure

We represent concrete stores by ordinary 2-valued logical structures over a fixed finite set of predicate symbols $\mathcal{P} = \{eq, p_1, \ldots, p_n\}$, where eq is a designated binary predicate, denoting equality of nodes. We also use $maxR$ to denote the maximal arity of the predicates in $\mathcal{P}$. Without loss of generality we exclude constant and function symbols from the logic.[4]

Example 21 *Table 1 lists the set of predicates used in the running example. The unary predicates x, y, t, and e correspond to the program variables* `x`, `y`, `t`, *and* `e`, *respectively. The binary predicate n corresponds to the* `n` *fields of* `List` *elements. The unary predicate is ("is shared") captures "heap sharing", i.e.,* `List` *elements pointed to by more than one field. (It was introduced in [5] to capture list and tree data structures.) The unary predicates r_x, r_y, r_t, and r_e hold for heap nodes reachable from the program variables* `x`, `y`, `t`, *and* `e`, *respectively. A heap node u is said to be* reachable *from a program variable if the variable points to a heap node u', and it is possible to go from u' to u by following zero or more* `n`*-links. Reachability is defined in term of the reflexive transitive closure of the predicate n.*

The notion of reachability plays a crucial role in defining abstractions that are useful for proving program properties in practice. For instance, it may have the effect of preventing disjoint lists from being collapsed in the abstract representation. This may significantly improve the precision of the answers obtained by a program analysis.

Predicate	**Intended Meaning**
$eq(v_1, v_2)$	Do v_1 and v_2 denote the same heap node?
$q(v)$	Does pointer variable `q` point to node v?
$n(v_1, v_2)$	Does the `n` field of v_1 point to v_2?
$is(v)$	Is v pointed to by more than one field ?
$r_q(v)$	Is the node v reachable from `q` ?

Table 1. The set of predicates for representing the stores manipulated by programs that use the `List` data-type from Fig. 1(a). q denotes an arbitrary predicate in the set $PVar$, which contains a predicate for each program variable of type `List`. In the case of `insert`, $PVar = \{\mathtt{x}, \mathtt{y}, \mathtt{t}, \mathtt{e}\}$.

We define first-order formulas inductively over the **vocabulary** $\mathcal{P}$ using the logical connectives $\vee$ and $\neg$, the quantifier $\exists$, and the operator 'TC' in the standard way:

$$\varphi ::= \mathbf{0} \mid \mathbf{1} \mid p(v_1, \ldots, v_k) \mid (\neg\varphi_1) \mid (\varphi_1 \vee \varphi_2) \mid (\exists v_1 : \varphi_1) \mid (\mathit{TC}\ v_1, v_2 : \varphi_1)(v_3, v_4)$$
$$where\ p \in \mathcal{P};\ v_i \text{ are variables; } \varphi, \varphi_i \text{ are formulas}$$

The set of free variables of a formula is defined as usual. A formula is **closed** when it has no free variables. The operator 'TC' denotes transitive closure. If φ_1 is a formula with free variables V, then $(\mathit{TC}\ v_1, v_2 : \varphi_1)(v_3, v_4)$ is a formula with free variables $(V - \{v_1, v_2\}) \cup \{v_3, v_4\}$.

We use several shorthand notations: $\varphi_1 \Rightarrow \varphi_2 \stackrel{\text{def}}{=} (\neg\varphi_1 \vee \varphi_2)$; $\varphi_1 \wedge \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \vee \neg\varphi_2)$; $\varphi_1 \Leftrightarrow \varphi_2 \stackrel{\text{def}}{=} (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$; and $\forall v : \varphi \stackrel{\text{def}}{=} \neg\exists v : \neg\varphi$. The

[4] Constant symbols can be encoded via unary predicates, and n-ary functions via $(n+1)$-ary predicates.

transitive closure of a binary predicate p is $p^+(v_3, v_4) \stackrel{\text{def}}{=} (\mathit{TC}\ v_1, v_2 : p(v_1, v_2))(v_3, v_4)$. The *reflexive* transitive closure of a binary predicate p is $p^*(v_3, v_4) \stackrel{\text{def}}{=} ((\mathit{TC}\ v_1, v_2 : p(v_1, v_2))(v_3, v_4)) \lor eq(v_3, v_4)$. The order of precedence among the connectives, from highest to lowest, is as follows: $\neg$, $\land$, $\lor$, 'TC', $\forall$, and $\exists$. We drop parentheses wherever possible, except for emphasis.

Definition 1. **2-valued Logical Structures** *Let $\mathcal{P}_i$ denote the set of predicate symbols with arity i. A* **logical structure over** *$\mathcal{P}$ is a pair $S = \langle U, \iota \rangle$ in which*

- *U is a (possibly infinite) set of nodes.*
- *ι is the interpretation of predicate symbols, i.e., for every predicate symbol $p \in \mathcal{P}_i$, $\iota(p)\colon U^i \to \{0, 1\}$ determines the tuples for which p holds. Also, $\iota(eq)$ is the interpretation of equality, i.e., $\iota(eq)(u_1, u_2) = 1$ iff $u_1 = u_2$.*

Below we define the standard Tarskian semantics for first-order logic.

Definition 2. **Semantics of First-Order Logical Formulas** *Consider a logical structure $S = \langle U, \iota \rangle$. An* **assignment** *$Z$ is a function that maps free variables to nodes (i.e., an assignment has the functionality $Z\colon \{v_1, v_2, \ldots\} \to U$). An assignment that is defined on all free variables of a formula φ is called* **complete** *for φ. In the sequel, we assume that every assignment Z that arises in connection with the discussion of some formula φ is complete for φ. We say that S and Z* **satisfy** *a formula φ (denoted by $S, Z \models \varphi$) when one of the following holds:*

- $\varphi \equiv \mathbf{1}$
- *$\varphi \equiv p(v_1, v_2, \ldots, v_i)$ and $\iota(p)(Z(v_1), Z(v_2), \ldots, Z(v_i)) = 1$.*
- *$\varphi \equiv \neg\varphi_0$ and $S, Z \models \varphi_0$ does not hold.*
- *$\varphi \equiv \varphi_1 \lor \varphi_2$, and either $S, Z \models \varphi_1$ or $S, Z \models \varphi_2$.*
- *$\varphi \equiv \exists v_1 : \varphi_1$ and there exists a node $u \in U$, $m \geq 2$, such that $S, Z[v_1 \mapsto u] \models \varphi_1$.*
- *$\varphi \equiv (\mathit{TC}\ v_1, v_2 : \varphi_1)(v_3, v_4)$ and there exists $u_1, u_2, \ldots, u_m \in U$, $m \geq 2$, such that $Z(v_3) = u_1$, $Z(v_4) = u_m$ and for all $1 \leq i < m$, $S, Z[v_1 \mapsto u_i, v_2 \mapsto u_{i+1}] \models \varphi_1$.*

For a closed formula φ, we will omit the assignment in the satisfaction relation, and merely write $S \models \varphi$.

2.2 Integrity Formula

Because not all logical structures represent stores, we use a designated closed formula F, called the *integrity formula*,[5] to exclude structures that are not of interest; in our application, such structures are ones that do not correspond to possible stores. This allows us to restrict the set of structures to the ones satisfying F.

Definition 3. *A structure S is* **admissible** *if $S \models F$.*

In the rest of the paper, we assume that we work with a fixed integrity formula F. All our notations are parameterized by $\mathcal{P}$ and F.

Example 22 *For the `List` data type, there are four conditions that define the admissible structures. At any time during execution,*

[5] In [51] these are called "hygiene conditions".

(a) *each program variable can point to at most one heap node.*
(b) *the* `n` *field of a heap node can point to at most one heap node.*
(c) *predicate is ("is shared") holds for exactly those nodes that have two or more predecessors.*
(d) *the reachability predicate for each variable* `q` *holds for exactly those nodes that are reachable from program variable* `q`.

The set $PVar$ contains a predicate for each program variable of type `List`*; in the case of* `insert`*, $PVar = \{\mathtt{x}, \mathtt{y}, \mathtt{t}, \mathtt{e}\}$. Thus, the integrity formula F_{List} for the* `List` *data-type is:*

$$\begin{array}{lrr} & \wedge_{p \in PVar} \forall v_1, v_2 : p(v_1) \wedge p(v_2) \Rightarrow eq(v_1, v_2) & (a) \\ \wedge & \forall v, v_1, v_2 : n(v, v_1) \wedge n(v, v_2) \Rightarrow eq(v_1, v_2) & (b) \\ \wedge & \forall v : is(v) \iff \exists v_1, v_2 : \neg eq(v_1, v_2) \wedge n(v_1, v) \wedge n(v_2, v) & (c) \\ \wedge & \wedge_{q \in PVar} \forall v : r_q(v) \iff \exists v_1 : q(v_1) \wedge n^*(v_1, v) & (d) \end{array}$$

2.3 3-Valued Logical Structures and Embedding

In this section, we define 3-valued logical structures, which provide a way to represent a set of 2-valued logical structures in a compact and conservative way.

We say that the values 0 and 1 are *definite values* and that 1/2 is an *indefinite value*, and define a partial order $\sqsubseteq$ on truth values to reflect information content. $l_1 \sqsubseteq l_2$ denotes that l_1 possibly has more definite information than l_2:

Definition 4. [Information Order]. *For $l_1, l_2 \in \{0, 1/2, 1\}$, we define the* **information order** *on truth values as follows: $l_1 \sqsubseteq l_2$ if $l_1 = l_2$ or $l_2 = 1/2$.*

Definition 5. *A* **3-valued logical structure** *over $\mathcal{P}$ is the generalization of 2-valued structures given in Definition 1, in that predicates may have the value 1/2. This means that $S = \langle U, \iota \rangle$ where for $p \in \mathcal{P}_i$, $\iota(p)$: $(U^S)^i \rightarrow \{0, 1, 1/2\}$. In addition, (i) for all $u \in U^S$, $\iota^S(eq)(u, u) \sqsupseteq 1$, and (ii) for all $u_1, u_2 \in U^S$ such that u_1 and u_2 are distinct nodes, $\iota^S(eq)(u_1, u_2) = 0$.*

A node $u \in U$ having $\iota^S(eq)(u, u) = 1/2$ is called a **summary node***. As we shall see, such a node may represent more than one node from a given 2-valued structure.*

We denote the set of 2-valued logical structures by 2-STRUCT[$\mathcal{P}$]. The set of 3-valued logical structures is denoted by 3-STRUCT[$\mathcal{P}$].

A 3-valued structure can be depicted as a directed graph, with nodes as graph nodes. A unary predicate p is represented in the graph by having a solid arrow from the predicate name p to node u for each node u for which $\iota(p)(u) = 1$. An arrow between two nodes indicates whether a binary predicate holds for the corresponding pair of nodes. An indefinite value of a predicate is shown by a dotted arrow; the value 1 is shown by a solid arrow; and the value 0 is shown by the absence of an arrow.

Example 23 *Fig. 2(d) shows a 3-valued structure that represents possible inputs of the* `insert` *program. This structure represents all lists that are pointed to by program variable* `x` *and have at least two elements. The structure has 2 nodes, u_1 and u_2, where u_1 is the head of the list pointed to by* `x`*, and u_2 is a summary node (drawn as a double circle), which represents the tail of the list. Predicate r_x holds for u_1 and u_2, indicating that all elements of the list are reachable from* `x`*. Other unary predicates are not shown,* *indicating that their values are 0 for all nodes, i.e., the program variables* `y`*,* `e`*, and* `t` *are* `NULL`*, and there is no sharing in the list. The dotted edge from u_1 to u_2 indicates that there may be* `n`*-links from the head of the list to some elements in the tail. In fact, the (u_1, u_2)-edge represents exactly one* `n`*-link that points to exactly one list element, because of conjunct (b) of the integrity formula Example 22. In contrast, the dotted self-loop on u_2 represents all* `n`*-links that may occur in the tail.*

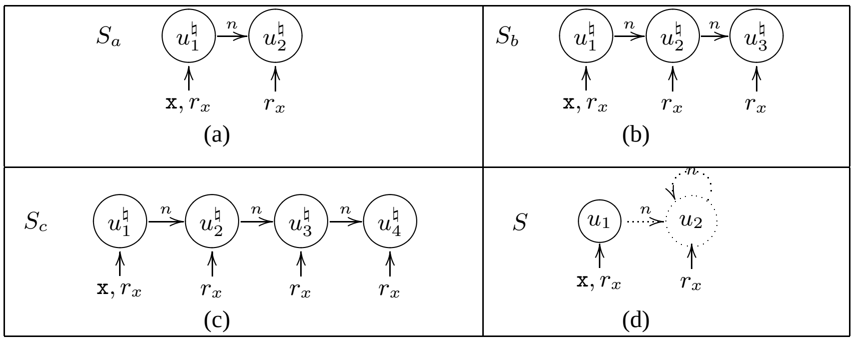


Fig. 2. (a),(b),(c) Examples of 2-valued structures representing linked-lists that are pointed to by program variable `x`, of length 2, 3, and 4, respectively. (d) S represents all lists that are pointed to by program variable `x` and that have at least two elements, including the lists represented by (a)-(c).

2.4 Embedding Order

We define the *embedding ordering* on structures as follows:

Definition 6. *Let $S = \langle U^S, \iota^S \rangle$ and $S' = \langle U^{S'}, \iota^{S'} \rangle$ be two logical structures, and let $f\colon U^S \to U^{S'}$ be a surjective. We say that f* **embeds** *S in S' (denoted by $S \sqsubseteq^f S'$) if for every predicate symbol $p \in \mathcal{P}_i$ and all $u_1, \ldots, u_i \in U^S$,*

$$\iota^S(p)(u_1, \ldots, u_i) \sqsubseteq \iota^{S'}(p)(f(u_1), \ldots, f(u_i)) \tag{1}$$

We say that S **can be embedded in** *S' (denoted by $S \sqsubseteq S'$) if there exists a function f such that $S \sqsubseteq^f S'$.*

Example 24 *Fig. 2(a)-(c) show some of the 2-valued structures that can be embedded into the 3-valued structure S shown in Fig. 2(d). The function that embeds S_a into S maps the node $u_i^\natural \in U^{S_a}$ to $u_i \in U^S$, for $i = 1, 2$. The function that embeds S_b into S maps the node $u_1^\natural \in U^{S_b}$ to $u_1 \in U^S$, and both $u_2^\natural, u_3^\natural \in U^{S_b}$ to $u_2 \in U^S$. Also, Eq. (1) holds, because whenever a predicate has a definite value in S, the corresponding predicate in S_b has the same value. For example, $\iota^S(x)(u_2)$ is 0 and $f(u_2^\natural) = f(u_3^\natural) = u_2$, and both $\iota^{S_b}(x)(u_2^\natural)$ and $\iota^{S_b}(x)(u_3^\natural)$ are 0. Similarly, $\iota^S(r_x)(u_2) = 1$, and both $\iota^{S_b}(r_x)(u_2^\natural)$ and $\iota^{S_b}(r_x)(u_3^\natural)$ are 1. For a binary predicate, $\iota^S(n)(u_2, u_1) = 0$, and both $\iota^{S_b}(n)(u_2^\natural, u_1^\natural)$ and $\iota^{S_b}(n)(u_3^\natural, u_1^\natural)$ are 0.*

Remark. Embedding can be viewed as a variant of homomorphism [19]. In cases where S is a 2-valued structure (i.e., all predicates in S have definite values, including eq, which is interpreted as standard equality), checking whether a 2-valued structure S'

embeds into S is equivalent to checking whether there is an isomorphism between S' and S. In cases where all nodes in S are summary nodes (i.e., for all $u \in U^S$, $\iota^S(eq)(u,u) = 1/2$), and all other values of predicates are definite, embedding is equivalent to strong homomorphism. In cases where all nodes in S are summary nodes and all other values of predicates are either 0 or $1/2$, embedding is equivalent to homomorphism. In all other cases, i.e, when a predicate value for some tuple in S is 1, embedding generalizes the notion of homomorphism.

Remark. In Definition 6, we require that f be surjective in order to guarantee that a quantified formula, such as $\exists v : \varphi$, has consistent values in two 3-valued structures S and S' related by embedding. For example, if f were not surjective, then there could exist an individual $u' \in U^{S'}$, not in the range of f, such that the value of S' on φ is 1 when v is assigned to u'. This would permit there to be structures S and S' for which the value of $\exists v : \varphi$ on S is 0 but its value on S' is 1.

Concretization of 3-Valued Structures. Embedding allows us to define the (potentially infinite) set of concrete structures that a set of 3-valued structures represents:

Definition 7. Concretization of 3-Valued Structures *For a set of structures $X \subseteq$ 3-STRUCT$[\mathcal{P}]$, we denote by $\gamma(X)$ the set of 2-valued structures that X represents, i.e.,*

$$\gamma(X) = \{S^\natural \in \textit{2-STRUCT}[\mathcal{P}] \mid \textit{exists } S \in X \textit{such that } S^\natural \sqsubseteq S \textit{ and } S^\natural \models F\} \quad (2)$$

Also, for a singleton set $X = \{S\}$ we write $\gamma(S)$ instead of $\gamma(X)$.

Example 25 *Example 24 shows that $S_a \sqsubseteq S$, $S_b \sqsubseteq S$, and $S_c \sqsubseteq S$ for the 2-valued structures in Figs. 2(a-c); also, the integrity formula is satisfied for S_a, S_b, and S_c. Therefore, S_a, S_b, and S_c are in the concretization of 3-valued structure S: $S_a, S_b, S_c \in \gamma(S)$. Note that the indefinite values of predicates in S allow the corresponding values in S_b to be either 0 or 1. In particular, $\iota^S(eq)(u_2, u_2) = 1/2$ reflects the fact that the abstract node u_2 may represent more than one concrete node. Indeed, S_b contains two nodes, $u_2^\natural$ and $u_3^\natural$, that are represented by $u_2 \in S$. Also, $\iota^S(eq)(u_2^\natural, u_3^\natural) = 0$, but $\iota^S(eq)(u_2^\natural, u_2^\natural) = 1$.*

The abstract domain we consider is the powerset of 3-valued structures, where the ordering relation $\sqsubseteq$ is defined as follows: for every two sets of 3-valued structures X_1 and X_2, $X_1 \sqsubseteq X_2$ iff for all $S_1 \in X_1$ there exists $S_2 \in X_2$ such that S_1 is embedded into S_2.

The Analysis Technique The TVLA ([38]) system carries out an abstract interpretation [9] to collect a set of structures at each program point P. This involves finding the least fixed point of a certain set of equations. To ensure termination, the analysis is carried out with respect to a finite abstract domain, that is, the set of different structures is finite. When the fixed point is reached, the structures that have been collected at program point p describe a superset of all the concrete stores that can occur at p. To determine whether a query is always satisfied at p, one checks whether it holds in all of the structures that were collected there. Instantiations of this framework are capable of establishing nontrivial properties of programs that perform complex pointer-based manipulations of *a priori* unbounded-size heap-allocated data structures.

3 Characterizing 3-Valued Structures by First-Order Formulas

This section presents our results on characterizing 3-valued structures using first-order formulas. Given a 3-valued structure S, the question that we wish to answer is whether it is possible to give a formula $\widehat{\gamma}(S)$ that accepts exactly the set of 2-valued structures that S represents, i.e., $S^\natural \models \widehat{\gamma}(S)$ iff $S^\natural \in \gamma(S)$.

This question has different answers depending on what assumptions are made. The task of generating a characteristic formula for a 3-valued structure S is challenging because we have to find a formula that identifies when embedding is possible, i.e., that is satisfied by exactly those 2-valued structures that embed into S. It is not always possible to characterize an *arbitrary* 3-valued structure by a first-order formula, i.e., there exists a 3-valued structure S for which there is no first-order formula with transitive closure that accepts exactly the set of 2-valued structures γ(S).

For example, consider the 3-valued structure S shown in Fig. 3. The absence of a self loop on any of the three summary nodes implies that a 2-valued structure can be embedded into this structure if and only if it can be colored using 3 colors (Lemma D1 in the appendix). It is well-known that there exists no first-order formula, even with transitive closure, that expresses 3-colorability of undirected graphs, unless $P = NP$ (e.g., see [25, 8]). Therefore, there is no first-order formula that accepts exactly the set $\gamma(S)$.

Remark. In fact, the condition is even stronger. First-order logic with transitive closure can only express non-deterministic logspace (NL) computations, thus, the NP-complete problem of 3-colorability is not expressible in first-order logic, unless $NL = NP$. It is shown in [25] using an ordering relation on the nodes. In our context, without the ordering, the logic is less expressive. Thus, the condition under which 3-colorability is expressible is even stronger than $NL = NP$. We believe that there is an example of a 3-valued structure that is not expressible in the logic, independently of the question whether $P = NP$. However, it is not the main focus of the current paper.

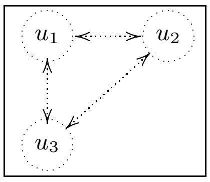


Fig. 3. A 3-valued structure that represents 3-colorable undirected graphs. A 2-valued structure can be embedded into this structure if and only if it can be colored using 3 colors.

3.1 FO-Identifiable Structures

Intuitively, the difficulty in characterizing 3-valued structures is how to uniquely identify the correspondence between concrete and abstract nodes using a first-order formula. Fortunately, as we will see, for the subclass of 3-valued structures used in shape analysis (also known as "bounded structures"), the correspondence can be easily defined using first-order formulas. The bounded structures are a subclass of the 3-valued structures in which it is possible to identify uniquely each node using a first-order formula.

Definition 8. *A 3-valued structure S is called* **FO-identifiable** *if for every node $u \in U^S$ there exists a first-order formula node$^S_u(w)$ with designated free variable w such* *that for every* 2*-valued structure* $S^\natural$ *that embeds into* S *using a function* f*, for every concrete node* $u^\natural \in U^{S^\natural}$ *and for every node* $u_i \in U^S$*:*

$$f(u^\natural) = u_i \iff S^\natural, [w \mapsto u^\natural] \models \mathit{node}^S_{u_i}(w) \tag{3}$$

The idea behind this definition is to have a formula that uniquely identifies each node u of the 3-valued structure S. This will be used to identify the set of nodes of a 2-valued structure that are mapped to u by embedding. In other words, a concrete node $u^\natural$ satisfies the $node$ formula of at most one abstract node, as formalized by the lemma:

Lemma 1. *Let* S *be an FO-identifiable structure, and let* $u_1, u_2 \in S$ *be distinct nodes. Let* $S^\natural$ *be a 2-valued structure that embeds into* S *and let* $u^\natural \in S^\natural$*. At most one of the following hold:*

1. $S^\natural, [w \mapsto u^\natural] \models \mathit{node}^S_{u_1}(w)$
2. $S^\natural, [w \mapsto u^\natural] \models \mathit{node}^S_{u_2}(w)$

Remark. Definition 8 can be generalized to handle arbitrary 2-valued structures, by also allowing extra designated free variables for every concrete node and using equality to check if the concrete node is equal to the designated variable: $\mathrm{node}^S_{u_i}(w, v_1, \ldots, v_n) \stackrel{\text{def}}{=} w = v_i$. However, the equality formula cannot be used to identify nodes in a 3-valued structure because equality evaluates to $1/2$ on summary nodes.

We now introduce a standard concept for turning valuations into formulas.

Definition 9. *For a predicate* p *of arity* k *and truth value* $B \in \{0, 1, 1/2\}$*, we define the formula* $p^B(v_1, v_2, \ldots, v_k)$ *to be the* **characteristic formula of** B **for** p*, by*

$$\begin{array}{ll} p^0(v_1, v_2, \ldots, v_k) & \stackrel{\text{def}}{=} \neg p(v_1, v_2, \ldots, v_k) \\ p^1(v_1, v_2, \ldots, v_k) & \stackrel{\text{def}}{=} p(v_1, v_2, \ldots, v_k) \\ p^{1/2}(v_1, v_2, \ldots, v_k) & \stackrel{\text{def}}{=} 1 \end{array}$$

The main idea in the above definition is that, for $B \in \{0, 1\}$, p^B holds when the value of p is B, and for $B = 1/2$ the value of p is unrestricted. This is formalized by the following lemma:

Lemma 2. *For every* 2*-valued structure* $S^\natural$ *and assignment* Z

$$S^\natural, Z \models p^B(v_1, \ldots, v_k) \textit{ iff } \iota^{S^\natural}(p)(Z(v_1), \ldots, Z(v_k)) \sqsubseteq B$$

Definition 8 is not a constructive definition, because the premises range over arbitrary 2-valued structures and arbitrary embedding functions. For this reason, we now introduce a testable condition that implies FO-identifiability.

Bounded Structures. The following subclass of 3-values structures was defined in [50];[6] the motivation there was to guarantee that shape analysis was carried out with respect to a finite set of abstract structures, and hence that the analysis would always terminate.

[6] This definition of bounded structures was given in [50]; it is slightly more restrictive than the one given in [51, 35], which did not impose requirement 10(ii). However, it does not limit the set of problems handled by our method, if the structure that is bounded in the weak sense is also FO-identifiable.

Definition 10. *A* **bounded structure** *over vocabulary $\mathcal{P}$ is a structure $S = \langle U^S, \iota^S \rangle$ such that for every $u_1, u_2 \in U^S$, where $u_1 \neq u_2$, there exists a predicate symbol $p \in \mathcal{P}_1$ such that (i) $\iota^S(p)(u_1) \neq \iota^S(p)(u_2)$ and (ii) both $\iota^S(p)(u_1)$ and $\iota^S(p)(u_2)$ are not $1/2$, i.e., $\iota^S(p)(u_1), \iota^S(p)(u_2) \in \{0, 1\}$.*

Intuitively, for each pair of nodes in a bounded structure, there is at least one predicate that has different definite values for these nodes. Thus, there is a finite number of different bounded structures (up to isomorphism).

The following lemma shows that bounded structures are FO-identifiable using formulas over unary predicates only (denoted by $\mathcal{P}_1$):

Lemma 3. *Every bounded 3-valued structure S is FO-identifiable , where*

$$\textit{node}^S_{u_i}(w) \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}_1} p^{\iota^S(p)(u_i)}(w) \tag{4}$$

Example 31 *The first-order $node$ formulas for the structure S shown in Fig. 2, are:*

$$\begin{aligned}\textit{node}^S_{u_1}(w) = {} & x(w) \wedge r_x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w) \\ & \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w) \\ \textit{node}^S_{u_2}(w) = {} & \neg x(w) \wedge r_x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w) \\ & \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)\end{aligned}$$

Remark. In the case that S is a bounded 2-valued structure, the definition of a bounded structure becomes trivial. The reason is that every node in S can be named by a quantifier-free formula built from unary predicates. This is essentially the same as saying that every node can be named by a constant. If structure S' embeds into S, then S' must be isomorphic to S, therefore it is possible to name all nodes of S' by the same constants. However, this restricted case is not of particular interest for us, because, to guarantee termination, shape analysis operates on structures that contain summary nodes and indefinite values. In the case that S contains a summary node, a structure S' that embeds into S may have an unbounded number of nodes; hence the nodes of S' cannot be named by a finite set of constants in the language.

We already know of interesting cases of FO-identifiable structures that are not bounded, which can be used to generalize the abstraction defined in [50]:

Example 32 *The 3-valued structure S' in Fig. 4 is FO-identifiable by:*

$$\begin{aligned}\textit{node}^{S'}_{u_1}(w) \stackrel{\text{def}}{=} {} & x(w) \wedge r_x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w) \\ & \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w) \\ \textit{node}^{S'}_{u_2}(w) \stackrel{\text{def}}{=} {} & \underline{\exists w_1 : x(w_1) \wedge n(w_1, w)} \wedge \neg x(w) \wedge r_x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w) \\ & \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w) \\ \textit{node}^{S'}_{u_3}(w) \stackrel{\text{def}}{=} {} & \neg(\exists w_1 : x(w_1) \wedge n(w_1, w)) \wedge \neg x(w) \wedge r_x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w) \\ & \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)\end{aligned}$$

However, S' is not a bounded structure because nodes u_2 and u_3 have the same values of unary predicates. To distinguish between these nodes, we extended node$^{S'}_{u_2}(w)$ with the underlined subformula, which captures the fact that only u_2 is directly pointed to by an n-edge from u_1.

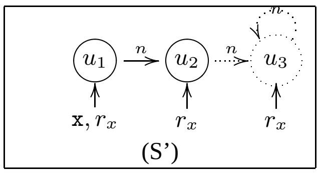


Fig. 4. A 3-valued structure S' is FO-identifiable, but not bounded.

It can be shown that every FO-identifiable structure can be converted into a bounded structure by introducing more instrumentation predicates. For methodological reasons, we use the notion of FO-identifiable which directly capture the ability to uniquely identify embedding via (FO) formulas.[7] One of the interesting features of FO-identifiable structures is that the structures generated by a common TVLA operation "focus", defined in [35], are all FO-identifiable (see Lemma D2 in Appendix D). For example, Fig. 4 shows the structure S', which is one of the structures resulting from applying the"focus" operation to the structure S from Fig. 2(d) with the formula $\exists v_1, v_2 : x(v_1) \wedge n(v_1, v_2)$. S' is FO-identifiable, but not bounded. However, structures like the one shown in Fig. 3 are not FO-identifiable unless $P = NP$.

3.2 Characterizing FO-identifiable structures

To characterize an FO-identifiable 3-valued structure, we must ensure

1. the existence of a surjective embedding function.
2. that every concrete node is represented by some abstract node.
3. that corresponding concrete and abstract predicate values meet the embedding condition of Eq. (1).

Definition 11. First-order Characteristic Formula *Let $S = \langle U = \{u_1, u_2, \ldots, u_n\}, \iota\rangle$ be an FO-identifiable* 3*-valued structure.*

We define the **totality characteristic formula** *to be the closed formula:*

$$\xi^S_{total} \overset{\text{def}}{=} \forall w : \bigvee_{i=1}^{n} \mathit{node}^S_{u_i}(w) \tag{5}$$

We define the **nullary characteristic formula** *to be the closed formula:*

$$\xi^S_{nullary} \overset{\text{def}}{=} \bigwedge_{p \in \mathcal{P}_0} p^{\iota^S(p)()} \tag{6}$$

For a predicate p *of arity* $r \geq 1$, *we define the* **predicate characteristic formula** *to be the closed formula:*

$$\xi^S[p] \overset{\text{def}}{=} \forall w_1, \ldots, w_r : \bigwedge_{\{u'_1, \ldots, u'_r\} \in U}$$
$$\bigwedge_{j=1}^{r} \mathit{node}^S_{u'_j}(w_j) \Rightarrow p^{\iota^S(p)(u'_1, \ldots, u'_r)}(w_1, \ldots, w_r) \tag{7}$$

[7] In subsequent sections, we redefine this notion to capture other classes of structures.

The **characteristic formula of** S *is defined by:*

$$\begin{aligned}\xi^S \overset{\text{def}}{=} & \textstyle\bigwedge_{i=1}^{n} (\exists v : \mathit{node}_{u_i}^S(v)) \\ & \wedge\ \xi_{total}^S \\ & \wedge\ \xi_{nullary}^S \\ & \wedge\ \textstyle\bigwedge_{r=1}^{maxR} \bigwedge_{p \in \mathcal{P}_r} \xi^S[p]\end{aligned} \tag{8}$$

The **characteristic formula of set** $X \subseteq$ **3-STRUCT**$[\mathcal{P}]$ *is defined by:*

$$\widehat{\gamma}(X) = F \wedge (\bigvee_{S \in X} \xi^S) \tag{9}$$

Finally, for a singleton set $X = \{S\}$ *we write* $\widehat{\gamma}(S)$ *instead of* $\widehat{\gamma}(X)$*.*

The main ideas behind the four conjuncts of Eq. (8) are:

– The existential quantification in the first conjunct requires that the 2-valued structures have at least n distinct nodes. For each abstract node in S, the first sub-formula locates the corresponding concrete node. Overall, this conjunct guarantees that embedding is surjective.
– The totality formula ensures that every concrete node is represented by some abstract node. It guarantees that the embedding function is well-defined.
– The nullary characteristic formula ensures that the values of nullary predicates in the 2-valued structures are at least as precise as the values of the corresponding nullary predicates in S.
– The predicate characteristic formulas guarantee that predicate values in the 2-valued structures obey the requirements imposed by an embedding into S.[8]

Example 33 *After a small amount of simplification, the characteristic formula* $\widehat{\gamma}(S)$ *for the structure* S *shown in Fig. 2 is* $F_{List} \wedge \xi^S$*, where* ξ^S *is:*

$$\begin{aligned}& \exists v : \mathit{node}_{u_1}^S(v) \wedge \exists v : \mathit{node}_{u_2}^S(v) \\ \wedge\ & \forall w : \mathit{node}_{u_1}^S(w) \vee \mathit{node}_{u_2}^S(w) \\ \wedge\ & \textstyle\bigwedge_{p \in \mathcal{P}_1} \forall w_1 : \bigwedge_{i=1,2} (\mathit{node}_{u_i}^S(w_1) \Rightarrow p^{\iota^S(p)(u_i)}(w_1)) \\ \wedge\ & \forall w_1, w_2 :\ (\mathit{node}_{u_1}^S(w_1) \wedge \mathit{node}_{u_1}^S(w_2) \Rightarrow eq(w_1, w_2) \wedge \neg n(w_1, w_2) \wedge \neg n(w_2, w_1)) \\ & \qquad \wedge\ (\mathit{node}_{u_1}^S(w_1) \wedge \mathit{node}_{u_2}^S(w_2) \Rightarrow \neg eq(w_1, w_2) \wedge \neg n(w_2, w_1))\end{aligned}$$

The $node$ *formulas are given in Example 31, and the predicates for the* `insert` *program in Fig. 1(b) are shown in Table 1. Above, we simplified the formula from Eq. (8) by combining implications that had the same premises. The integrity formula* F_{List} *is given in Example 22. Note that it uses transitive closure to define the reachability predicates; consequently,* $\widehat{\gamma}(S)$ *is a formula in first-order logic with transitive closure.*

[8] Definition 11 relates to all FO-identifiable structures, not only to bounded structures. For bounded structures, it can be simplified by omitting $\xi^S[p]$ for all unary predicates p, because it is implied by ξ_{total}^S. In fact, it can be omitted only for the abstraction predicates, as defined in [51]; however throughout this paper we consider all unary predicates to be abstraction predicates.

When a predicate has an indefinite value on some node tuple, a corresponding conjunct of Eq. (7) can be omitted, because it simplifies to **1**.

Thus, the size of this simplified version of ξ^S is linear in the number of definite values of predicates in S. Assuming that the $node^S$ formulas contain no quantifiers or transitive-closure operator, e.g., when S is bounded, the ξ^S formula has no quantifier alternation, and does not contain any occurrences of the transitive-closure operator. Thus, the formula $\widehat{\gamma}$ is in Existential-Universal normal form (and thus decidable for satisfiability) whenever F is in Existential-Universal normal form and does not contain transitive closure.[9] Moreover, if the maximal arity of the predicate in $\mathcal{P}$ is 2, then $\widehat{\gamma}$ is in the two-variable fragment of first-order logic [43], wherever F is. In Section 5, we discuss other conditions under which $\widehat{\gamma}$ can be expressed in a decidable logic.

The following theorem shows that for every FO-identifiable structure S, the formula $\widehat{\gamma}(S)$ accepts exactly the set of 2-valued structures represented by S.

Theorem 1. *For every FO-identifiable* 3*-valued structure* S*, and* 2*-valued structure* $S^\natural$*,* $S^\natural \in \gamma(S)$ *iff* $S^\natural \models \widehat{\gamma}(S)$*.*

4 Supervaluational Semantics for First-Order Formulas

In this section, we consider the problem of how to extract information from a 3-valued structure by evaluating a query. A compositional semantics for 3-valued first-order logic is defined in [51]; however, that semantics is not as precise as the one defined here. The semantics given in this section can be seen as providing the limit of obtainable precision.

The Notion of Supervaluational Semantics defined below, has been used in [56, 4].

Definition 12. Supervaluational Semantics of First-Order Formulas *Let* X *be a set of* 3*-valued structures and* φ *be a closed formula. The* **supervaluational semantics of** φ **in** X*, denoted by* $\langle\!\langle\varphi\rangle\!\rangle(X)$*, is defined to be the join of the values of* φ *obtained from each of the* 2*-valued structures that* X *represents, i.e., the most-precise conservative value that can be reported for the value of formula* φ *in the* 2*-valued structures represented by* X *is*

$$\langle\!\langle\varphi\rangle\!\rangle(X) = \begin{cases} 1 & \text{if } S^\natural \models \varphi \text{ for all } S^\natural \in \gamma(X) \\ 0 & \text{if } S^\natural \not\models \varphi \text{ for all } S^\natural \in \gamma(X) \\ 1/2 & \text{otherwise} \end{cases} \tag{10}$$

The compositional semantics given in [51] and used in TVLA can yield $1/2$ for φ, even when the value of φ is 1 for all the 2-valued structures $S^\natural$ that S represents (or when the value of φ is 0 for all the $S^\natural$). In contrast, when the supervaluational semantics yields $1/2$, we *know* that any sound extraction of information from S must return $1/2$.

Example 41 *We demonstrate now that the supervaluational semantics of the formula* $\varphi_{\texttt{x}\to\texttt{next}\neq\texttt{NULL}} \overset{\text{def}}{=} \exists v_1, v_2 : x(v_1) \wedge n(v_1, v_2)$ *on the structure S from Fig. 2(d) is* 1*. That is, we wish to argue that for all of the* 2*-valued structures that structure* S *from Fig. 2(d) represents, the value of the formula* $\varphi_{\texttt{x}\to\texttt{next}\neq\texttt{NULL}}$ *must be* 1*.*

[9] For practical reasons, we often replace the $node$ formula by a new (definable) predicate, and add its definition to the integrity formula.

We reason as follows: S represents a list with at least two nodes; i.e., all 2-valued structures represented by S have at least two nodes. One node, $u_1^\natural$, corresponding to u_1 in S, is pointed to by program variable `x`*. The other node, corresponding to the summary node u_2, must be reachable from* `x`*. Consider the sequence of nodes reachable from* `x`*, starting with $u_1^\natural$. Denote the first node in the sequence that embeds into u_2 by $u_2^\natural$. By the definition of reachability, there must be an* `n`*-link to $u_2^\natural$ from a node embedded into u_1. But the integrity rules guarantee that there is exactly one node that embeds into u_1, namely, $u_1^\natural$. Therefore, the formula $x(v_1) \wedge n(v_1, v_2)$ holds for $[v_1 \mapsto u_1^\natural, v_2 \mapsto u_2^\natural]$.*

Note that this formula will be evaluated to $1/2$ by TVLA, because $x(v_1) \wedge n(v_1, v_2)$ evaluates to $1/2$ under the assignment $[v_1 \mapsto u_1, v_2 \mapsto u_2]$: the compositional semantics yields $x(u_1) \wedge n(u_1, u_2) = 1 \wedge 1/2 = 1/2$.

Notice that Definition 12 does not provide a constructive way to compute $\langle\!\langle \varphi \rangle\!\rangle(X)$ because $\gamma(X)$ is usually an infinite set.

Computing Supervaluational Semantics using Theorem Provers. If an appropriate theorem prover is at hand, $\langle\!\langle \varphi \rangle\!\rangle(S)$ can be computed with the procedure shown in Fig. 5. This procedure is not an algorithm, because the theorem prover might not terminate. Termination can be assured by using standard techniques (e.g., having the theorem prover return a safe answer if a time-out threshold is exceeded) at the cost of losing the ability to guarantee that a most-precise result is obtained. If the queries posed by operation `Supervaluation` can be expressed in a decidable logic, the algorithm for computing supervaluation can be implemented using a decision procedure for that logic. In Section 5, we discuss such decidable logics that are useful for shape analysis.

```
procedure Supervaluation($\varphi$: Formula,
                         X: Set of $3$-valued structures): Value
    if ($\widehat{\gamma}(X) \Rightarrow \varphi$ is valid) return $1$;
    else if ($\widehat{\gamma}(X) \Rightarrow \neg\varphi$ is valid) return $0$;
    otherwise return $1/2$;
```

Fig. 5. A procedure for computing the supervaluational value of a formula φ that encodes a query on a 3-valued structures S.

5 Applications

The experiments discussed in this section demonstrate how the $\widehat{\gamma}$ operation can be harnessed in the context of program analysis: the results described below go beyond what previous systems were capable of. In Section 5.1, we discuss the use existing theorem provers and their limitations. In Section 5.2, we suggest a way to overcome these limitations, using decidable logic.

We present two examples that use $\widehat{\gamma}$ to read out information from 3-valued structures in a conservative, but rather precise way. The first example demonstrates how supervaluational semantics allows us to obtain more precise information from a 3-valued structure than we would have using compositional semantics. The second example demonstrates how to use the 3-valued structures obtained from a TVLA analysis to construct a loop invariant; this is then used to show that certain properties of a linked data structure hold on each loop iteration. In addition, we briefly describe how $\widehat{\gamma}$ can be used in algorithms for computing most-precise abstraction operations for shape analysis. Finally, we report on other work that employs $\widehat{\gamma}$ to generate a concrete counter-example for shape analysis.

Remark. The $\widehat{\gamma}$ operation defines a symbolic concretization with respect to a given abstract domain. In Section 3, we defined $\widehat{\gamma}$ for the abstract domain of sets of 3-valued structures. In Appendix A, we describe a related abstract domain and define $\widehat{\gamma}$ for it. The applications described in this section can be used with any domain for which $\widehat{\gamma}$ is defined in some logic and a theorem prover for that logic exists. In our examples, we use $\widehat{\gamma}$ defined in Section 3 and the first-order logic with transitive closure.

5.1 Using the First-Order Theorem Prover SPASS

The TVLA ([38]) system performs an iterative fixed-point computation, which yields at every program point p a set X_p of bounded structures. It guarantees that $\gamma(X_p)$ is a superset of the 2-valued structures that can arise at p in any execution. We have implemented the $\widehat{\gamma}$ operation in TVLA, and employed SPASS [57] to check, using the formula $\widehat{\gamma}(X_p)$, that certain properties of the heap hold at program point p. Also, we implemented the supervaluational procedure described in Section 4, employing SPASS. The enhanced version of TVLA generates the formula $\widehat{\gamma}(S)$ and makes at most two calls to SPASS to compute the supervaluational value of a query φ in structure S. In this section, we report on our experience in using SPASS and the problems we have encountered.

First, calls to SPASS theorem prover need not terminate, because first-order logic is undecidable in general. However, in the examples described below, SPASS always terminated.

Example 51 *In Example 41 we (manually) proved that the supervaluational value of the formula $\varphi_{\texttt{x}\rightarrow\texttt{next}\neq\texttt{NULL}}$ on the structure S from Fig. 2(d) is 1. To check this automatically, we used SPASS to determine the validity of $\widehat{\gamma}(S) \Rightarrow \varphi_{\texttt{x}\rightarrow\texttt{next}\neq\texttt{NULL}}$; SPASS indicated that the formula is valid. This guarantees that the formula $\varphi_{\texttt{x}\rightarrow\texttt{next}\neq\texttt{NULL}}$ evaluates to 1 on all of the 2-valued structures that embed into S.*

In contrast, TVLA uses Kleene semantics for 3-valued formulas, and will evaluate the formula $\varphi_{\texttt{x}\rightarrow\texttt{next}\neq\texttt{NULL}}$ to $1/2$: under the assignment $[v_1 \mapsto u_1, v_2 \mapsto u_2]$, $x(v_1) \wedge n(v_1, v_2)$ evaluates to $1 \wedge 1/2$, which equals $1/2$.

Generating and Querying a Loop Invariant We used TVLA to compute, for each program point p, a set X_p of bounded structures that overapproximate the set of stores that may occur at that point. We then generated $\widehat{\gamma}(X_p)$. Because TVLA is sound, $\widehat{\gamma}(X_p)$ must be an invariant that holds at program point p, according to Theorem 1. In particular, when p is a program point that begins a loop, $\widehat{\gamma}(X_p)$ is a loop invariant.

Example 52 *Let $X = \{S_i \mid i = 1, \ldots, 5\}$ denote the set of five 3-valued structures that TVLA found at the beginning of the loop in the `insert` program from Fig. 2. Table 2 and Table 3 of Appendix C show the S_i and their characteristic formulas. The loop invariant is defined by*

$$\widehat{\gamma}(X) = F_{List} \wedge (\bigvee_{i=1}^{5} \xi^{S_i})$$

Using SPASS, this formula was then used to check that in every structure that can occur at the beginning of the loop, x *points to a valid list, i.e., one that is acyclic and unshared. This property is defined by the following formulas:*

$$\begin{array}{ll} acyc_x & \stackrel{\text{def}}{=} \forall v_1, v_2 : r_x(v_1) \land n^+(v_1, v_2) \Rightarrow \neg n^+(v_2, v_1) \\ uns_x & \stackrel{\text{def}}{=} \forall v\colon r_x(v) \Rightarrow\ \neg(\exists w_1, w_2\colon \neg eq(w_1, w_2) \land n(w_1, v) \land n(w_2, v)) \\ list_x & \stackrel{\text{def}}{=} acyc_x \land uns_x \end{array}$$

We applied SPASS to check the validity of $\widehat{\gamma}(S) \Rightarrow list_x$; SPASS indicated that the formula is valid.[10]

In addition to the termination issue, a second obstacle is that SPASS considers infinite structures, which are not allowed in our setting.[11] As a consequence, SPASS can fail to verify that a formula is valid for our intended set of structures; however, the opposite can never happen: whenever SPASS indicates that a formula is valid, it is indeed valid for our intended set of structures.

Example 53 *We tried to verify that every concrete linked-list represented by the 3-valued structure S from Fig. 2(d) has a last element. This condition is expressed by the formula $\varphi_{last} \stackrel{\text{def}}{=} \exists v_1 \forall v_2 : \neg n(v_1, v_2)$. The supervaluational value of φ_{last} on a structure S is $\langle\!\langle \varphi \rangle\!\rangle(S) = 1$, for the following reasons. Because r_x has the definite value 1 on u_2 in S, all concrete nodes represented by the summary node u_2 must be reachable from x. Thus, these nodes must form a linked list, i.e., each of these concrete nodes, except for one node that is the "last", has an n-edge to another concrete node represented by u_2. The last node does not have an n-edge back to any of the nodes represented by u_2, because that would create sharing, whereas the value of predicate is in S is 0 on u_2. Also, the last node cannot have an n-edge to the concrete node represented by u_1, because the value of predicate n on the pair $\langle u_2, u_1 \rangle$ in S is 0. Therefore, the last element cannot have an outgoing n-edge.*

We used SPASS to determine the validity of $\widehat{\gamma}(S) \Rightarrow \varphi_{last}$; SPASS indicated that the formula is not *valid, because it considered a structure that has infinitely many concrete nodes, all represented by u_2. Each of these concrete nodes has an n-edge to the next node.*

The validity test of the formula $\widehat{\gamma}(S) \Rightarrow \neg\varphi_{last}$ failed, of course, because there exists a finite structure that is represented by S (and thus satisfies $\widehat{\gamma}(S)$) and has a last element. For example, the structure in Fig. 2(a) that represents a list of size 2. Therefore, the procedure Supervaluation(φ_{last}, S) implemented using SPASS returns $1/2$, even though the supervaluational value of φ_{last} on S is 1.

The third, and most severe, problem that we face is that SPASS does not support transitive closure. Because transitive closure is not expressible in first-order logic, we could only partially model transitive closure in SPASS, as described below.

SPASS follows other theorem provers in allowing axioms to express requirements on the set of structures considered. We used SPASS axioms to model integrity rules. To partially model transitive closure, we replaced uses of $n^+(v_1, v_2)$ by uses of a new

[10] SPASS input is available from `www.cs.tau.ac.il/~gretay`.

[11] Our intended structures are finite, because they represent memory configurations, which are guaranteed to be finite, although their size is not bounded.

designated predicate $t[n](v_1, v_2)$. Therefore, SPASS will consider some structures that do not represent possible stores. As a consequence, SPASS can fail to verify that a formula is valid for our intended set of structures; however, the opposite can never happen: whenever SPASS indicates that a formula is valid, it is indeed valid for our intended set of structures. To avoid some of the spurious failures to prove validity, we added axioms to guarantee that (i) $t[n](v_1, v_2)$ is transitive and (ii) $t[n](v_1, v_2)$ includes all of $n(v_1, v_2)$; thus, $t[n](v_1, v_2)$ includes all of $n^+(v_1, v_2)$. Because transitive closure requires a minimal set, which is not expressible in first-order logic, this approach provides a looser set of integrity rules than we would like. However, it is still the case that whenever SPASS indicates that a formula is valid, it is indeed valid for the set of structures in which $t[n](v_1, v_2)$ is exactly $n^+(v_1, v_2)$.

Example 54 *SPASS takes into account the structure shown in Fig. 6, in which the value of* $t[n](u_1, u_3)$ *is* 1*, but the value of* $n^+(u_1, u_3)$ *is* 0 *because there is no* n*-edge from* u_2 *to* u_3*.*

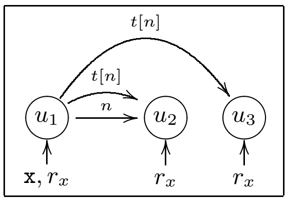


Fig. 6. SPASS takes into account structures in which the $t[n]$ predicate overapproximates the n^+ predicate, such as the structure shown in this figure.

Remark. For practical purposes, the success of using symbolic methods depends on having a terminating theorem prover. We have successfully used SPASS as part of a prototype implementation of the $assume$ operation (Section 5.3), and the path-pruning optimization for counter-example generation (Section 5.4). Although these experiments are rather preliminary, we believe that this approach can be made to work in practice. For example, there has been some recent progress in using SPASS, including the use of transitive closure [36]. Also, we have investigated a complementary approach, discussed in Section 5.2.

5.2 Decidable Logic

The obstacles mentioned in Section 5.1 are not specific to SPASS. They occur in all theorem provers for first-order logic that we are aware of. To address these obstacles, we are investigating the use of a decidable logic. To reason about linked data structures, we need a notion of reachability to be expressible, for example, using transitive closure. However, a logic that is both decidable and includes reachability must be limited in other aspects.

One such example is the decidable second-order theory of two successors *WS2S* [46]; its decision procedure is implemented in a tool called MONA [23]. Second-order quantification suffices to express reachability, but there are still two problems. First, the decision procedure for *WS2S* is necessarily non-elementary [41]. Second, *WS2S* only applies to trees, or, equivalently, to function graphs (graphs with at most one edge leaving any vertex).

Another example is $EA(TC, f^1)$, which is a subset of first-order logic with transitive closure, in which the following restriction are imposed on formulas: (i) they must be in existential-universal form, and (ii) they must use at most a single unary function f, but can use an arbitrary number of unary predicates. [26] shows that the decision procedure for satisfiability of $EA(TC, f^1)$ is NEXPTIME-complete.

In spite of their limitations, both *WS2S* and $EA(TC, f^1)$ can be useful for reasoning about shape invariants and mutation operations on data structures, such as singly and doubly linked lists, (shared) trees, and graph types [30]. The key is the *simulation technique* [27], which encodes complex data-structures using *tractable* structures, e.g., function graphs or simple trees, where we can reason with decidable logics.

For example, given a suitable simulation, $\widehat{\gamma}$ formula can be expressed in *WS2S* and $EA(TC, f^1)$ if the integrity formula F can. This follows from the definition of $\widehat{\gamma}$ in Eq. (9) and the fact that ξ^S does not contain quantifier alternation. This makes $EA(TC, f^1)$ and *WS2S* candidate implementations for the decision procedure used in the supervaluational semantics and in the algorithms described below.

5.3 Assume-Guarantee Shape Analysis

The $\widehat{\gamma}$ operation is useful beyond computing supervaluational semantics: it is a necessary operation used in the algorithms described in [62, 48]. These algorithms perform abstract operations symbolically by representing abstract values as logical formulas, and use a theorem prover to check validity of these formulas. These algorithms improve on existing shape-analysis techniques by:

- conducting abstract interpretation in the most-precise fashion, improving the technique used in the TVLA system [38, 51], which provides no guarantees about the precision of its basic mechanisms.
- performing modular verification using assume-guarantee reasoning and procedure specifications. This is perhaps the most-exciting potential application of $\widehat{\gamma}$ (and $EA(TC, f^1)$ logic), because existing mechanisms for shape analysis, including TVLA, do not support assume-guarantee reasoning.

5.4 Counter-example Generation

Some preliminary work to use the techniques presented in this paper to improve the applicability of TVLA has been carried out. The tool described in [13, 12] uses the $\widehat{\gamma}$ operation to generate a concrete counter-example for a potential error message produced by TVLA for an intermediate 3-valued structure S at a program point p. Such a tool is useful to check if a reported error is a real error or a false-alarm, i.e., it never occurs on any concrete store.

Generation of concrete counter-examples from S proceeds as follows. First, S is converted to the formula $\widehat{\gamma}(S)$. Then, the tool uses weakest precondition to generate a formula that represents the stores at the entry point that lead to an execution trace that reaches p with a store that satisfies $\widehat{\gamma}(S)$. Finally, a separate tool [39] generates a concrete store that satisfies the formula for the entry point.

6 Related Work

There is a sizeable literature on *structure-description formalisms* for describing properties of linked data structures (see [2, 51] for references). The motivation for the present paper was to understand the expressive power of the shape abstractions defined in [51].

In previous work, Benedikt et al. [2] showed how to translate two kinds of shape descriptors, "path matrices" [20, 22] and the variant of shape graphs discussed in [49], into a logic called L_r ("logic of reachability expressions"). The shape graphs from [49] are also amenable to the techniques presented in the present paper: the characteristic formula defined in Eq. (8) is much simpler than the translation to L_r given in [2]; moreover, Eq. (8) applies to a more general class of shape descriptors. However, the logic used in [2] is decidable, which guarantees that terminating procedures can be given for problems that can be addressed using L_r.

The Pointer Analysis Logic Engine (PALE) [42] provides a structure-description formalism that serves as an assertion language; assertions are translated to second-order monadic logic and fed to MONA. PALE does not handle all data structures, but can handle data structures describable as graph types [30]. Because the logic used by MONA is decidable, PALE is guaranteed to terminate.

One point of contrast between the shape abstractions based on 3-valued structures studied in this paper and both L_r and the PALE assertion language is that the powerset of 3-valued structures forms an abstract domain. This means that 3-valued structures can be used for program analysis by setting up an appropriate set of equations and finding its fixed point [51]. In contrast, when PALE is used for program analysis, an invariant must be supplied for each loop.

Other structure-description formalisms in the literature include ADDS [21] and shape types [16].

The supervaluational semantics for first-order logic discussed in Section 4 is related to a number of other supervaluational semantics for partial logics and 3-valued logics discussed in the literature [56, 3, 4]. Compared to previous work, an innovation of Fig. 5 is the use of $\widehat{\gamma}$ to translate a 3-valued structure to a formula. In fact, Fig. 5 is an example of a general reductionist strategy for providing a supervaluational evaluation procedure for abstract domains by using existing logics and theorem-provers/decision-procedures.

A recent work [32], which is an abbreviated version of a more extensive presentation of the results reported in [33], provides an alternative characterization of 3-valued structures using logical formulas, equivalent to the characterization presented in the present paper. The present paper, which extends and elaborates on the results of [61], unlike [32, 33], reports on experience and algorithmic issues in using logical characterization of structures for shape analysis; this material is important because shape analysis is the primary motivation and the intended application of this paper, as well as [32, 33]. Also, Section A.4 of the present paper gives a simple semantic argument for the property of closure under negation, shown in [33] using a different formalism. The technical similarities and differences between the two works are described in a note available from `www.cs.tau.ac.il/~gretay`.

7 Final Remarks

In [48], we discuss how to perform all operations required for abstract interpretation in the most-precise way possible (relative to the abstraction in use), if certain primitive operations can be carried out, and if a sufficiently powerful theorem prover is at hand. Chief among the primitive operations that must be available is $\widehat{\gamma}$; thus, the material that has been presented in this paper shows how to fulfill the requirements of [48] for a family of abstractions based on 3-valued structures (essentially those used in our past work [51] and in the TVLA system [38]).

In ongoing work, we are investigating the feasibility of actually applying the techniques from [48] to perform abstract interpretation for abstractions based on 3-valued structures. This approach could be more precise than TVLA because, for instance, it would take into account in a first-class way the integrity formula of the abstraction. In contrast, in TVLA some operations temporarily ignore the integrity formula, and rely on later clean-up steps to rectify matters.

Another step can be taken in this direction, which is to eliminate the use of 3-valued structures, and directly carry out fixed-point computations over logical formulas.

We are also investigating the feasibility of using the results from this paper to develop a more precise and modular version of TVLA by using *assume-guarantee* reasoning [62]. The idea is to allow arbitrary first-order formulas with transitive closure to be used to express pre- and post-conditions, and to analyze the code for each procedure separately.

References

1. L. O. Andersen. Binding-time analysis and the taming of C pointers. In David Schmidt, editor, *Proc. of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'93*, pages 47–58, New York, NY, 1993. ACM Press.
2. M. Benedikt, T. Reps, and M. Sagiv. A decidable logic for describing linked data structures. In *Proceedings of the 1999 European Symposium On Programming*, pages 2–19, March 1999.
3. S. Blamey. Partial logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Phil. Logic, 2nd. Ed., Vol. 5*, pages 261–353. Kluwer Academic Publishers, 2002.
4. G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In *Proc. CONCUR*, pages 168–182. Springer-Verlag, 2000.
5. D.R. Chase, M. Wegman, and F. Zadeck. Analysis of pointers and structures. In *SIGPLAN Conf. on Prog. Lang. Design and Impl.*, pages 296–310, New York, NY, 1990. ACM Press.
6. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. Computer-Aided Verif.*, pages 154–169, July 2000.
7. E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *Trans. on Prog. Lang. and Syst.*, 16(5):1512–1542, 1994.
8. B. Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. In N. Immerman and P.G. Kolaitis, editors, *Descriptive Complexity and Finite Models: Proceedings of a DIAMCS Workshop*, chapter 2, pages 33–57. American Mathematical Society, 1996.
9. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *Symp. on Princ. of Prog. Lang.*, pages 238–252, New York, NY, 1977. ACM Press.
10. D. Dams. *Abstract Interpretation and Partial Refinement for Model Checking*. PhD thesis, Technical Univ. of Eindhoven, Eindhoven, The Netherlands, July 1996.

11. M. Das. Unification-based pointer analysis with directional assignments. In *Conf. on Prog. Lang. Design and Impl.*, pages 35–46, 2000.
12. G. Erez. Generating concrete counter examples for arbitrary abstract domains. Master's thesis, Tel-Aviv University, Tel-Aviv, Israel, 2004. In Preparation.
13. G. Erez, M. Sagiv, and E. Yahav. Generating concrete counter examples for arbitrary abstract domains. Unpublished Manuscript, 2003.
14. R. Fagin. Monadic generalized spectra. *Z. Math. Logik*, 21:89–96, 1975.
15. M. Fähndrich, J. Foster, Z. Su, and A. Aiken. Partial online cycle elimination in inclusion constraint graphs. In *SIGPLAN Conf. on Prog. Lang. Design and Impl.*, pages 85–96, New York, NY, June 1998. ACM Press.
16. P. Fradet and D. Le Metayer. Shape types. In *Symp. on Princ. of Prog. Lang.*, pages 27–39, New York, NY, 1997. ACM Press.
17. P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *VMCAI*, pages 206–222, 2003.
18. N. Heintze and O. Tardieu. Ultra-fast aliasing analysis using CLA: A million lines of C code in a second. In *SIGPLAN Conf. on Prog. Lang. Design and Impl.*, New York, NY, June 2001. ACM Press.
19. P. Hell and J. Nesetril. *Graphs and Homomorphisms*. Oxford University Press, 2004.
20. L. Hendren. *Parallelizing Programs with Recursive Data Structures*. PhD thesis, Cornell Univ., Ithaca, NY, Jan 1990.
21. L. Hendren, J. Hummel, and A. Nicolau. Abstractions for recursive pointer data structures: Improving the analysis and the transformation of imperative programs. In *SIGPLAN Conf. on Prog. Lang. Design and Impl.*, pages 249–260, New York, NY, June 1992. ACM Press.
22. L. Hendren and A. Nicolau. Parallelizing programs with recursive data structures. *IEEE Trans. on Par. and Dist. Syst.*, 1(1):35–47, January 1990.
23. J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Proc. of TACAS 95*, pages 89–110, 1996.
24. M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP*, pages 155–169, 2001.
25. N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.
26. N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, 2004.
27. N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. Verification via structure simulation. In *CAV*, 2004.
28. N.D. Jones and S.S. Muchnick. Flow analysis and optimization of Lisp-like structures. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 4, pages 102–131. Prentice-Hall, Englewood Cliffs, NJ, 1981.
29. N.D. Jones and S.S. Muchnick. A flexible approach to interprocedural data flow analysis and programs with recursive data structures. In *Symp. on Princ. of Prog. Lang.*, pages 66–74, New York, NY, 1982. ACM Press.
30. N. Klarlund and M. Schwartzbach. Graph types. In *Symp. on Princ. of Prog. Lang.*, pages 196–205, New York, NY, January 1993. ACM Press.
31. V. Kuncak, P. Lam, and M. C. Rinard. Role analysis. In *POPL*, pages 17–32, 2002.
32. V. Kuncak and M. Rinard. Boolean algebra of shape analysis constraints. In *VMCAI*, pages 59–72, 2003.
33. V. Kuncak and M. Rinard. On Boolean algebra of shape analysis constraints. Technical report, MIT, CSAIL, 2003. Available at "http://www.mit.edu/$\sim$ vkuncak/papers/index.html".
34. P. Lam, V. Kuncak, and M. Rinard. Hob: A tool for verifying data structure consistency. In *Conf. on Compiler Construction (tool demo)*, 2005.
35. T. Lev-Ami. TVLA: A framework for Kleene based static analysis. Master's thesis, Tel-Aviv University, Tel-Aviv, Israel, 2000.

36. T. Lev-Ami, N. Immerman, T. Reps, M. Sagiv, S. Srivastava, and G. Yorsh. Simulating reachability using first-order logic with applications to verifciation of linked data structures. Submitted for publication, 2005.
37. T. Lev-Ami, T. Reps, M. Sagiv, and R. Wilhelm. Putting static analysis to work for verification: A case study. In *Proc. of the Int. Symp. on Software Testing and Analysis*, pages 26–38, 2000.
38. T. Lev-Ami and M. Sagiv. TVLA: A system for implementing static analyses. In *Static Analysis Symp.*, pages 280–301, 2000.
39. W. McCune. Mace 2.0 reference manual and guide. Available at "http://www-unix.mcs.anl.gov/AR/mace/", 2001.
40. K. L. McMillan. Verification of infinite state systems by compositional model checking. In *CHARME*, pages 219–234, 1999.
41. Albert R. Meyer. Weak monadic second-order theory of successor is not elementary recursive. In *Logic Colloquium, (Proc. Symposium on Logic, Boston, 1972)*, pages 132–154, 1975.
42. A. Møller and M.I. Schwartzbach. The pointer assertion logic engine. In *SIGPLAN Conf. on Prog. Lang. Design and Impl.*, pages 221–231, 2001.
43. M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math*, 21:135–140, 1975.
44. F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.
45. F. Nielson, H.R. Nielson, and M. Sagiv. A Kleene analysis of mobile ambients. In G. Smolka, editor, *Proc. of ESOP 2000*, volume 1782 of *LNCS*, pages 305–319. Springer, 2000.
46. M. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.
47. G. Ramalingam, A. Varshavsky, J. Field, D. Goyal, and M. Sagiv. Deriving specialized program analyses for certifying component-client conformance. In *PLDI*, pages 83–94, 2002.
48. T. Reps, M. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In *VMCAI*, pages 252–266, 2004.
49. M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *Trans. on Prog. Lang. and Syst.*, 20(1):1–50, January 1998.
50. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *Symp. on Princ. of Prog. Lang.*, pages 105–118, New York, NY, January 1999. ACM Press.
51. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *Trans. on Prog. Lang. and Syst.*, 2002.
52. R. Shaham, E. Yahav, E.K. Kolodner, and Mooly Sagiv. Establishing local temporal heap safety properties with applications to compile-time memory management. In *Proc. of Static Analysis Symposium (SAS'03)*, volume 2694 of *LNCS*, pages 483–503. Springer, June 2003.
53. M. Shapiro and S. Horwitz. Fast and accurate flow-insensitive points-to analysis. In *Symp. on Princ. of Prog. Lang.*, pages 1–14, 1997.
54. B. Steensgaard. Points-to analysis in almost-linear time. In *Symp. on Princ. of Prog. Lang.*, pages 32–41, 1996.
55. Z. Su, M. Fähndrich, and A. Aiken. Projection merging: Reducing redundancies in inclusion constraint graphs. In T. Reps, editor, *Symp. on Princ. of Prog. Lang.*, pages 81–95, New York, NY, January 2000. ACM Press.
56. B. van Fraassen. Singular terms, truth-value gaps, and free logic. *J. Phil*, 63(17):481–495, 1966.
57. C. Weidenbach. SPASS: An automated theorem prover for first-order logic with equality. Available at "http://spass.mpi-sb.mpg.de/index.html".
58. E. Yahav. Verifying safety properties of concurrent Java programs using 3-valued logic. *Symp. on Princ. of Prog. Lang.*, 36(3):27–40, 2001.

59. E. Yahav and G. Ramalingam. Verifying safety properties using separation and heterogeneous abstractions. In *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, pages 25–34. ACM Press, 2004.
60. E. Yahav and M. Sagiv. Automatically verifying concurrent queue algorithms. In Byron Cook, Scott Stoller, and Willem Visser, editors, *Electronic Notes in Theoretical Computer Science*, volume 89. Elsevier, 2003.
61. G. Yorsh. Logical characterizations of heap abstractions. Master's thesis, Tel-Aviv University, Tel-Aviv, Israel, 2003. Available at "http://www.math.tau.ac.il/∼ gretay".
62. G. Yorsh, T. W. Reps, and M. Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *TACAS*, pages 530–545, 2004.

A Characterizing Canonical Abstraction by First-Order Formulas

This section defines an alternative abstract domain for use in shape analysis (and other logic-based analyses). This domain keeps more explicit information than the one in Section 2.4 and enjoys nice closure properties (see Section A.4). This domain uses a particular class of embedding functions that are defined by a simple operation, called *canonical abstraction*, which maps 2-valued structures into a limited subset of bounded structures.

A.1 Canonical Abstraction

Canonical abstraction was defined in [50] as an abstraction with the following properties:

– It provides a uniform way to obtain 3-valued structures of a priori bounded size. This is important to automatically derive properties of programs with loops by employing iterative fixed-point algorithms. Canonical abstraction maps concrete nodes into abstract nodes according to the definite values of the unary predicates.
– The information loss is minimized when multiple nodes of S are mapped to the same node in S',

This is formalized by the following definition:

Definition 13. *A structure $S' = \langle U^{S'}, \iota^{S'} \rangle$ is a* **canonical abstraction** *of a structure S, if $S \sqsubseteq^{canonical} S'$, where $canonical\colon U^S \to U^{S'}$ is the following surjective mapping:*

$$canonical(u) = u_{\{p\in\mathcal{P}_1 \mid \iota^S(p)(u)=1\},\{p\in\mathcal{P}_1 \mid \iota^S(p)(u)=0\}} \tag{11}$$

and, for every $p \in \mathcal{P}_k$ of arity k,

$$\iota^{S'}(p)(u'_1,\ldots,u'_k) = \bigsqcup_{\substack{u_i \in U^S,\text{ s.t.}\\ canonical(u_i) = u'_i \in U^{S'},\\ 1\le i\le k}} \iota^S(p)(u_1,\ldots,u_k) \tag{12}$$

We say that $S' = canonical(S)$.

The name "$u_{\{p\in\mathcal{P}_1 \mid \iota^S(p)(u)=1\},\{p\in\mathcal{P}_1 \mid \iota^S(p)(u)=0\}}$" is known as the **canonical name** of node u. The subscript on the canonical name of u involves two sets of unary predicate symbols: (i) those that are true at u, and (ii) those that are false at u.

Example A1 *In structure S from Fig. 2, the canonical names of the nodes are as follows:*

Node	Canonical Name
u_1	$u_{\{x,r_x\},\{y,t,e,is,r_y,r_t,r_e\}}$
u_2	$u_{\{r_x\},\{x,y,t,e,is,r_y,r_t,r_e\}}$

In the context of canonical abstraction, S shown in Fig. 2 represents S_b and S_c, but not S_a; i.e., S represents lists that are pointed to by x that have at least three nodes, but it does not represent a list with just two nodes. The reason is that predicates n and eq have indefinite values in S, but a list with only two nodes cannot have both 0 and 1 values for the corresponding entries, as required for minimizing information loss as defined in Eq. (12).[12] *In contrast, according to the abstraction that relies on embedding, defined in Section 2.4, S represents lists with two or more elements.*

To characterize canonical abstraction, we define the set of 3-valued structures that are "images of canonical abstraction" (*ICA*), i.e., the results of applying canonical abstraction to 2-valued structures.

Definition 14. Image of canonical abstraction (*ICA*) *Structure S is an* ICA *if there exists a 2-valued structure $S^\natural$ such that S is the canonical abstraction of $S^\natural$.*

Concretization of 3-Valued Structures. Canonical abstraction allows us to define the (potentially infinite) set of 2-valued structures represented by a set of 3-valued structures, that are *ICA*

Definition 15. Concretization of ICA Structures *For a set of structures $X \subseteq$ 3-STRUCT$[\mathcal{P}]$, that are* ICA *structures, we denote by $\gamma_c(X)$ the set of 2-valued structures that X represents, i.e.,*

$$\gamma_c(X) = \left\{ \begin{array}{l} S^\natural \in \textit{2-STRUCT}[\mathcal{P}] \mid \textit{exists } S \in X \textit{such that} \\ S \textit{ is the canonical abstraction of } S^\natural \textit{ and } S^\natural \models F \end{array} \right\} \tag{13}$$

Also, for a singleton set $X = \{S\}$ we write $\gamma_c(S)$ instead of $\gamma_c(X)$.

The abstract domain is the powerset of *ICA* structures, where the order relation is set inclusion. Note that this abstract domain is finite, because there is a finite number of different ICA structures (up to isomorphism). Denote by α_c the extension of the abstraction function $canonical$ to sets. This defines a Galois connection $\langle \alpha_c, \gamma_c \rangle$ between sets of 2-valued structures and sets of *ICA* structures.

A.2 Canonical-FO-Identifiable Structures

We define the notion of canonical-FO-identifiable nodes using canonical abstraction rather than embedding, which was used for the notion of FO-identifiable nodes in Definition 8.

Definition 16. *We say that a node u in a 3-valued structure S is* **canonical-FO-identifiable** *if there exists a formula node$^S_u(w)$ with designated free variable w, such that for every*

[12] Eq. (12) is called the *tight-embedding* condition in [51].

2-valued structure $S^\natural$, if S is the canonical abstraction of $S^\natural$, i.e., $S^\natural \in \gamma_c(S)$, then for every concrete node $u^\natural \in U^{S^\natural}$:

$$canonical(u^\natural) = u \iff S^\natural, [w \mapsto u^\natural] \models \mathit{node}^S_u(w) \tag{14}$$

S is called canonical-FO-identifiable if all the nodes in S are canonical-FO-identifiable.

We can also prove Lemma 1 for the case of canonical abstraction rather than embedding.

A.3 Characterizing Canonical Abstraction

An *ICA* structure is always a bounded structure, in which all nullary and unary predicates have definite values.[13] This is formalized by the following lemma:

Lemma 4. *If 3-valued structure $S = \langle U^S, \iota^S \rangle$ over vocabulary $\mathcal{P}$ is* ICA *then:*
(i) *S is a bounded structure.*
(ii) *For each nullary predicate p, $\iota^S(p)() \in \{0, 1\}$.*
(iii) *For each element $u \in U$ and each unary predicate p, $\iota^S(p)(u) \in \{0, 1\}$.*

The following lemma shows that *ICA* structures are canonical-FO-identifiable:

Lemma 5. *Every 3-valued structure S that is an* ICA *is canonical-FO-identifiable, where*

$$\mathit{node}^S_{u_i}(w) \overset{\text{def}}{=} \bigwedge_{p \in \mathcal{P}_1} p^{\iota^S(p)(u_i)}(w) \tag{15}$$

Using this fact, we can define a formula τ^S that accepts exactly the set of 2-valued structures represented by S under canonical abstraction. The formula τ^S is merely ξ^S with additional conjuncts to ensure that the information loss is minimized, i.e., for every predicate p and every $1/2$ entry of p, the 2-valued structure has both a corresponding 1 entry and a corresponding 0 entry.

Definition 17. First-Order Characteristic Formula for Canonical Abstraction *Let 3-valued structure $S = \langle U^S, \iota \rangle$ be an ICA.*

For a predicate p of arity r, we define the closed formula for p:

$$\tau^S[p] \overset{\text{def}}{=} \bigwedge_{\substack{\{u'_1, \ldots, u'_r\} \subseteq U^S \\ \text{s.t. } \iota^S(p)(u'_1, \ldots, u'_r) = 1/2}} \begin{pmatrix} \exists w_1, \ldots, w_r : \bigwedge_{j=1}^r \mathit{node}^S_{u'_j}(w_j) \land p(w_1, \ldots, w_r) \\ \land\ \exists w_1, \ldots, w_r : \bigwedge_{j=1}^r \mathit{node}^S_{u'_j}(w_j) \land \neg p(w_1, \ldots, w_r) \end{pmatrix} \tag{16}$$

The formula of S is defined by:

$$\tau^S \overset{\text{def}}{=} \xi^S \land \bigwedge_{r=2}^{maxR} \bigwedge_{p \in \mathcal{P}_r} \tau^S[p] \tag{17}$$

[13] If not all unary predicates are defined as abstraction predicates, then the result may be a bounded structure of the less restrictive kind mentioned in Section 3.1. Also, unary predicates that are not abstraction predicates may have indefinite values.

The **characteristic formula for canonical abstraction of a set of *ICA* structures** $X \subseteq$ *3-STRUCT*$[\mathcal{P}]$ *is defined by*

$$\widehat{\gamma}_c(X) = F \wedge (\bigvee_{S \in X} \tau^S) \tag{18}$$

Also, for a singleton set $X = \{S\}$*, where* S *is an* ICA *structure, we write* $\widehat{\gamma}_c(S)$ *instead of* $\widehat{\gamma}_c(X)$*.*

Example A2 *The characteristic formula for canonical abstraction of the structure* S *shown in Fig. 2(d) is:*

$$\begin{aligned}
\widehat{\gamma}_c(S) = {} & \widehat{\gamma}(S) \\
& \wedge \exists w_1, w_2 : \textit{node}^S_{u_1}(w_1) \wedge \textit{node}^S_{u_2}(w_2) \wedge n(w_1, w_2) \\
& \wedge \exists w_1, w_2 : \textit{node}^S_{u_1}(w_1) \wedge \textit{node}^S_{u_2}(w_2) \wedge \neg n(w_1, w_2) \\
& \wedge \exists w_1, w_2 : \textit{node}^S_{u_2}(w_1) \wedge \textit{node}^S_{u_2}(w_2) \wedge n(w_1, w_2) \\
& \wedge \exists w_1, w_2 : \textit{node}^S_{u_2}(w_1) \wedge \textit{node}^S_{u_2}(w_2) \wedge \neg n(w_1, w_2) \\
& \wedge \exists w_1, w_2 : \textit{node}^S_{u_2}(w_1) \wedge \textit{node}^S_{u_2}(w_2) \wedge eq(w_1, w_2) \\
& \wedge \exists w_1, w_2 : \textit{node}^S_{u_2}(w_1) \wedge \textit{node}^S_{u_2}(w_2) \wedge \neg eq(w_1, w_2)
\end{aligned} \tag{19}$$

where $\widehat{\gamma}(S)$ *is given in Example 33. As explained in Example A1,* S *does not represent a list of two nodes; the corresponding* 2*-valued structure* S_a*, shown in Fig. 2(a), does not satisfy Eq. (19), because the last four lines cannot be satisfied by any assignment in* S_a*.*

Remark. The formula τ^S does not contain quantifier alternation and transitive closure. Therefore, $\widehat{\gamma}_c$ is in Existential-Universal normal form (and thus decidable) whenever F is in Existential-Universal form and does not contain transitive closure.

Theorem 2. *For every* 3*-valued structure* S *that is an* ICA *and* 2*-valued structure* $S^\natural$

$$S^\natural \in \gamma_c(S) \textit{ iff } S^\natural \models \widehat{\gamma}_c(S)$$

A.4 Closure Properties of ICA Structures

This section gives a simple semantic proof that the class of formulas that characterize ICA structures is closed under negation. This result was shown in [33] using a different formalism.

From Eq. (12) it follows that for two distinct ICA structures S_1 and S_2, $\gamma_c(S_1) \cap \gamma_c(S_2) = \varnothing$. Intuitively, each 2-valued structure can be represented by exactly one ICA structure. This implies that the complement of the concretization of an ICA structure can be represented precisely by a finite set of ICA structures.

Denote by $\mathcal{D}$ the set of all 2-valued structures that satisfy the integrity formula F: $\mathcal{D} \stackrel{\text{def}}{=} \{S^\natural \in \text{2-STRUCT}[\mathcal{P}] \mid S^\natural \models F\}$.

Lemma 6. *Let* S *be an ICA structure. There exists a set of ICA structures* X *such that* $\gamma_c(X) = \mathcal{D} \setminus \gamma_c(S)$*.*

This can be reformulated using Theorem 2 in terms of characteristic formulas for ICA structures. This shows that the class of formulas that characterize canonical abstraction is closed under negation, in the following sense:

Lemma 7. *Consider the formula τ^S from Eq. (17), for some ICA structure S. There exists a set of ICA structures X, such that the formula $F \land \neg\tau^S$ is equivalent to the formula $\widehat{\gamma}_c(X)$.*

Remark. Note that Lemma 6 and Lemma 7 do not hold for bounded structures using γ, described in Section 3.1, instead of γ_c. The reason, intuitively, is that some 2-valued structures can be represented by more than one bounded structure.

For example, consider the 2-valued structure S_a from Fig. 2, which denotes a linked-list of length exactly 2. It is in the concretization of two different 3-valued structures: the first is S_a itself, considered as a 3-valued structure S' (that represents a single 2-valued structure: $\gamma(S') = \{S_a\}$); the second is the structure S from Fig. 2.

For the purpose of this example, assume that the integrity formula F (that defines $\mathcal{D}$) requires that all elements be reachable from x, in addition to the integrity formula F_{List} from Example 22. The complement $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{D} \setminus \gamma(S') = \mathcal{D} \setminus S_a$ is the set that contains an empty linked list, a linked list of length 1, and linked lists of length 3 or more. The representation of $\mathcal{C}$ is a set X of bounded structures. To capture linked lists of length 3 or more, X must contain a 3-valued structure S from Fig. 2. However, $\gamma(S)$ includes a list of length 2 as well, denoted by S_a, which is not in $\mathcal{C}$. Therefore, X cannot contain S, and a contradiction is obtained.

B Characterizing General 3-Valued Structures by NP Formulas

In this section, we show how to characterize general 3-valued structures.

B.1 Motivating Example

If the input structure is FO-identifiable, Theorem 1 ensures that the result of operation $\widehat{\gamma}$ precisely captures the concretization of the input structure. The purpose of this example is to show what happens if we apply the $\widehat{\gamma}$ operation, as defined in Section 3, to a structure that is not FO-Identifiable. When S is not FO-identifiable, $\widehat{\gamma}(S)$ only provides a sufficient test for the embedding of 2-valued structures into S.

Example B1 *The 3-valued structure S shown in Fig. 3 describes undirected graphs. We draw undirected edges as two-way directed edges. This structure uses a set of predicates $\mathcal{P} = \{eq, f, b\}$, where $f(v_1, v_2)$ and $b(v_2, v_1)$ denote the forward and backward directions of an edge between nodes v_1 and v_2.*

When Eq. (8) is applied to the 3-valued structure S shown in Fig. 3, we get

$$\begin{array}{ll} \bigwedge_{i=1}^{3} & \exists v : \mathit{node}^S_{u_i}(v) \\ \land & \forall w : \bigvee_{i=1}^{3} \mathit{node}^S_{u_i}(w) \\ \land & \forall w_1, w_2 : \bigwedge_{k \neq j} (\mathit{node}^S_{u_k}(w_1) \land \mathit{node}^S_{u_j}(w_2) \Rightarrow f^{1/2}(w_1, w_2)) \\ \land & \forall w_1, w_2 : \bigwedge_{k \neq j} (\mathit{node}^S_{u_k}(w_1) \land \mathit{node}^S_{u_j}(w_2) \Rightarrow b^{1/2}(w_1, w_2)) \\ \land & \forall w_1, w_2 : \bigwedge_{i=1}^{3} (\mathit{node}^S_{u_i}(w_1) \land \mathit{node}^S_{u_i}(w_2) \Rightarrow b^{0}(w_1, w_2)) \\ \land & \forall w_1, w_2 : \bigwedge_{i=1}^{3} (\mathit{node}^S_{u_i}(w_1) \land \mathit{node}^S_{u_i}(w_2) \Rightarrow f^{0}(w_1, w_2)) \end{array} \quad (20)$$

Because this example does not include unary predicates, the node *formula given in Lemma 3 evaluates to* $\mathbf{1}$ *on all elements. Hence, Eq. (20) can be simplified to:*

$$\begin{array}{ll}\bigwedge_{i=1}^{3} \exists v : \mathbf{1} & \\ \wedge & \forall w : \bigvee_{i=1}^{3} \mathbf{1} \\ \wedge & \forall w_1, w_2 : \bigwedge_{k \neq j} (\mathbf{1} \wedge \mathbf{1} \Rightarrow \mathbf{1}) \\ \wedge & \forall w_1, w_2 : \bigwedge_{k \neq j} (\mathbf{1} \wedge \mathbf{1} \Rightarrow \mathbf{1}) \\ \wedge & \forall w_1, w_2 : \bigwedge_{i=1}^{3} (\mathbf{1} \wedge \mathbf{1} \Rightarrow \neg b(w_1, w_2)) \\ \wedge & \forall w_1, w_2 : \bigwedge_{i=1}^{3} (\mathbf{1} \wedge \mathbf{1} \Rightarrow \neg f(w_1, w_2))\end{array}$$

After further simplification, we get the formula $\forall w_1, w_2 : \neg f(w_1, w_2) \wedge \forall w_1, w_2 : \neg b(w_1, w_2)$*. The simplification is due to the fact that the implication in Eq. (7) unconditionally holds for all pairs of distinct nodes, because* f *and* b *evaluate to* $1/2$ *on those pairs, except for the requirement imposed by the absence of self-loops in* S*.*

This formula is only fulfilled by graphs with no edges, which are obviously 3*-colorable. But this formula is too restrictive: it does not capture some* 3*-colorable graphs.*

B.2 Characterizing General 3-Valued Structures

Existential monadic second-order formulas are a subset of Fagin's second-order formulas [14], named NP formulas, which capture NP computations. A formula in existential monadic second-order logic has the form:

$$\exists V_1, V_2, \ldots, V_n : \varphi$$

where the V_i are set variables, and φ is a first-order formula that can use membership tests in V_i. We show that in this subset of second-order logic, the characteristic formula from Definition 11 can be generalized to handle arbitrary 3-valued structures using existential quantification over set variables (with one set variable for each abstract node).[14]

Definition 18. NP Characteristic Formula *Let* $S = \langle U = \{u_1, u_2, \ldots, u_n\}, \iota \rangle$ *be a* 3*-valued structure.*

We define the following formula to ensure that the sets are non_empty:

$$\xi^S_{non\_empty}[i] \stackrel{\text{def}}{=} \exists w_i : \mathit{node}^S_{u_i}(w_i) \tag{21}$$

We define the following formula to ensure that the sets V_k*,* V_j *are disjoint:*

$$\xi^S_{disjoint}[k, j] \stackrel{\text{def}}{=} \forall w_1, w_2 : \mathit{node}^S_{u_k}(w_1) \wedge \mathit{node}^S_{u_j}(w_2) \Rightarrow \neg eq(w_1, w_2) \tag{22}$$

The **NP characteristic formula of** S *is defined by:*

$$\begin{array}{rl}\xi^S \stackrel{\text{def}}{=} \exists V_1, \ldots, V_n : & \bigwedge_{i=1}^{n} \xi^S_{non\_empty}[i] \wedge \bigwedge_{k \neq j} \xi^S_{disjoint}[k, j] \\ & \wedge\ \xi^S_{total} \\ & \wedge\ \xi^S_{nullary} \\ & \wedge\ \bigwedge_{r=1}^{maxR} \bigwedge_{p \in \mathcal{P}_r} \xi^S[p]\end{array} \tag{23}$$

[14] This result is mostly theoretical. In principle, this encoding falls into monadic-second order logic, which is decidable if we restrict the concrete structures of interest to trees. However, we have not investigated this direction further.

where ξ^S_{total}, $\xi^S_{nullary}$, $\xi^S[p]$ are defined as in Definition 11, except that node$^S_{u_i}$ is the NP formula node$^S_{u_i}(w) \stackrel{\text{def}}{=} (w \in V_i)$. (Here, we abuse notation slightly by referring to V_i in node$^S_{u_i}(w)$. This could have been formalized by passing $V_1, \ldots, V_n$ as extra parameters to node$^S_{u_i}$.)

The **NP characteristic formula of a finite set** $X \subseteq$ **3-STRUCT**$[\mathcal{P}]$ *is defined by:*

$$\widehat{\gamma}_{NP}(X) = F \wedge (\bigvee_{S \in X} \xi^S) \tag{24}$$

Finally, for a singleton set $X = \{S\}$ we write $\widehat{\gamma}_{NP}(S)$ instead of $\widehat{\gamma}_{NP}(X)$.

Example B2 *After a small amount of simplification, the NP characteristic formula ξ^S for the graph shown in Fig. 3 is:*

$$\begin{array}{lll}
\exists V_1, V_2, V_3 : & \bigwedge_{i=1}^3 (\exists w : w \in V_i) & (i) \\
& \wedge\ \bigwedge_{k \neq j} \forall w_1, w_2 : (w_1 \in V_k \wedge w_2 \in V_j \Rightarrow \neg eq(w_1, w_2)) & (ii) \\
& \wedge\ \forall w : \bigvee_{i=1}^3 w \in V_i & (iii) \\
& \wedge\ \forall w_1, w_2 : \bigwedge_{i=1}^3 (\bigwedge_{j=1,2} w_j \in V_i \Rightarrow \neg e(w_1, w_2) \wedge \neg e(w_2, w_1)) & (iv)
\end{array}$$

In this formula, V_1, V_2, and V_3 represent the three color classes. Line by line, the formula says: (i) each color class has at least one member; (ii) the color classes are pairwise disjoint; (iii) every node is in a color class; (iv) nodes in the same color class are not connected by an undirected edge.

The following theorem generalizes the result in Theorem 1 for an arbitrary 3-valued structure S, using NP-formula $\widehat{\gamma}_{NP}(S)$ to accept exactly the set of 2-valued structures represented by S.

Theorem 3. *For every 3-valued structure S, and 2-valued structure $S^\natural$:*

$$S^\natural \in \gamma(S) \text{ iff } S^\natural \models \widehat{\gamma}_{NP}(S)$$

C Generating and Querying a Loop Invariant

Table 2 and Table 3 show the structures and the characteristic formulas for the experiment described in Example 52.

It is interesting to note that the size of ξ^{S_2} is bigger than the size of ξ^{S_1}. This is natural because S_2 has more definite values, which impose more restrictions than are imposed by S_1.

D Proofs

Lemma D1 *Consider the 3-valued structure S shown in Fig. 3. For all 2-valued structures C, C can be embedded into S if and only if C can be colored using 3 colors.*
Proof of the if direction: Suppose that C is 3-colorable, let c be a mapping from the nodes of C to the colors $\{1, 2, 3\}$. We define embedding function f from C to S as follows: $f(u) = u_{c(u)}$, i.e., a node $u \in C$ that has color i is mapped to $u_i \in S$. It is easy to see that f preserves predicate values in S, because the only definite values in S

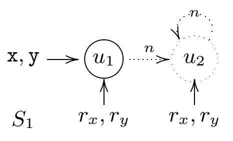
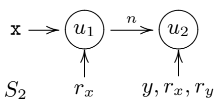
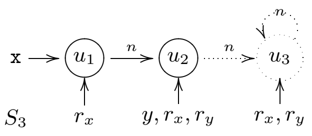
Structure	**CharacteristicFormula**
 x, y u_1 n u_2 n r_x, r_y r_x, r_y S_1	$\text{node}^{S_1}_{u_1}(w) = x(w) \wedge y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\text{node}^{S_1}_{u_2}(w) = \neg x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\xi^{S_1} = \bigwedge_{i=1,2}(\exists v : \text{node}^{S_1}_{u_i}(v))$ $\wedge\ \forall w : \bigvee_{i=1,2} \text{node}^{S_1}_{u_i}(w)$ $\wedge\ \forall w_1, w_2 : \bigwedge_{i=1,2} \text{node}^{S_1}_{u_i}(w_i) \Rightarrow$ $\neg eq(w_1, w_2) \wedge \neg n(w_2, w_1)$ $\wedge\ \forall w_1, w_2 : \bigwedge_{i=1,2} \text{node}^{S_1}_{u_1}(w_i) \Rightarrow$ $\wedge eq(w_1, w_2) \wedge \neg n(w_1, w_2)$
 x u_1 n u_2 r_x y, r_x, r_y S_2	$\text{node}^{S_2}_{u_1}(w) = x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\text{node}^{S_2}_{u_2}(w) = \neg x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\xi^{S_2} = \bigwedge_{i=1,2}(\exists v : \text{node}^{S_2}_{u_i}(v))$ $\wedge\ \forall w : \bigvee_{i=1,2} \text{node}^{S_2}_{u_i}(w)$ $\wedge\ \forall w_1, w_2 : \bigwedge_{i=1,2} \text{node}^{S_1}_{u_i}(w_i) \Rightarrow$ $\neg eq(w_1, w_2) \wedge \neg n(w_2, w_1) \wedge n(w1, w2)$ $\wedge\ \forall w_1, w_2 : \bigwedge_{i=1,2} \text{node}^{S_1}_{u_1}(w_i) \Rightarrow$ $\wedge eq(w_1, w_2) \wedge \neg n(w_1, w_2)$ $\wedge\ \forall w_1, w_2 : \bigwedge_{i=1,2} \text{node}^{S_1}_{u_2}(w_i) \Rightarrow$ $\wedge eq(w_1, w_2) \wedge \neg n(w_1, w_2)$
 x u_1 n u_2 n u_3 n r_x y, r_x, r_y r_x, r_y S_3	$\text{node}^{S_3}_{u_1}(w) = x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge \neg r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\text{node}^{S_3}_{u_2}(w) = \neg x(w) \wedge y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\text{node}^{S_3}_{u_3}(w) = \neg x(w) \wedge \neg y(w) \wedge \neg t(w) \wedge \neg e(w)$ $\wedge\ r_x(w) \wedge r_y(w) \wedge \neg r_t(w) \wedge \neg r_e(w) \wedge \neg is(w)$ $\xi^{S_3} = \bigwedge_{i=1,2,3}(\exists v : \text{node}^{S_3}_{u_i}(v))$ $\wedge\ \forall w : \bigvee_{i=1,2,3} \text{node}^{S_3}_{u_i}(w)$ $\wedge\ \forall w_1, w_2 : (\bigwedge_{i=1,2} \text{node}^{S_3}_{u_1}(w_i) \Rightarrow$ $eq(w_1, w_2) \wedge \neg n(w_1, w_2))$ $\wedge\ (\bigwedge_{i=1,2} \text{node}^{S_3}_{u_2}(w_i) \Rightarrow$ $eq(w_1, w_2) \wedge \neg n(w_1, w_2))$ $\wedge\ (\text{node}^{S_3}_{u_1}(w_1) \wedge \text{node}^{S_3}_{u_2}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \wedge \neg n(w_2, w_1) \wedge n(w_1, w_2))$ $\wedge\ (\text{node}^{S_3}_{u_2}(w_1) \wedge \text{node}^{S_3}_{u_3}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \wedge \neg n(w_2, w_1))$ $\wedge\ (\text{node}^{S_3}_{u_1}(w_1) \wedge \text{node}^{S_3}_{u_3}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \wedge \neg n(w_2, w_1) \wedge \neg n(w_1, w_2))$

Table 2. (Continued in Table 3.) The left column shows the structures that arise at the beginning of the loop in the `insert` program from Fig. 1(b). The right column shows the characteristic formula for each structure. Note that we omit the redundant sub-formulas $\xi^S[p]$, for $p \in \mathcal{P}_1$, that are part of ξ^S_{total} and $\text{node}^{S_i}_{u_j}(w)$ definitions.

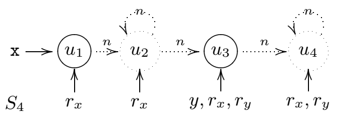
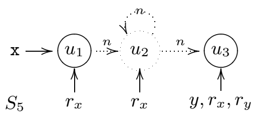
Structure	**CharacteristicFormula**
	$\text{node}^{S_4}_{u_1}(w) = x(w) \land \neg y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land \neg r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\text{node}^{S_4}_{u_1}(w) = \neg x(w) \land \neg y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land \neg r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\text{node}^{S_4}_{u_3}(w) = \neg x(w) \land y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\text{node}^{S_4}_{u_4}(w) = \neg x(w) \land \neg y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\xi^{S_1} = \bigwedge_{i=1,\ldots,4}(\exists v : \text{node}^{S_4}_{u_i}(v))$ $\land\ \forall w : \bigvee_{i=1,\ldots,4} \text{node}^{S_4}_{u_i}(w)$ $\land\ \forall w_1, w_2 :$ $(\bigwedge_{i=1,2} \text{node}^{S_4}_{u_1}(w_i) \Rightarrow$ $eq(w_1, w_2) \land \neg n(w_1, w_2))$ $\land\ (\bigwedge_{i=1,2} \text{node}^{S_4}_{u_3}(w_i) \Rightarrow$ $eq(w_1, w_2) \land \neg n(w_1, w_2))$ $\land\ (\text{node}^{S_4}_{u_1}(w_1) \land \text{node}^{S_4}_{u_2}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1))$ $\land\ (\text{node}^{S_4}_{u_2}(w_1) \land \text{node}^{S_4}_{u_3}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1))$ $\land\ (\text{node}^{S_4}_{u_1}(w_1) \land \text{node}^{S_4}_{u_3}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1) \land \neg n(w_1, w_2))$ $\land\ (\text{node}^{S_4}_{u_3}(w_1) \land \text{node}^{S_4}_{u_4}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1))$ $\land\ (\text{node}^{S_4}_{u_1}(w_1) \land \text{node}^{S_4}_{u_4}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1) \land \neg n(w_1, w_2))$ $\land\ (\text{node}^{S_4}_{u_2}(w_1) \land \text{node}^{S_4}_{u_4}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1) \land \neg n(w_1, w_2))$
	$\text{node}^{S_5}_{u_1}(w) = x(w) \land \neg y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land \neg r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\text{node}^{S_5}_{u_2}(w) = \neg x(w) \land \neg y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land \neg r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\text{node}^{S_5}_{u_3}(w) = \neg x(w) \land y(w) \land \neg t(w) \land \neg e(w)$ $\land\ r_x(w) \land r_y(w) \land \neg r_t(w) \land \neg r_e(w) \land \neg is(w)$ $\xi^{S_3} = \bigwedge_{i=1,2,3}(\exists v : \text{node}^{S_5}_{u_i}(v))$ $\land\ \forall w : \bigvee_{i=1,2,3} \text{node}^{S_5}_{u_i}(w)$ $\land\ \forall w_1, w_2 :$ $(\bigwedge_{i=1,2} \text{node}^{S_5}_{u_1}(w_i) \Rightarrow$ $eq(w_1, w_2) \land \neg n(w_1, w_2))$ $\land\ (\bigwedge_{i=1,2} \text{node}^{S_5}_{u_3}(w_i) \Rightarrow$ $eq(w_1, w_2) \land \neg n(w_1, w_2))$ $\land\ (\text{node}^{S_5}_{u_1}(w_1) \land \text{node}^{S_5}_{u_2}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1))$ $\land\ (\text{node}^{S_5}_{u_2}(w_2) \land \text{node}^{S_5}_{u_3}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1))$ $\land\ (\text{node}^{S_5}_{u_1}(w_1) \land \text{node}^{S_5}_{u_3}(w_2) \Rightarrow$ $\neg eq(w_1, w_2) \land \neg n(w_2, w_1) \land \neg n(w_1, w_2))$

Table 3. Table 2 continued.

indicate the absence of self-loops. It is preserved, because there are no edges in C with both endpoints in the same color.
Proof of the only-if direction: Suppose that C is embedded into S using f. We show that C is 3-colorable. For each node $u \in C$, let the color of u, $c(u)$, be the name of the corresponding node in S, i.e., $c(u) = f(u)$. The absence of self loops on any of the three summary nodes guarantees that a pair of adjacent nodes in C cannot be mapped by f to the same summary node. That is, for any edge in C the endpoints must be mapped by f to different summary nodes, thus they have different colors.

Lemma 1 *Let S be an FO-identifiable structure and let $u_1, u_2 \in S$ be distinct individuals. Let $S^\natural$ be a 2-valued structure that embeds into S and let $u^\natural \in S^\natural$. At most one of the following can hold, but not both:*

1. $S^\natural, [w \mapsto u^\natural] \models \mathit{node}^S_{u_1}(w)$
2. $S^\natural, [w \mapsto u^\natural] \models \mathit{node}^S_{u_2}(w)$

Proof. Because $S^\natural$ embeds into S, there exists an embedding function f, such that $S^\natural \sqsubseteq^f S$. For the sake of argument, assume that both claims hold. By Definition 8, we get that $f(u^\natural) = u_1$ and $f(u^\natural) = u_2$; because f is a function, we get that $u_1 = u_2$. This yields a contradiction to the assumption that u_1 and u_2 are distinct individuals.

Lemma 2 *For every* 2*-valued structure $S^\natural$ and assignment Z*

$$S^\natural, Z \models p^B(v_1, v_2, \ldots, v_k) \textit{ iff } \iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) \sqsubseteq B$$

Proof of the if direction: Suppose that $\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) \sqsubseteq B$. There are two cases to consider: (i) $B = 1/2$ or (ii) $\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) = B$. If $B = 1/2$, then by Definition 9, $p^B(v_1, v_2, \ldots, v_k) = 1$ and thus $S^\natural, Z \models p^B(v_1, v_2, \ldots, v_k)$ for all Z. If B = 1, then $\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) = 1$, thus $S^\natural, Z \models p(v_1, v_2, \ldots, v_k)$ which is $S^\natural, Z \models p^1(v_1, v_2, \ldots, v_k)$ by Definition 9. Similarly, if B = 0, then $\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) = 0$ implies that $S^\natural, Z \models \neg p(v_1, v_2, \ldots, v_k) = p^0(v_1, v_2, \ldots, v_k)$.
Proof of the only-if direction: Assume that $S^\natural, Z \models p^B(v_1, v_2, \ldots, v_k)$. If $B = 1/2$, then $\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) \sqsubseteq B$ trivially holds. If $B = 0$, apply Definition 9 to the assumption to get $S^\natural, Z \models \neg p(v_1, v_2, \ldots, v_k)$, which implies
$\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) = 0 = B$. Similarly, if $B = 1$, the assumption implies $\iota^{S^\natural}(p)(Z(v_1), Z(v_2), \ldots, Z(v_k)) = 1 = B$.
Lemma 3 *Every bounded* 3*-valued structure S is FO-identifiable, where*

$$\mathit{node}^S_{u_i}(w) \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}_1} p^{\iota^S(p)(u_i)}(w)$$

Proof: Consider a bounded 3-valued structure $S = \{U, \iota^S\}$. We shall show that every element $u \in U$ is FO-identifiable using the formula defined in Eq. (4). Let $S^\natural$ be a 2-valued structure that embeds into S using a function f, and let $u^\natural$ be a concrete element in $U^{S^\natural}$. By Definition 8, we have to show that the following holds:

$$f(u^\natural) = u \iff S^\natural, [w \mapsto u^\natural] \models \text{node}^S_u(w)$$

Proof of the if direction: Suppose that $S^\natural, [w \mapsto u^\natural] \models \text{node}_u^S(w)$. In particular, each conjunct of node_u^S must hold, i.e., for each predicate $p \in \mathcal{P}_1$, $S^\natural, [w \mapsto u^\natural] \models p^{\iota^S(p)(u)}(w)$. Using Lemma 2 we get that $\iota^{S^\natural}(p)(u^\natural) \sqsubseteq \iota^S(p)(u)$. In addition, the embedding condition in Eq. (1), requires, in particular, that for each unary predicate p $\iota^{S^\natural}(p)(u^\natural) \sqsubseteq \iota^S(p)(f(u^\natural))$ holds. Let $u_1 = f(u^\natural)$. For the sake of argument, assume that $u_1 \neq u$. Recall that S is a bounded structure, in which every individual must have a unique combination of definite values of unary predicates. As a consequence, there must be a unary predicate p such that $\iota^S(p)(u_1) \neq \iota^S(p)(u)$ and the value of p on both u_1 and u is definite. This yields a contradiction, because $\sqsubseteq$ on definite values implies equality; however $\iota^{S^\natural}(p)(u^\natural) = \iota^S(p)(u)$ and $\iota^{S^\natural}(p)(u^\natural) = \iota^S(p)(f(u^\natural)) = \iota^S(p)(u_1)$ can not hold simultaneously, by the assumption.
Proof of the only-if direction: Suppose that $f(u^\natural) = u$. Using Eq. (1), the embedding function f guarantees that for each unary predicate p, $\iota^{S^\natural}(p)(u^\natural) \sqsubseteq \iota^S(p)(f(u^\natural))$. This means that $S^\natural, [w \mapsto u^\natural] \models p^{\iota^S(p)(f(u^\natural))}(w)$ by Lemma 2, or $S^\natural, [w \mapsto u^\natural] \models p^{\iota^S(p)(u)}(w)$ by the assumption. This holds for all unary predicates, and thus holds for their conjunction as well, namely, for the formula node_u^S.
Lemma D2 *Given a set of formulas F and a 3-valued structure S, if the "focus" algorithm [35, Sec.6] terminates, it returns a set of structures X such that $\gamma(S) = \gamma(X)$ and every formula $\varphi \in F$ evaluates, using the compositional semantics, to a definite value in every structure in X, for every assignment. If the input structure S is FO-Identifiable, then all structures in X are FO-Identifiable.*
Proof: By induction on the iterations of the loop in the "focus" algorithm, it is sufficient to show that the structures returned by the procedure `FocusAssignment` from [35, Fig.17] are FO-Identifiable. The only interesting case is when the input literal of `FocusAssignment` is of the form $p(u_1, \ldots, u_k)$. The resulting set of structures X is $\{S_0, S_1, S''\}$ where S_0 and S_1 are copies of S with $p(u_1, \ldots, u_k)$ set to 0 and 1, respectively. Thus, if S is FO-identifiable, then S_0 and S_1 are FO-identifiable. S'' is a result of splitting a node $u_i \in S$ into $u.0$ and $u.1$, and setting $p(u_1, \ldots, u_k)$ to 0 on one of the copies, and to 1 on the other. To simplify the exposition, suppose that the first node u_1 is split. Then S'' is FO-identifiable using the formulas $\text{node}_u^S(w)$ for all u except $u.0, u.1$, and

$$\text{node}_{u.0}^{S''}(w) \stackrel{\text{def}}{=} \exists v_2, \ldots, v_k. \neg p(w, v_2, \ldots, v_k) \wedge \text{node}_u^S(w) \wedge \bigwedge_{j=2,\ldots,k} \text{node}_{u_j}^S(v_j)$$
$$\text{node}_{u.1}^{S''}(w) \stackrel{\text{def}}{=} \exists v_2, \ldots, v_k. p(w, v_2, \ldots, v_k) \wedge \text{node}_u^S(w) \wedge \bigwedge_{j=2,\ldots,k} \text{node}_{u_j}^S(v_j)$$

Theorem 1 *For every FO-identifiable 3-valued structure S, and 2-valued structure $S^\natural$*

$$S^\natural \in \gamma(S) \textit{ iff } S^\natural \models \widehat{\gamma}(S)$$

Proof: In Lemma D3, we show that the if-direction holds, even when S is not FO-identifiable, i.e., every concrete structure satisfying the characteristic formula $\widehat{\gamma}(S)$ is indeed in $\gamma(S)$. In Lemma D4 we show the only-if part, i.e., for an FO-identifiable structure, the other direction is also true.
Lemma D3 *Let S be a first-order structure with set of individuals $U = \{u_1, u_2, \ldots, u_n\}$. Let $\text{node}_{u_i}^S(w)$ used in $\widehat{\gamma}(S)$ be an arbitrary first-order formula free in w, such that Lemma 1 holds. Then, for all $S^\natural$ such that $S^\natural \models \widehat{\gamma}(S)$, $S^\natural \in \gamma(S)$.*

Proof: Let $S^\natural = \langle U^\natural, \iota^\natural \rangle$ be a concrete structure such that $S^\natural \models \widehat{\gamma}(S)$. We shall construct a surjective function $f\colon U^\natural \to U$ such that $S^\natural \sqsubseteq^f S$. Let $Z^\natural$ be an assignment over $v_1, \ldots, v_n$ such that $S^\natural, Z^\natural \models \varphi$, where $\varphi \stackrel{\text{def}}{=} \bigwedge_{i=1}^n \text{node}_{u_i}^S(v_i)$, i.e., φ is the first line of Eq. (8) without the existential quantification. Note that all $Z^\natural(v_i)$ are distinct, according to Lemma 1. Define the function $f\colon U^\natural \to U$ by:

$$f(u^\natural) = \begin{cases} u_i \text{ if } Z^\natural(v_i) = u^\natural \\ u_j \text{ if for all } i,\ Z^\natural(v_i) \neq u^\natural \text{ and } u_j \text{ is an arbitrary element such that} \\ \quad S^\natural, [w \mapsto u^\natural] \models \text{node}_{u_j}^S(w) \end{cases} \tag{25}$$

Let us show that every concrete element is mapped to some element in U. In the case that $Z(v_i) = u^\natural$, the concrete element $u^\natural$ is mapped to $u_i \in U$ by f. Otherwise, because $S^\natural \models \xi^S[total]$ holds, at least one of its disjuncts must be satisfied by each $u^\natural$, i.e. $S^\natural, [w \mapsto u^\natural]$ must satisfy $\text{node}_{u_j}^S(w)$ for some u_j; thus f's definition will map $u^\natural$ to this u_j. Therefore, $f(u^\natural)$ is well-defined.

In addition, every element $u_i \in U$ is assigned by f to some concrete element $u_i^\natural \in U^\natural$ such that $Z(v_i) = u_i^\natural$. According to Lemma 1, all such elements $u_i^\natural$ are different. Therefore, $f(u^\natural)$ is surjective.

Let p be a nullary predicate. Because $S^\natural$ satisfies $\xi_{nullary}^S$, it must satisfy each conjunct, in particular $S^\natural \models p^{\iota^S(p)()}$. Using Lemma 2 we get that $\iota^{S^\natural}(p)() \sqsubseteq \iota^S(p)()$.

Let $p \in P$ be a predicate of arity $r \geq 1$. Let $u_1^\natural, u_2^\natural, \ldots, u_r^\natural \in U^\natural$ and let us show that

$$\iota^{S^\natural}(p)(u_1^\natural, u_2^\natural, \ldots, u_r^\natural) \sqsubseteq \iota^S(p)(f(u_1^\natural), f(u_2^\natural), \ldots, f(u_r^\natural)) \tag{26}$$

Let Z be an assignment such that $Z(w_i) = u_i^\natural$ for $i = 1, \ldots, r$. Because $S^\natural \models \xi^S[p]$, we conclude that $S^\natural, Z$ satisfies the body of Eq. (7). Consider the conjunct of the body with premise $\bigwedge_{j=1}^r \text{node}_{f(u_j^\natural)}^S(w_j)$. By definition of f, $S^\natural, w_j \mapsto u_j^\natural$ satisfies $\text{node}_{f(u_j^\natural)}^S(w_j)$ for all $j = 1, \ldots, r$, which means that the premise is satisfied by $S^\natural, Z$. Therefore, the conclusion must hold: $S^\natural, Z \models p^{\iota^S(p)(f(u_1^\natural),\ldots,f(u_r^\natural))}(w_1, \ldots, w_r))$ and the result follows from Lemma 2.

Lemma D4 *For every* 3*-valued FO-identifiable structure* S*, and* 2*-valued structure* $S^\natural$ *such that* $S^\natural \models F$ *and* $S^\natural \sqsubseteq S$*,* $S^\natural \models \xi^S$*.*

Proof: Let $f\colon S^\natural \to S$ be a surjective function such that $S^\natural \sqsubseteq^f S$. Let $u_i^\natural$ be an arbitrary element such that $f(u_i^\natural) = u_i$. Define an assignment $Z^\natural$ such that $Z^\natural(v_i) = u_i^\natural$; $u_i^\natural$ must exist because f is surjective. Because S is FO-identifiable, by Definition 8 we conclude that for every $1 \leq i \leq n$, $S^\natural, Z^\natural \models \text{node}_{u_i}^S(v_i)$. Because f is a function, all $u_i^\natural$ are distinct elements, according to Lemma 1.

Because f is a function, for every $u^\natural$ there is u such that $f(u^\natural) = u$. Then, by Definition 8, $S^\natural, [w \mapsto u^\natural] \models \text{node}_u^S(w)$, i.e., every assignment to w in $S^\natural$ satisfies some disjunct of ξ_{total}^S. That is $S^\natural$ satisfies ξ_{total}^S.

For every nullary predicate $p \in \mathcal{P}_0$, using Eq. (1) and Lemma 2, we conclude that $S^\natural$ satisfies $p^{\iota^S(p)()}$. Therefore, $S^\natural$ satisfies $\xi_{nullary}^S$.

Let $p \in P$ be a predicate of arity r. Let $u_1^\natural, \ldots, u_r^\natural \in U^\natural$ and let $Z^\natural$ be an assignment such that $Z^\natural(w_i) = u_i^\natural$. We shall show that $S^\natural, Z^\natural$ satisfy the body of Eq. (7). If the premise of the implication is not satisfied then the formula vacuously holds. Otherwise, $S^\natural, Z^\natural \models \text{node}^S_{u_i}(w_i)$ for all $i = 1, \ldots, r$. Then, by Definition 8, $f(u_i^\natural) = u_i$. Using Eq. (1) on f, we get $\iota^{S^\natural}(p)(u_1^\natural, \ldots, u_r^\natural) \sqsubseteq \iota^S(p)(f(u_1^\natural), \ldots, f(u_r^\natural))$, which means that $\iota^{S^\natural}(p)(u_1^\natural, \ldots, u_r^\natural) \sqsubseteq \iota^S(p)(u_1, \ldots, u_r)$ holds. By Lemma 2, we conclude that $S^\natural, Z^\natural$ satisfies $p^{\iota^S(p)(u_1,\ldots,u_r)}(w_1, \ldots, w_r)$.

Lemma 4 *If* 3*-valued structure* $S = \langle U, \iota^S \rangle$ *over vocabulary* $\mathcal{P}$ *is* ICA *then:*

(i) *S is a bounded structure.*
(ii) *For each nullary predicate p, $\iota^S(p)() \in \{0, 1\}$.*
(iii) *For each element $u \in U$, and each unary predicate p, $\iota^S(p)(u) \in \{0, 1\}$.*

Proof: Let $S^\natural = \{U^\natural, \iota^{S^\natural}\}$ be a 2-valued structure, such that S is the canonical abstraction of $S^\natural$. Let $canonical\colon U^\natural \to U$ be the mapping that identifies S as the canonical abstraction of $S^\natural$.

(i) Show that S is a bounded structure. By Eq. (11), every abstract element represents concrete elements with the same canonical name. Thus, for two distinct abstract elements $u_0, u_1 \in U^S$, the canonical name of concrete elements represented by u_0 is different from the canonical name of concrete elements represented by u_1. Without loss of generality, assume that the canonical names differ in a unary predicate p, such that p evaluates to 0 on all concrete elements represented by u_0, and p evaluates to 1 on all concrete elements represented by u_1. From the join operation in Eq. (12), it follows that the value of p on u_0 must be 0 and the value of p on u_1 must be 1. This shows that, in general, every pair of distinct elements in S differs in a definite value of some unary predicate, proving that S is a bounded structure.
(ii) Let p be a nullary predicate. Show that $\iota^S(p)() \in \{0, 1\}$. By Eq. (12), $\iota^S(p)() = \sqcup\{\iota^{S^\natural}(p)()\} = \iota^{S^\natural}(p)()$. This means that p has the same value in S and $S^\natural$. Because $S^\natural$ is a concrete structure, the value of p must be definite.
(iii) Let p be a unary predicate and let $u \in U$. Show that $\iota^S(p)(u) \in \{0, 1\}$. Suppose that the opposite holds: $\iota^S(p)(u) = 1/2$. By Eq. (12), there exist two concrete elements, denoted by u_0 and u_1, such that $canonical(u_0) = u$ and $canonical(u_0) = u$, and p evaluates to 0 on u_0 and to 1 on u_1. Hence, these concrete elements have different canonical names and by Eq. (11) they cannot be mapped by $canonical$ to the same abstract element; this contradicts the supposition and hence $\iota^S(p)(u) \in \{0, 1\}$.

Lemma 5 *Every* 3*-valued structure S that is an* ICA *is canonical-FO-identifiable, where*

$$\textit{node}^S_{u_i}(w) \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}_1} p^{\iota^S(p)(u_i)}(w) \qquad (27)$$

Proof: Let $S = \{U, \iota^S\}$ be a 3-valued structure that is *ICA*. We shall show that every element $u \in U$ is canonical-FO-identifiable using the formula defined in Eq. (15). Let $S^\natural = \{U^\natural, \iota^{S^\natural}\}$ be a 2-valued structure, such that S is the canonical abstraction of $S^\natural$, induced by a function $canonical$, and let $u^\natural \in U^{S^\natural}$. By Definition 16, we have to show that the following holds:

$$canonical(u^\natural) = u \iff S^\natural, [w \mapsto u^\natural] \models \text{node}^S_u(w)$$

Proof of the if direction: Suppose that $S^\natural, [w \mapsto u^\natural] \models \text{node}^S_u(w)$. Let $u_1 = \mathit{canonical}(u^\natural)$. For the sake of argument, assume that $u_1 \neq u$. S is an *ICA* and using Lemma 4(i) we get that S is a bounded structure. By Definition 10, there exists a unary predicate p that evaluates to different definite values on u and u_1. Without loss of generality, suppose that p evaluates to 0 on u and to 1 on u_1. This implies the following two facts. First, from property Eq. (12) of the definition of canonical abstraction, p also evaluates to 1 on all concrete values mapped to u_1 by $\mathit{canonical}$; in particular, p must evaluate to 1 on $u^\natural$. Second, recall that by assumption, each conjunct of node^S_u must hold, i.e., for each predicate $p \in \mathcal{P}_1$, $S^\natural, [w \mapsto u^\natural] \models p^{\iota^S(p)(u)}(w)$. Because p evaluates to 0 on u, we get from Definition 9 that $S^\natural, [w \mapsto u^\natural] \models p^0(w)$, which means $\iota^{S^\natural}(p)(u^\natural) = 0$ and a contradiction is obtained.
Proof of the only-if direction: Suppose that $\mathit{canonical}(u^\natural) = u$. Because S is an *ICA* by Lemma 4(iii) we know that all unary predicates have definite values in S. Let p be a unary predicate. Let $B \in \{1, 0\}$ be such that $\iota^S(p)(u) = B$. Because p has definite value B on u in S, by Eq. (12) it must have the same definite value B on all concrete nodes in $S^\natural$ that are mapped to u by $\mathit{canonical}$; in particular, on $u^\natural$: $\iota^{S^\natural}(p)(u^\natural) = B$. Therefore, using Definition 9, $S^\natural, [w \mapsto u^\natural] \models p^B(w)$, in other words, $S^\natural, [w \mapsto u^\natural] \models p^{\iota^S(p)(u)}(w)$. This holds for all unary predicates, and thus holds for their conjunction as well, i.e., for the formula node^S_u.
Theorem 2 *For every* 3*-valued structure* S *that is an* ICA *and* 2*-valued structure* $S^\natural$

$$S^\natural \in \gamma_c(S) \textit{ iff } S^\natural \models \widehat{\gamma_c}(S)$$

Proof: In Lemma D5, we show that the if-direction holds, i.e., a 3-valued structure S is the canonical abstraction of every concrete structure satisfying the characteristic formula $\widehat{\gamma_c}(S)$; in Lemma D6 we show the other direction.

Lemma D5 *Let S be an* ICA *with set of individuals* $U = \{u_1, u_2, \ldots, u_n\}$*. Let* $\text{node}^S_{u_i}(w)$ *be an arbitrary formula free in* w*, used in* $\widehat{\gamma_c}$*, such that Lemma 1 holds. Then, for all* $S^\natural$ *such that* $S^\natural \models \widehat{\gamma_c}(S)$*,* S *is a canonical abstraction of* $S^\natural$*.*
Proof: Let $S^\natural = \langle U^\natural, \iota^\natural \rangle$ be a concrete structure such that $S^\natural \models \widehat{\gamma_c}(S)$. We shall construct a surjective function $\mathit{canonical}\colon U^\natural \to U$ such that $S^\natural$ is a canonical abstraction of S. From Definition 17 it follows, in particular, that $S^\natural \models \xi^S$. Let $Z^\natural$ be an assignment over $v_1, \ldots, v_n$ such that $S^\natural, Z^\natural \models \varphi$, where $\varphi \stackrel{\text{def}}{=} \bigwedge_{i=1}^n \text{node}^S_{u_i}(v_i)$, i.e., φ is the first line of Eq. (8) without the existential quantification). Note that all $Z^\natural(v_i)$ are distinct, according to Lemma 1. Define the function $\mathit{canonical}\colon U^\natural \to U$ by:

$$\mathit{canonical}(u^\natural) = \begin{cases} u_i \text{ if } Z^\natural(v_i) = u^\natural \\ u_j \text{ if for all } i,\ Z^\natural(v_i) \neq u^\natural \text{ and } u_j \text{ is an arbitrary element such that} \\ \quad S^\natural, [w \mapsto u^\natural] \models \text{node}^S_{u_j}(w) \end{cases} \tag{28}$$

Let us show that every concrete element is mapped to some element in U. In the case that $Z(v_i) = u^\natural$, the concrete element $u^\natural$ is mapped to $u_i \in U$ by $\mathit{canonical}$. Otherwise, because $S^\natural \models \xi^S[\mathit{total}]$ holds, at least one of its disjuncts must be satisfied by each $u^\natural$, i.e., $S^\natural, [w \mapsto u^\natural]$ must satisfy $\text{node}^S_{u_i}(w)$ for some u_i; thus $\mathit{canonical}$'s definition will map $u^\natural$ to this u_i. Therefore, $\mathit{canonical}(u^\natural)$ is well-defined.

In addition, every element $u_i \in U$ is assigned by $canonical$ to some concrete element $u_i^\natural \in U^\natural$ such that $Z(v_i) = u_i^\natural$. According to Lemma 1, all such elements $u_i^\natural$ are different. Therefore, $canonical(u^\natural)$ is surjective.

We shall show that $canonical$ satisfies Eq. (11) and Eq. (12); that is, $canonical$ identifies S as the canonical abstraction of $S^\natural$.

First, let us show that Eq. (12) holds for the abstraction imposed by $canonical$, namely that a predicate p in S has the most precise abstract value w.r.t. the concrete values that it represents, as is imposed by $canonical$.

Because S is an *ICA*, all nullary predicates in S must have definite values, by Lemma 4(ii). $S^\natural$ satisfies $\xi^S_{nullary}$; therefore, by Definition 9, nullary predicates in $S^\natural$ must have the same definite values as in S; this shows that Eq. (12) holds for nullary predicates.

Because S is an *ICA*, all unary predicates in S must have definite values, by Lemma 4(iii). Let p be a unary predicate and let $u \in U$ be an individual of S such that $\iota^S(p)(u) = b$. We shall show that p has the same definite value b on all concrete elements mapped to u by $canonical$. Because the join of these values is also b, we will get that Eq. (12) holds for p and u. Recall that $S^\natural$ satisfies formula $\xi^S[p]$, hence each assignment to w satisfies the conjunct $\text{node}^S_u(w) \Rightarrow p^b(w)$ of $\xi^S[p]$. Let $u^\natural \in U^\natural$ be an individual of $U^\natural$ such that $canonical(u^\natural) = u$ and consider an assignment in which w is mapped to $u^\natural$. By the definition of $canonical$, this assignment satisfies $\text{node}^S_u(w)$, the premise of the conjunct. Therefore, it satisfies the conclusion, i.e., $S^\natural, [w \mapsto u^\natural]$ satisfies $p^b(w)$. Using Definition 9 we get that $\iota^{S^\natural}(p)(u^\natural) = b$.

Let p be a predicate of arity $r > 1$. If p has a definite value b in S on a tuple $u_1, \ldots, u_r$, $\xi^S[p]$ requires that p evaluates to the same definite value b on every concrete tuple $u_1^\natural, \ldots, u_r^\natural$ such that $canonical(u_i^\natural) = u_i$ (by the same argument as for unary predicates). Therefore, the join operation returns b as the most precise abstract value of p for these concrete tuples. Otherwise, if p evaluates to $1/2$ on $u_1, \ldots, u_r \in U$, there must be two tuples of elements in $U^\natural$, say $u_{01}^\natural, \ldots, u_{0r}^\natural$ and $u_{11}^\natural, \ldots, u_{1r}^\natural$, such that $S^\natural, [w_1 \mapsto u_{01}^\natural, \ldots, w_r \mapsto u_{0r}^\natural] \models \neg p(w_1, \ldots, w_r)$ and $S^\natural, [w_1 \mapsto u_{11}^\natural, \ldots, w_1 \mapsto u_{1r}^\natural] \models p(w_1, \ldots, w_r)$, because $S^\natural \models \tau^S[p]$. Thus, p evaluates to 0 on the first tuple and to 1 on the second tuple of the concrete structure; therefore, the most precise value obtained by the join operation on these values is $1/2$.

We shall show that $canonical$ satisfies Eq. (11), i.e., it maps elements according to their canonical names. This involves showing two directions:

1. For the sake of contradiction, assume that there are two distinct elements $u_0^\natural, u_1^\natural \in U^\natural$ that have the same canonical name (meaning that for all $p \in \mathcal{P}_1$, $\iota^{S^\natural}(p)(u_0^\natural) = \iota^{S^\natural}(p)(u_1^\natural)$), but $canonical(u_0^\natural) \neq canonical(u_1^\natural)$. Because S is a bounded structure, there must be unary predicate p that evaluates to 0 on $canonical(u_0^\natural)$ and to 1 on $canonical(u_1^\natural)$. As shown above, p evaluates to the same definite values in the concrete structure $S^\natural$: $\iota^{S^\natural}(p)(u_0^\natural) = 0$, and $\iota^{S^\natural}(p)(u_1^\natural) = 1$ and a contradiction is obtained.

2. For the sake of contradiction, assume that two concrete elements, denoted by $u_0^\natural, u_1^\natural \in U^\natural$, have different canonical names, but are mapped by $canonical$ to the same same element in U: $canonical(u_0^\natural) = canonical(u_1^\natural)$, denoted by u. By definition of

$canonical$, $S^\natural, [w \mapsto u_i^\natural]$ satisfies $\text{node}^S_{canonical(u_i^\natural)}(w)$, for $i = 0, 1$, in other words $S^\natural, [w \mapsto u_i^\natural]$ satisfies $\text{node}^S_u(w)$. Therefore, it satisfies each conjunct of $node$ formula, i.e., for all p, $S^\natural, [w \mapsto u_i^\natural]$ satisfies $p^{\iota^S(p)(u)}(w)$. From this and the fact that all unary predicates in S have definite values because S is an *ICA*, we conclude by Definition 9, that $\iota^{S^\natural}(p)(u_i^\natural) = \iota^S(p)(u)$. Therefore, $\iota^{S^\natural}(p)(u_0^\natural) = \iota^S(p)(u)$ and $\iota^{S^\natural}(p)(u_1^\natural) = \iota^S(p)(u)$, for all $p \in \mathcal{P}_1$. Therefore, $u_0^\natural$ and $u_1^\natural$ have the same canonical name and a contradiction is obtained.

Lemma D6 *For every* 3*-valued structure* S *that is an* ICA *and* 2*-valued structure* $S^\natural$ *such that* $S^\natural \models F$*, such that* S *is the canonical abstraction of* $S^\natural$*,* $S^\natural \models \tau^S$*.*
Proof: Let $canonical\colon U^\natural \to U$ be the mapping that identifies S as the canonical abstraction of $S^\natural$. $canonical$ is a surjective function and possesses the properties in Eq. (11) and Eq. (12).

First, we show that $S^\natural \models \xi^S$. Let $u_i^\natural$ be an arbitrary element such that $canonical(u_i^\natural) = u_i$. Define an assignment $Z^\natural$ such that $Z^\natural(v_i) = u_i^\natural$; $u_i^\natural$ must exist because $canonical$ is surjective. Because S is canonical-FO-identifiable, by Lemma 5 we conclude that for every $1 \leq i \leq n$, $S^\natural, Z^\natural \models \text{node}^S_{u_i}(v_i)$. According to Lemma 1, all the $u_i^\natural$ are distinct elements.

Because $canonical$ is a function, for every $u^\natural$ there is a u such that $canonical(u^\natural) = u$. Then, by Definition 16, $S^\natural, [w \mapsto u^\natural] \models \text{node}^S_u(w)$, i.e., every assignment to w in $S^\natural$ satisfies some disjunct of ξ^S_{total}. That is, $S^\natural$ satisfies ξ^S_{total}.

Because S is an *ICA*, nullary predicates have the same definite values in S and in $S^\natural$, by Lemma 4(ii). Therefore, by Definition 9, $S^\natural$ satisfies $p^{\iota^S(p)()}$, for every nullary predicate $p \in \mathcal{P}_0$, which means that $S^\natural$ satisfies $\xi^S_{nullary}$.

Let $p \in P$ be a predicate of arity r. Let $u_1^\natural, \ldots, u_r^\natural \in U^\natural$ and let $Z^\natural$ be an assignment such that $Z^\natural(w_i) = u_i^\natural$. We shall show that $S^\natural, Z^\natural$ satisfies the body of Eq. (7). Consider a conjunct of the body. If the premise of the implication in this conjunct is not satisfied, then the conjunct vacuously holds. Otherwise, $S^\natural, Z^\natural \models \text{node}^S_{u_i}(w_i)$ for all $i = 1, \ldots, r$. Then, by Lemma 5, $canonical(u_i^\natural) = u_i$. We have two cases to consider: (i) if $\iota^S(p)(u_1, \ldots, u_r) = b \in \{1, 0\}$ then by Eq. (12) $\iota^{S^\natural}(p)(u_1^\natural, \ldots, u_r^\natural) = b$, in other words, $S^\natural, Z^\natural$ satisfies $p^b(w_1, \ldots, w_r)$. (ii) if $\iota^S(p)(u_1, \ldots, u_r) = 1/2$ then by Definition 9, $p^{\iota^S(p)(u_1,\ldots,u_r)}(w_1, \ldots, w_r) = p^{1/2}(w_1, \ldots, w_r) = \mathbf{1}$, which holds for any assignment.

To complete the proof, we show that for every $p \in \mathcal{P}_r$ of arity $r > 1$, $\tau^S[p]$ holds. Let p be a predicate that evaluates to $1/2$ on a tuple $u_1, \ldots, u_r \in S$. Because S is an *ICA* $\iota^S(p)(u_1, \ldots, u_r) = 1/2$ means that the join operation in Eq. (12) yields $1/2$. By the definition of join as the least upper bound, and using the information order in Definition 4, we conclude that (i) $S^\natural$ must contain at least two distinct tuples; denoted by $u_{01}^\natural, \ldots, u_{0r}^\natural$ and $u_{11}^\natural, \ldots, u_{1r}^\natural$. Because $canonical(u_{ij}^\natural) = u_j$ for $i = 0, 1$ and $j = 1, \ldots, r$, by Lemma 5 we get that $S^\natural, [w \mapsto u_{ij}^\natural] \models \text{node}^S_{u_j}(w)$. Therefore, each tuple satisfies $\bigwedge_{j=1}^r \text{node}^S_{u_j}(w_j)$. (ii) p evaluates to 0 on the first tuple and 1 on the second tuple. This shows that $S^\natural \models \tau^S[p]$.

Lemma 6 *Denote by $\mathcal{D}$ the set of all 2-valued structures that satisfy the integrity formula F: $\mathcal{D} \stackrel{\text{def}}{=} \{S^\natural \in$ 2-STRUCT$[\mathcal{P}] \mid S^\natural \models F\}$. Let S be an ICA structure. There exists a set of ICA structures X such that $\gamma_c(X) = \mathcal{D} \setminus \gamma_c(S)$.*
Proof: Denote by Y the set of all ICA structures over a fixed vocabulary $\mathcal{P}$, i.e., $\gamma_c(Y) = \mathcal{D}$. We claim that X is defined by $Y \setminus S$. By definition, $\gamma_c(X) = \gamma_c(Y \setminus S)$, and we show that $\gamma_c(Y \setminus S) = \gamma_c(Y) \setminus \gamma_c(S)$. By the definitions of Y and γ_c in Eq. (13), $\gamma_c(Y \setminus S) \supseteq \mathcal{D} \setminus \gamma_c(S)$ holds. To complete the proof, we show that the other direction of inclusion holds as well. For the sake of argument, assume that there exists a 2-valued structure $S^\natural$ that belongs to both $\gamma_c(S)$ and $\gamma_c(Y \setminus S)$. Thus, by Definition 15, there exists an ICA structure S' such that $canonical(S^\natural) = S'$, and S' is different from S. From Eq. (12), it follows that $canonical(S^\natural) \neq S$, which contradicts the assumption that $S^\natural \in \gamma_c(S)$.
Lemma 7 *Consider the formula τ^S from Eq. (17), for some ICA structure S. There exists a set of ICA structures X, such that the formula $F \wedge \neg\tau^S$ is equivalent to the formula $\widehat{\gamma}_c(X)$.*
Proof: Let $\mathcal{D}$ be the set of all 2-valued structures that satisfy the integrity formula F. Let X be the set of ICA structures that describes the complement of $\gamma_c(S)$, as given by Lemma 6. Let $S^\natural$ be a 2-valued structure such that $S^\natural \in \gamma_c(X)$ if and only if $S^\natural \in \mathcal{D} \setminus \gamma_c(S)$. The right-hand side simplifies to $S^\natural \in \mathcal{D}$ and $S^\natural \notin \gamma_c(S)$. Applying Theorem 2, we get that $S^\natural \models \widehat{\gamma}_c(X)$ if and only if $S^\natural$ satisfies F but does not satisfy $\widehat{\gamma}_c(S)$. Using Eq. (18), this is equivalent to $S^\natural \models F \wedge \neg\tau^S$.
Theorem 3 *For every 3-valued structure S, and a 2-valued structure $S^\natural$:*

$$S^\natural \in \gamma(S) \textit{ iff } S^\natural \models \widehat{\gamma}_{NP}(S)$$

Proof: In Lemma D7, we show that the if-direction holds, i.e., every concrete structure satisfying the NP-characteristic formula $\widehat{\gamma}_{NP}$ is indeed in $\gamma(S)$. In Lemma D8 we show the only-if part.

Lemma D7 *Let S be a logical structure with set of individuals $U = \{u_1, u_2, \ldots, u_n\}$. Then, for all $S^\natural$ such that $S^\natural \models \widehat{\gamma}_{NP}(S)$, $S^\natural \in \gamma(S)$.*
Proof: Let $S^\natural = \langle U^\natural, \iota^\natural \rangle$ be a concrete structure such that $S^\natural \models \widehat{\gamma}(S)$. We shall construct a surjective function $f\colon U^\natural \to U$ such that $S^\natural \sqsubseteq^f S$. Let $Z^\natural$ be an assignment such that $S^\natural, Z^\natural \models \varphi$ where φ is the body of ξ^S without the existential quantifiers on sets. Let $Z^\natural(V_i) = U_i \subseteq U^\natural$. Consider the following definition:

$$f(u^\natural) = \{u_i \mid u^\natural \in U_i\} \tag{29}$$

$f(u^\natural)$ is a set of size at most 1 because the pair $S^\natural, Z^\natural$ satisfies the sub-formula $\xi^S_{disjoint}$. This insures that the sets $U_1, \ldots, U_n$ are disjoint, i.e., each concrete element belongs to at most one set. For simplicity, we say that $f(u^\natural) = u_i$, whenever $f(u^\natural) = \{u_i\}$.

We shall show that every concrete element is mapped by f to some element in U. Because $S^\natural, Z^\natural$ satisfies ξ^S_{total}, we conclude that every concrete element satisfies the formula $\text{node}^S_{u_i}(w)$ for some u_i. Also, $\text{node}^S_{u_i}(w)$ given in Definition 18 is a membership test in the set V_i; therefore, every concrete element must be a member of some set U_i. Thus, $u^\natural$ is mapped to $u_i \in U$, by the definition of f in Eq. (29). This shows that f is well-defined.

Because $S^\natural, Z^\natural$ satisfies $\models \xi^S_{non\_empty}[i]$ for $i = 1, \ldots, n$, it must be that every U_i contains at least one element, say $u_i^\natural$, that is mapped to u_i by f. Because the sets are disjoint, all such elements $u_i^\natural$ are different. Therefore, f is surjective.

Let p be a nullary predicate. Because $S^\natural$ satisfies $\xi^S_{nullary}$, it must satisfy each conjunct, in particular $S^\natural \models p^{\iota^S(p)()}$. Using Lemma 2 we get that $\iota^{S^\natural}(p)() \sqsubseteq \iota^S(p)()$.

Let $p \in P$ be a predicate of arity $r \geq 1$. Let $u_1^\natural, u_2^\natural, \ldots, u_r^\natural \in U^\natural$ and let us show that

$$\iota^{S^\natural}(p)(u_1^\natural, u_2^\natural, \ldots, u_r^\natural) \sqsubseteq \iota^S(p)(f(u_1^\natural), f(u_2^\natural), \ldots, f(u_r^\natural)) \tag{30}$$

Let $Z_1^\natural$ be an extension of assignment $Z^\natural$ such that $Z_1^\natural(w_i) = u_i^\natural$ for $i = 1, \ldots, r$. Because $S^\natural, Z^\natural \models \xi^S[p]$, we conclude that $S^\natural, Z_1^\natural$ satisfies the body of Eq. (7). Consider the conjunct of the body with premise $\bigwedge_{j=1}^r \mathrm{node}^S_{f(u_j^\natural)}(w_j)$. By definition of f, $S^\natural, w_j \mapsto u_j^\natural$ satisfies $\mathrm{node}^S_{f(u_j^\natural)}(w_j)$ for all $j = 1, \ldots, r$, which means that the premise is satisfied by $S^\natural, Z_1^\natural$. Therefore, the conclusion must hold:
$S^\natural, Z_1^\natural \models p^{\iota^S(p)(f(u_1^\natural),\ldots,f(u_r^\natural))}(w_1, \ldots, w_r))$ and the result follows from Lemma 2.

Lemma D8 *For every* 3*-valued structure* S*, and* 2*-valued structure* $S^\natural$ *such that* $S^\natural \models F$ *and* $S^\natural \sqsubseteq S$*,* $S^\natural \models \xi^S$*.*

Proof: Let $f\colon S^\natural \to S$ be a surjective function such that $S^\natural \sqsubseteq^f S$. Define an assignment $Z^\natural$ such that $Z^\natural(V_i) = U_i \subseteq U^\natural$ and $U_i = \{u_i^\natural \mid f(u_i^\natural) = u_i\}$.

Because f is a surjective function, there must exist at least one concrete element that is mapped to u_i by f. This element belongs to the set U_i. Therefore, $S^\natural, Z^\natural \models \bigwedge_{i=1}^n \xi^S_{non\_empty}[i]$.

Because f is a well-defined function, it maps each concrete element to exactly one element $u_i \in U$, which induces the set U_i. Therefore, a concrete element cannot belong to more than one set; hence $S^\natural, Z^\natural \models \bigwedge_{k \neq j} \xi^S_{disjoint}[k, j]$.

Because f is a function, f maps every concrete element to some element in U. Therefore, every concrete element belongs to some set, i.e., satisfies some disjunct of ξ^S_{total}. That is $S^\natural, Z^\natural \models \xi^S_{total}$.

For every nullary predicate $p \in \mathcal{P}_0$, using Eq. (1) and Lemma 2, we conclude that $S^\natural, Z^\natural$ satisfies $p^{\iota^S(p)()}$. Therefore, $S^\natural, Z^\natural \models \xi^S_{nullary}$.

Let $p \in P$ be a predicate of arity r. Let $u_1^\natural, \ldots, u_r^\natural \in U^\natural$ and let $Z_1^\natural$ be an extension of assignment $Z^\natural$ such that $Z_1^\natural(w_i) = u_i^\natural$. We shall show that $S^\natural, Z_1^\natural$ satisfy the body of Eq. (7). If the premise of the implication is not satisfied, then the formula vacuously holds. Otherwise, $S^\natural, Z_1^\natural \models \mathrm{node}^S_{u_i}(w_i)$ for all $i = 1, \ldots, r$. Then, by Definition 18, $u_i^\natural$ belongs to the set U_i. The definition of U_i implies that $f(u_i^\natural) = u_i$. Using Eq. (1), we get $\iota^{S^\natural}(p)(u_1^\natural, \ldots, u_r^\natural) \sqsubseteq \iota^S(p)(f(u_1^\natural), \ldots, f(u_r^\natural))$ which means $\iota^{S^\natural}(p)(u_1^\natural, \ldots, u_r^\natural) \sqsubseteq \iota^S(p)(u_1, \ldots, u_r)$. By Lemma 2 we conclude that $S^\natural, Z^\natural$ satisfies $p^{\iota^S(p)(u_1,\ldots,u_r)}(w_1, \ldots, w_r)$.