On the Boolean Algebra of Shape Analysis Constraints

Viktor Kuncak and Martin Rinard
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{vkuncak,rinard}@csail.mit.edu
MIT CSAIL Technical Report No 916
VK0108, August 2003

Abstract

Shape analysis is a promising technique for statically verifying and extracting properties of programs that manipulate complex data structures. We introduce a new characterization of constraints that arise in parametric shape analysis based on manipulation of three-valued structures as dataflow facts.

We identify an interesting syntactic class of first-order logic formulas that captures the meaning of three-valued structures under concretization. This class is broader than previously introduced classes, allowing for a greater flexibility in the formulation of shape analysis constraints in program annotations and internal analysis representations. Three-valued structures can be viewed as one possible normal form of the formulas in our class.

Moreover, we characterize the meaning of three-valued structures under "tight concretization". We show that the seemingly minor change from concretization to tight concretization increases the expressive power of three-valued structures in such a way that the resulting constraints are closed under all boolean operations. We call the resulting constraints *boolean shape analysis constraints*.

The main technical contribution of this paper is a natural syntactic characterization of boolean shape analysis constraints as arbitrary boolean combinations of first-order sentences of certain form, and an algorithm for transforming such boolean combinations into the normal form that corresponds directly to three-valued structures.

Our result holds in the presence of arbitrary shape analysis instrumentation predicates. The result enables the reduction (without any approximation) of the entailment and the equivalence of shape analysis constraints to the satisfiability of shape analysis constraints. When the satisfiability of the constraints is decidable, our result implies that the entailment and the equivalence of the constraints are also decidable, which enables the use of constraints in a compositional shape analysis with a predictable behavior.

Keywords: Shape Analysis, Static Analysis, Program Verification, Program Specification, First-Order Logic

*This research was supported in part by DARPA Contract F33615-00-C-1692, NSF Grant CCR00-86154, NSF Grant CCR00-63513, and the Singapore-MIT Alliance.

†Version of November 4, 2003, 1:35pm

Contents

1 Introduction

Dynamically Allocated Data Structures Modern software is becoming increasingly complex. This complexity corresponds to the complex web of relationships between different program entities. Object-oriented programming languages such as Java use references between objects dynamically allocated in the heap to model the relationships between entities of the application domain. Dynamic allocation of objects provides flexibility that helps applications adapt to the dynamically changing environment. To model the evolution of the relationships between objects, applications perform destructive updates of the heap. Because writing applications in this programming model is error-prone, tools for statically verifying partial correctness of such programs are very valuable.

Shape Analysis Shape analysis techniques [49], [20, 21, 43] can verify and derive precise properties of objects in the heap. Shape analysis therefore appears essential for reasoning about programs written in modern imperative programming languages. Shape analysis is promising as a general-purpose verification technique, because of its ability to reason about graphs as general structures, and the ability to summarize properties of unbounded sets of objects. Shape analysis such as [49] is effective in deriving program properties at each program point and synthesizing loop invariants while maintaining high precision and strong soundness guarantees.

Program Specifications The ability to write program specifications can greatly improve the effectiveness of shape analysis (and, for that matter, the effectiveness of any static or dynamic analysis in general). First of all, specifications indicate the desired property to be verified. Next, specifications allow the use of assume/guarantee reasoning, which improves the scalability of the analysis and enables its application to reusable program components. Finally, if necessary, specifications can guide the static analysis and provide hints for it, while at the same time leaving a documentation trace explaining the correctness of the program.

Analysis-Specification Gap The representation of program properties used by the program analysis is often different from the representation of program properties that is appropriate for program annotations. To synthesize invariants using a fixpoint computation, program analysis often uses a finite lattice of program properties. On the other hand, program annotations should be expressed in some convenient, well-known notation, such as a variation of first-order logic. A program analysis that utilizes program specifications must bridge the gap between the analysis representation and the program annotations.

Logic-Based Shape Analysis A promising shape analysis approach [49] based on abstract interpretation [14] uses the lattice of three-valued logical structures for fixpoint computation. The fact that the approach is based on logic makes bridging the gap between the program annotations and the analysis representation much easier, yet it does not eliminate it entirely. The original TVLA system [38] uses three-valued structures to specify preconditions, which corresponds to specifications with disjoint, non-empty sets of objects and is sometimes unnecessarily verbose. The followup work [46, 52] shows how to use arbitrary first-order formulas for program annotations and convert the annotations to three-valued structures using a theorem prover. Because the first-order logic is undecidable in general, it is interesting to consider alternative approaches with a potentially more predictable behavior.

1.1 Contributions

Mediating the Analysis-Specification Gap This paper addresses the gap between program annotations and three-valued structures by providing an algorithm for transforming annotations (expressed as formulas) into three-valued structures, as well as a way of viewing a class of canonical program annotations as three-valued structures. Because we restrict our attention to formulas of a particular form, we are able to find a complete and sound algorithm for generating three-valued structures. The completeness makes our algorithm potentially more predictable than the use of theorem provers on arbitrary formulas. Our algorithm shows that the expressive power of our specifications is equal to the expressive power of three-valued structures. Nevertheless, our specifications may use sets that are potentially intersecting or empty, which makes the annotations more flexible than three-valued structures themselves where summary nodes represent only disjoint sets of nodes. Moreover, the characterization of existing shape analysis constraints as disjunctive normal forms of formulas suggests that alternative representations for three-valued structures may be possible [5, 41, 42].

The characterization of three-valued structures by formulas allows us to easily prove properties that are less obvious in the three-valued structure view, such as closure of three-valued structures under conjunction. To compute the conjunction of three-valued structures, we use the fact that three-valued structures correspond to disjunctive normal forms of positive boolean combinations of formulas; the computation of the conjunction of three-valued structures then corresponds to a transformation of a conjunction of two disjunctive normal forms into a new disjunctive normal form.

Boolean Shape Analysis Constraints By considering the "tight concretization" semantics instead of the concretization semantics of three-valued structures, we obtain a richer class of formulas, namely the class of all boolean combinations of certain atomic formulas. This characterization implies that three-valued structures under concretization are closed under all boolean operations. We therefore call the constraints arising from tight concretization of three-valued structures *boolean shape analysis constraints*.

Although the notion of tight concretization is not new, the characterization of boolean shape analysis constraints as boolean combinations of certain formulas is surprisingly elegant and has not been observed before.

Consequences of Boolean Closure The resulting closure properties of boolean shape analysis constraints have several potential uses. The closure under disjunction is necessary for fixpoint computation in dataflow analysis and can be conveniently computed even for shape analysis constraints; what our results show is that boolean shape analysis constraints are also closed under conjunction and negation.

The conjunction of constraints is needed, for example, in compositional interprocedural shape analysis, which computes the relation composition of relations on states. Conjunction allows the analysis to simultaneously retain the call-site specific information that the callee preserves across the call, and the postcondition which summarizes the actions of the callee.

The negation of constraints is useful for expressing deterministic branches in control-flow graphs. For example, an `if` statement with the condition c results in conjoining the dataflow fact d to yield $d \wedge c$ in the `if` branch, and $d \wedge \neg c$ in the `else` branch. Similarly, the `assert`(c) statement, which is an important mechanism for program specification, has (in the relational semantics) the condition $\neg c$ for the branch which leads to an error state.

Finally, the closure under negation implies that both the implication and the equivalence of shape analysis constraints are reducible to the satisfiability of shape analysis constraints. This result is in contrast to "regular graph constraints" of [35], which have a decidable satisfiability problem but undecidable implication and the equivalence problems. The entailment problem is also important for compositional analysis which uses assume/guarantee reasoning. By introducing history variables that store the initial state of the program, a compositional interprocedural shape analysis can use shape analysis constraints to represent relations on program states. The fundamental operations of such compositional shape analysis are computation of the best approximation of relation composition and checking the subset of relations. Closure under boolean operations allows reducing all these operations to the satisfiability of shape analysis constraints.

Scope of the Result Our result is relevant in the presence of shape analysis instrumentation predicates defined using arbitrary first-order formulas. What the particular choice of instrumentation predicates determines is whether the satisfiability problem for boolean shape analysis constraints is decidable. If the satisfiability problem for shape analysis constraints with a particular choice of instrumentation predicates is decidable, our closure results imply that the entailment problem is also decidable, and that the constraints are suitable for use in an instantiation of the shape analysis framework.

Summary of contributions We can summarize the contributions of this paper as follows:

1. We give a concrete example that shows how elements of the lattice for fixpoint computation can be viewed as formulas in a canonical form; we believe that this idea is useful in general.

2. We identify a syntactic class of formulas whose expressive power matches exactly the semantics of three-valued structures under concretization. The resulting constraints are closed under disjunction and conjunction, but are not necessarily closed under negation.

3. We identify a syntactic class of formulas whose expressive power matches exactly the semantics of three-valued structures under *tight* concretization. The resulting *boolean shape analysis constraints* are closed under *all* boolean operations such as disjunction, conjunction, negation, implication, and equivalence.

4. We observe that the closure under all boolean operations allows reducing the entailment and the equivalence problems to the satisfiability problem of boolean shape analysis constraints.

5. We show that each three-valued structure has a model within the set of two-valued structures, which means that the satisfiability problem of shape analysis constraints is trivial over the set of all two-valued structures.

6. We show that, even in the presence of instrumentation predicates, our results allow reducing the entailment and the equivalence problems of shape analysis constraints to the satisfiability problem.

1.2 Organization of the Paper

The rest of the paper is organized as follows. Section 2 reviews the basic notions of two-valued and three-valued structures. Section 3 presents a series of syntactic classes of formulas of equal expressive power that all characterize the meaning of two-valued structures under concretization (Corollary 28). As a consequence, we derive the closure of constraints under disjunction and conjunction (Corollary 29). Section 3 is to some extent a preparation for Section 4. Section 4 introduces a series of formulas that have the same expressive power (Corollary 45) as the three-valued structures under tight concretization (Definition 30), and introduces the name boolean shape analysis constraints (Definition 46). Section 4.1 observes that boolean shape analysis constraints are closed under all boolean operations and derives some consequences of these closure properties. Section 4.2 shows that boolean shape analysis constraints are the smallest extension of three-valued structures under concretization which is closed under all boolean operations (Proposition 51). Section 4.3 shows how to transform a three-valued structure into a structure where all unary predicates have definite values. Section 5 introduces the decidability problems for three-valued structures, shows that every three-valued structure is satisfiable, and derives the decidability of the implication and the equivalence as a consequence of the decidability of satisfiability and the closure under boolean operations. Section 6 generalizes the results of the previous sections to the case when the values of some predicates are constrained by first-order formulas. Section 7 presents the related work and Section 8 concludes.

2 Preliminaries

In this section we define some preliminary concepts used throughout the paper. We mostly follow the setup of [49] and for completeness repeat some of the definitions from [49, 52].

Let $\mathcal{A}$ be a finite set of unary relation symbols (with a typical element $A \in \mathcal{A}$) and $\mathcal{F}$ a finite set of binary relation symbols (with a element $f \in \mathcal{F}$). For simplicity, we consider only unary and binary relation symbols because they appear to be the most useful cases. Most of our results generalize naturally to n-ary relations.

Two-Valued Structures We next introduce two-valued structures. A two-valued structure consists of a domain $U^\sharp$ and the interpretation $\iota^\sharp$ of relation symbols. Our language does not contain function symbols because we represent all functions as relations. In model theory and logic, a two-valued structure corresponds to a structure (model) whose domain $U^\sharp$ is finite.

Definition 1 *A* two-valued structure *is a pair $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$ where $U^\sharp$ is a finite non-empty set (of "concrete individuals"), $\iota^\sharp(A) \in U^\sharp \to \{0,1\}$ for $A \in \mathcal{A}$, and $\iota^\sharp(f) \in (U^\sharp)^2 \to \{0,1\}$ for $f \in \mathcal{F}$. Let*

$$\text{2-STRUCT} = \{S^\sharp \mid S^\sharp = \langle U^\sharp, \iota^\sharp \rangle \textit{ is a two-valued structure}\}$$

In program analysis, each two-valued structure represents a state of the program. The use of structures for representing program state has proven useful in the shape analysis [49], Abstract State Machines [7], the Alloy modelling language and analyzer [27], and relational databases [13, 15].

Three-Valued Structures A three-valued structure is a model for Kleene's three-valued logic [32, 44] and differs from two-valued structure by the fact that predicates can have three-possible values: $\{0\}$, $\{1\}$, and $\{0,1\}$. (The truth-values $\{0\}$, $\{1\}$, $\{0,1\}$ of three-valued logic are denoted by, respectively, 0, 1, 1/2 in [49].)

Definition 2 *A* three-valued structure *is a pair $S = \langle U, \iota \rangle$ where U is a finite non-empty set (of "abstract individuals"), $\iota(A) \in U \to \{\{0\},\{1\},\{0,1\}\}$ for $A \in \mathcal{A}$ and and $\iota(f) \in U^2 \to \{\{0\},\{1\},\{0,1\}\}$ for $f \in \mathcal{F}$. Let*

$$\text{3-STRUCT} = \{S \mid S = \langle U, \iota \rangle \textit{ is a three-valued structure}\}$$

The parametric shape analysis framework [49] uses three-valued structures to specify sets of two-valued structures according to Definition 4 below.

Formulas We assume the usual syntax and semantics of first-order logic. We use an abstract view of the syntax of formulas in first-order logic which takes into account associativity, commutativity, and idempotence of conjunction and disjunction, and the property $\neg\neg p = p$. A conjunction with zero conjuncts denotes true; a disjunction with zero disjuncts denotes false.

If $S^\sharp$ is a two-valued structure and F a formula with free variables $x_1, \ldots, x_n$ and $u_1^\sharp, \ldots, u_n^\sharp \in S^\sharp$, then $e = [x_1 \mapsto u_1^\sharp, \ldots, x_n \mapsto u_n^\sharp]$ denotes an environment mapping x_i to $u_i^\sharp$ for all $1 \leq i \leq n$, and $(\llbracket F \rrbracket^{S^\sharp} e)$ denotes the value $v \in \{0,1\}$ of the formula F in the model $S^\sharp$ under the environment e. Instead of $(\llbracket F(x) \rrbracket^{S^\sharp} [x \mapsto u^\sharp])$ we sometimes write $S^\sharp \models F(u^\sharp)$ and omit $S^\sharp$ if it is understood from the context. If F has no free variables we denote the truth value v of F in $S^\sharp$ simply by $\llbracket F \rrbracket^{S^\sharp}$ and write $S^\sharp \models F$ for $\llbracket F \rrbracket^{S^\sharp}$=1. Definition 3 below defines the set of models of a formula in the expected way.

Definition 3 (Models of sets of Formulas) *Let F be a first-order formula. Then*

$$\gamma_{\mathrm{F}}^*(F) = \{S^\sharp \in \text{2-STRUCT} \mid \llbracket F \rrbracket^{S^\sharp} = 1\}$$

If C is a set of formulas, define

$$\mathsf{models}[C] = \{\gamma_{\mathrm{F}}^*(F) \mid F \in C\}$$

The transitive closure operator or inductive definitions are useful for describing instrumentation predicates (Section 6), but the presence of such constructs in logic is largely orthogonal to the results of this paper.

For simplicity we treat equality like any other binary relation symbol and do not treat summary nodes specially, but our results are also useful in the presence of summary nodes (see [36], as well as [44] and Section 6).

3 Three-Valued Structures with Concretization

This section uses first-order formulas to characterize the meaning of two-valued structures under the usual concretization function. Section 4 presents an alternative semantics using tight concretization, which yields constraints with better closure properties.

The following notion of concretization corresponds to [48, Definition 3.5]. The concretization function γ^* provides the semantics for sets of three-valued structures.

Definition 4 (Homomorphism and Concretization) *Let $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$ be a two-valued structure, $S = \langle U, \iota \rangle$ a three-valued structure, and $h : U^\sharp \to U$ a surjective total function. We write $S^\sharp \sqsubseteq^h S$, iff*

1. *for every $A \in \mathcal{A}$ and $u \in U$:*
$$\iota(A)(u) \supseteq \{\iota^\sharp(A)(u^\sharp) \mid h(u^\sharp) = u\}$$
2. *for every $f \in \mathcal{F}$ and $u_1, u_2 \in U$:*
$$\iota(f)(u_1, u_2) \supseteq \{\ \iota^\sharp(f)(u_1{}^\sharp, u_2{}^\sharp) \mid h(u_1{}^\sharp) = u_1 \ \wedge\ h(u_2{}^\sharp) = u_2\}$$

We write $S^\sharp \sqsubseteq S$ iff there exists a surjective total function h such that $S^\sharp \sqsubseteq^h S$. We call any such h homomorphism *from $S^\sharp$ to S. The concretization of a three-valued structure S, denoted $\gamma(S)$, is given by:*

$$\gamma(S) = \{S^\sharp \mid S^\sharp \sqsubseteq S\}$$

We extend γ to γ^* acting on sets of three-valued structures so that the set denotes a disjunction:

$$\gamma^*(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} \gamma(S)$$

The function h from Definition 4 is called "embedding" in [49]. (We choose to call h "homomorphism" because in literature the term "embedding" sometimes implies injectivity whereas in shape analysis h is not required to be injective, and almost never is injective.)

Bounded Structures Each set of three-valued structures $\mathcal{S}$ specifies a set of heaps $\gamma^*(\mathcal{S})$. Each such set $\gamma^*(\mathcal{S})$ is definable as the set of models of a formula in existential monadic second-order logic; the second-order existential quantification arises from the existential quantification over the homomorphisms h. Constraints that involve unrestricted second-order existential quantifications have several undesirable properties [34, 35]. We therefore restrict our attention to *bounded structures*, where the homomorphism h is determined as the natural map associated with the partition of the elements of $U^\sharp$ according to the values some chosen finite set of predicates.

For the purpose of this paper, we define bounded structures as follows. Let $\mathcal{A}_1 \subseteq \mathcal{A}$ be a finite subset of unary predicates. We call elements of $\mathcal{A}_1$ *abstraction predicates.*

Definition 5 (Bounded Structure) *We say that a three-valued structure $S = \langle U, \iota \rangle$ is $\mathcal{A}_1$-bounded iff both of the following two conditions hold:*

1. *$\iota(A)(u) \in \{\{0\}, \{1\}\}$ for all $A \in \mathcal{A}_1$ and all $u \in U$;*
2. *if $u_1, u_2 \in U$ and $u_1 \neq u_2$ then $\iota(A)(u_1) \neq \iota(A)(u_2)$ for some $A \in \mathcal{A}_1$.*

Definition 6 (Concretization Definability) *The set of sets of heaps definable via three-valued structures with concretization is defined by:*

$$\mathsf{models}[T_1] = \{\gamma^*(\mathcal{S}) \mid \ \mathcal{S} \text{ a finite set of } \mathcal{A}_1\text{-bounded three-valued structures } \}$$

Note that we use the same notation $\mathsf{models}[X]$ when X denotes a set of structures (Definition 6) and when X denotes a set of formulas (Definition 3). There is no confusion because we use distinct names for sets of structures and sets of formulas.

We proceed to characterize the set $\mathsf{models}[T_1]$ as the set of models of formulas of a certain form.

We define the notion of a *cube* first.

Definition 7 (Exponent Notation) *If $A \in \mathcal{A}$ and $\alpha \in \{0, 1\}$ then A^α is defined by $A^1 = A$ and $A^0 = \neg A$.*

Definition 8 (Cube) *A cube over $\mathcal{A}_1$ (or just "cube" for short) is an expression $P(x)$ of the form*

$$A_1^{\alpha_1}(x) \wedge \ldots \wedge A_q^{\alpha_q}(x)$$

where $\alpha_1, \ldots, \alpha_q \in \{0, 1\}$.

R_1-literals are the building blocks for formulas used to form constraints that characterize $\mathsf{models}[T_1]$.

Definition 9 (R_1-literal) *Let $P_1(x), P_2(y)$ range over cubes over $\mathcal{A}_1$, let A range over elements of $\mathcal{A} \setminus \mathcal{A}_1$, and let f range over $\mathcal{F}$.*

An R_1-literal is a formula of one of the following forms:

$\exists x.\ P_1(x)$	*node present*
$\neg\exists x.\ P_1(x)$	*node absent*
$\neg\exists x.\ P_1(x) \wedge A(x)$	*property does not hold*
$\neg\exists x.\ P_1(x) \wedge \neg A(x)$	*property holds*
$\neg\exists x \exists y.\ P_1(x) \wedge P_2(y) \wedge f(x, y)$	*no edge*
$\neg\exists x \exists y.\ P_1(x) \wedge P_2(y) \wedge \neg f(x, y)$	*must edge*

We first introduce the class of R_1-formulas that satisfy syntactic invariants that make them isomorphic to three-valued structures.

Definition 10 (R_1-formulas) *Let $P(x)$, $P_1(x)$, $P_2(y)$ denote cubes over $\mathcal{A}_1$. A* canonical *conjunction of R_1 literals is a conjunction of R_1-literals that satisfies the following conditions:*

1. *for each $P(x)$ a cube over $\mathcal{A}_1$, exactly one of the conjuncts $\exists x.P(x)$ and $\neg\exists x.P(x)$ occurs in the conjunction;*
2. *there is at least one cube $P(x)$ such that the conjunct $\exists x.P(x)$ occurs in the conjunction;*
3. *if the conjunct $\neg\exists x.P(x)$ occurs, then this conjunct is the only occurrence of the cube $P(x)$ (and the cube $P(y)$) in the conjunction;*
4. *for each cube $P(x)$, and $A \in \mathcal{A} \setminus \mathcal{A}_1$, at most one of the conjuncts $\neg\exists x.P(x) \wedge A(x)$ and $\neg\exists x.P(x) \wedge \neg A(x)$ occur;*
5. *for every two cubes $P_1(x)$ and $P_2(y)$, at most one of the conjuncts*

   $$\neg\exists x \exists y.\ P_1(x) \wedge P_2(y) \wedge f(x, y)$$

   and

   $$\neg\exists x \exists y.\ P_1(x) \wedge P_2(y) \wedge \neg f(x, y)$$

   occurs.

Define an R_1-formula as any disjunction of canonical conjunctions of R_1-literals.

In Definition 10 and throughout the paper, the symbol R_1 alone denotes R_1-*formulas*, so $\mathsf{models}[R_1]$ is the set of all models of all R_1-formulas (as opposed to, for example, the set of models of all R_1-*literals*).

The following Proposition 11 shows that three-valued structures and R_1-formulas define same sets of two-valued structures. The proof of Proposition 11 is straightforward because the set of R_1 formulas was chosen to facilitate the proof. The proof shows that there is a semantic-preserving bijection between three-valued structures and canonical conjunctions of R_1-literals.

Proposition 11 $\mathsf{models}[R_1] = \mathsf{models}[T_1]$

Proof. The idea of the proof is the following. Each bounded three-valued structure can be represented as a canonical conjunction of R_1-literals, and each canonical conjunction of R_1-literals can be represented as a bounded three-valued structure. Therefore, disjunctions of canonical conjunctions of R_1-literals correspond to sets of bounded three-valued structures.

We next give a function μ mapping each bounded three-valued structure S to a canonical conjunction of R_1-literals $\mu(S)$. We show that S and $\mu(S)$ represent same set of two-valued structures. Moreover, each canonical conjunction of R_1-literals is equal to $\mu(S)$ for some three-valued structure S.

Let $S = \langle U, \iota \rangle$ be an $\mathcal{A}_1$-bounded three-valued structure. Define the formula $\mu(S)$ as the conjunction of the following R_1-literals.

Define first, for each $u \in U$, a cube over $\mathcal{A}_1$ corresponding to u, denoted $\pi(u)(x)$, by

$$\pi(u)(x) = \bigwedge_{A \in \mathcal{A}_1} A^{\alpha(A)}(x)$$

where

$$\alpha(A) = \begin{cases} 1, & \text{if } \iota(A)(u) = \{1\} \\ 0, & \text{if } \iota(A)(u) = \{0\} \end{cases}$$

α is well-defined because $\iota(A) \in \{\{0\}, \{1\}\}$ for $A \in \mathcal{A}_1$. We next introduce the R_1-literals.

Node existence. For each $u \in U$, introduce an R_1-literal

$$\exists x.\pi(u)(x) \tag{1}$$

For each remaining $\mathcal{A}_1$-cube $P(x)$, that is, for each cube $P(x)$ such that $\pi(u)(x) \neq P(x)$ for all $u \in U$, introduce an R_1-literal

$$\neg\exists x.P(x) \tag{2}$$

Node properties. Let $u \in U$ and $A \in \mathcal{A} \setminus \mathcal{A}_1$. If $\iota(A)(u) = \{1\}$, introduce the R_1-literal

$$\neg\exists x.\ \pi(u)(x) \wedge \neg A(x) \tag{3}$$

If $\iota(A)(u) = \{0\}$, introduce the literal

$$\neg\exists x.\ \pi(u)(x) \wedge A(x) \tag{4}$$

If $\iota(A)(u) = \{0,1\}$, we do introduce no conjuncts.
Edges. Let $u_1, u_2 \in U$ (we allow $u_1 = u_2$) and let $f \in \mathcal{F}$. If $\iota(f)(u_1, u_2) = \{1\}$, introduce the must-edge R_1-literal

$$\neg\exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x,y) \tag{5}$$

If $\iota(f)(u_1, u_2) = \{0\}$, introduce the no-edge R_1-literal

$$\neg\exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x,y) \tag{6}$$

If $\iota(f)(u) = \{0,1\}$, we introduce no conjuncts.

Define formula $\mu(S)$ as the conjunction of all formulas (1), (2), (3), (4), (5), (6), introduced as described above. We next show $\gamma^*_{\mathrm{F}}(\mu(S)) = \gamma^*(S)$. In both directions, we establish the following property of the homomorphism h from $S^\sharp$ to S:

$$h(u^\sharp) = u \ \text{ iff } \ S^\sharp \models \pi(u)(u^\sharp) \tag{7}$$

Direction $\gamma^*_{\mathrm{F}}(\mu(S)) \supseteq \gamma^*(S)$. Let $S^\sharp \in \gamma^*(S)$. Then $S^\sharp \sqsubseteq^h S$ for some homomorphism h. We establish that (7) holds for h. For $A \in \mathcal{A}_1$, we have $\{\iota^\sharp(u^\sharp)(A)\} = \iota(u)(A)$, so $\models A^{\alpha(A)}(u^\sharp)$. Therefore, $\models \pi(u)(u^\sharp)$, which establishes (7).

We next show $S^\sharp \models C$ for each conjunct C of $\mu(S)$.

1) Consider $C \equiv \exists x.\pi(u)(x)$ for some u. Because h is a surjection, $h(u^\sharp) = u$ for some $u^\sharp$, so $\models \pi(u)(u^\sharp)$, and $\models C$.

2) Consider $C \equiv \neg\exists x.P(x)$ for the cube $P(x)$ distinct from all cubes $\pi(u)(x)$. Consider any $u^\sharp \in U^\sharp$. Then $\models \pi(h(u^\sharp))((^\sharp u))$, and $P(x)$ and $\pi(h(u^\sharp))(x)$ are distinct cubes, so $\neg \models P(u^\sharp)$. Therefore, $\models C$.

3) Consider $C \equiv \neg\exists x.\pi(u)(x) \wedge \neg A(x)$ for some $A \in \mathcal{A} \setminus \mathcal{A}_1$. Then $\iota(A)(u) = \{1\}$. Consider any $u^\sharp$. If $\neg \models \pi(u)(u^\sharp)$, then clearly $\models C$. If $\models \pi(u)(u^\sharp)$, then $h(u^\sharp) = u$ by (7), and because h is a homomorphism, $\iota^\sharp(u^\sharp) = 1$, so $\neg \models \neg A(u^\sharp)$, so again $\models C$.

4) Consider $C \equiv \neg\exists x.\pi(u)(x) \wedge A(x)$ for some $A \in \mathcal{A} \setminus \mathcal{A}_1$. Analogously to the previous case, $\iota(A)(u) = \{0\}$. Consider any $u^\sharp$. If $\models \pi(u)(u^\sharp)$, then $h(u^\sharp) = u$, and because h is a homomorphism, $\iota^\sharp(u^\sharp) = 0$, so $\neg \models A(u^\sharp)$ and thus $\models C$.

5) Consider $C \equiv \neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x,y)$. Then $\iota(f)(u_1, u_2) = \{1\}$. Consider any ${u_1}^\sharp, {u_2}^\sharp \in U^\sharp$. If $\neg \models \pi(u_1)({u_1}^\sharp)$ or $\neg \models \pi(u_2)({u_2}^\sharp)$, then $\models C$. Suppose $\models \pi(u_1)({u_1}^\sharp)$ and $\models \pi(u_1)({u_1}^\sharp)$. Then $h({u_1}^\sharp) = u_1$ and $h({u_2}^\sharp) = u_2$ by (7), and h is a homomorphism so $\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) = 1$. Then $\neg \models \neg f({u_1}^\sharp, {u_2}^\sharp)$ so $\models C$.

6) Consider $\neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x,y)$. Analogously to the previous case, $\iota(f)(u_1, u_2) = 0$; for any ${u_1}^\sharp, {u_2}^\sharp \in U^\sharp$, if $\models \pi(u_1)({u_1}^\sharp)$ and $\models \pi(u_1)({u_1}^\sharp)$ then $h({u_1}^\sharp) = u_1$ and $h({u_2}^\sharp) = u_2$, so $\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) = 0$, $\neg \models f({u_1}^\sharp, {u_2}^\sharp)$ so $\models C$.

Direction $\gamma^*_{\mathrm{F}}(\mu(S)) \subseteq \gamma^*(S)$. Let $S^\sharp \in \gamma^*_{\mathrm{F}}(\mu(S))$, then all conjuncts of $\mu(S)$ are true in $S^\sharp$. We show that $S^\sharp \sqsubseteq^h S$ where h is defined in the following way. Consider any $u^\sharp \in U^\sharp$. There is exactly one cube $C(x)$ such that $\models C(u^\sharp)$. Moreover, because $\mu(S)$ contains $\neg\exists x.P(x)$ for cubes $P(x)$ other than $\pi(u)(x)$, the cube $C(x)$ is of the form $\pi(u)(x)$ for some $u \in U$. Define $h(u^\sharp) = u$. This defines the function h. By construction, (7) holds. Furthermore, h is surjective: for each $u \in U$, the conjunct $\exists x.\pi(u)(x)$ is in $\mu(S)$, so there exists $u^\sharp$ such that $\pi(u)(u^\sharp)$ and thus $h(u^\sharp) = u$. We next show that h is a homomorphism.

1) Let us show

$$\{\iota^\sharp(A)(u^\sharp) \mid h(u^\sharp) = u\} \subseteq \iota(A)(u)$$

for all $A \in \mathcal{A}$ and for all $u \in U$. Consider $A \in \mathcal{A}_1$ and $u^\sharp$ such that $h(u^\sharp) = u$. Then $\models \pi(u)(u^\sharp)$, so $\models A^{\alpha(A)}(u^\sharp)$, which implies $\iota^\sharp(A)(u^\sharp) \in \iota(A)(u)$. Next, consider $A \in \mathcal{A} \setminus \mathcal{A}_1$. If $\iota(A)(u) = \{0,1\}$ the property trivially holds. Consider $\iota(A)(u) = \{1\}$. Then $\neg\exists x.\pi(u)(x) \wedge \neg A(x)$ occurs in $\mu(S)$. Therefore, if $h(u^\sharp) = u$, then $\models A(u^\sharp)$, otherwise the conjunct would be false. Therefore, $\iota^\sharp(u^\sharp) = 1 \in \iota(A)(u)$. The case $\iota(A)(u) = \{0\}$ is analogous: $\neg\exists x.\pi(u)(x) \wedge A(x)$ occurs in $\mu(S)$, so if $h(u^\sharp) = u$ then $\models A(u^\sharp)$, and $\iota^\sharp(u^\sharp) = 0 \in \iota(A)(u)$.

2) Let us show

$$\{\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) \mid h(u^\sharp) = u_1 \wedge h({u_2}^\sharp = u_2\} \subseteq \iota(f)(u_1, u_2) \tag{8}$$

for all $f \in \mathcal{F}$ and $u_1, u_2 \in U$. This is similar to 1). If $\iota(f) = \{0,1\}$, the inclusion trivially holds. Consider the case $\iota(f)(u_1, u_2) = \{1\}$. Then $\neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x,y)$ occurs in $\mu(S)$. Suppose that $h({u_1}^\sharp) = u_1$ and $h({u_2}^\sharp) = u_2$. Then $\models \pi(u_1)({u_1}^\sharp)$ and $\models \pi(u_2)({u_2}^\sharp)$, so $\models f(u_1, u_2)$ as well, otherwise the conjunct would be false. Therefore, $\iota^\sharp({u_1}^\sharp, {u_2}^\sharp) = 1$ and the inclusion (8) holds. The case $\iota(f)(u_1, u_2) = \{0\}$ is analogous: $\neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x,y)$ occurs in $\mu(S)$, so if $h({u_1}^\sharp) = u_1$ and $h({u_2}^\sharp) = u_2$, then $\models \pi(u_1)({u_1}^\sharp)$ and $\models \pi(u_2)({u_2}^\sharp)$ so $\neg \models f(u_1, u_2)$ and $\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) = 0$. The inclusion (8) holds, and $S^\sharp \sqsubseteq^h S$.

Because every structure S has a corresponding equivalent formula $\mu(S)$, we conclude $\mathsf{models}[T_1] \subseteq \mathsf{models}[R_1]$. To conclude $\mathsf{models}[T_1] \supseteq \mathsf{models}[R_1]$, we show that μ is surjective: every canonical conjunction F of R_1-formulas is equal to $\mu(S)$ for some structure S.

Let F be a canonical conjunction of R_1-literals. For each cube $P(x)$ such that $\exists x.P(x)$ occurs in F, let $u_{P(x)}$ be a distinct element. Let U be the set of all such elements $u_{P(x)}$. Property 2 of Definition 10 ensures that U is non-empty. Let α be such that $P(x) = \bigwedge_{A \in \mathcal{A}_1} A(x)^{\alpha(A)}$. Then define $\iota(A)(u_{P(x)}) = \{\alpha(A)\}$ for all $A \in \mathcal{A}_1$. For $A \in \mathcal{A} \setminus \mathcal{A}_1$, define $\iota(A)(u_{P(x)})$ as $\{1\}$ if $\neg\exists x.P(x) \wedge \neg A(x)$ occurs in F, as $\{0\}$ if $\neg\exists x.P(x) \wedge A(x)$ occurs in F, and as $\{0,1\}$ otherwise. Such definition of $\iota(u_{P(x)})(A)$ is possible because of the Property 4 of Definition 10. Analogously, using Property 5 of Definition 10, for each $f \in \mathcal{F}$, define $\iota(f)(u_{P_1(x)}, u_{P_2(x)})$ as $\{1\}$ if $\neg\exists xy.P_1(x) \wedge P_2(y) \wedge \neg f(x,y)$ occurs in F, as $\{0\}$ if $\neg\exists xy.P_1(x) \wedge P_2(y) \wedge f(x,y)$ occurs in F, and as $\{0,1\}$ otherwise. Let $S = \langle U, \iota \rangle$. To show $F = \mu(S)$, recall first that we use an abstract view of the syntax that takes into account associativity, commutativity and idempotence of conjunction. It therefore suffices to show that F and $\mu(S)$ contain the same *set* of conjuncts. It is easy to see that each conjunct of $\mu(S)$ occurs in F. The converse is also straightforward by Definition 10.

We conclude that μ is surjective, and $\mathsf{models}[R_1] \subseteq \mathsf{models}[T_1]$, which completes the proof.

Although this fact is not needed for the proof, we remark that μ is also injective, so μ is, in fact, a bijection between the set 3-STRUCT and the set of canonical conjunctions of R_1-literals. ∎

We proceed to show that a syntactically richer class of formulas defines the same set of constraints as R_1-formulas.

Definition 12 (R_2-formulas) *An R_2-formula is a disjunction of (not necessarily canonical) conjunctions of R_1-literals.*

The proof of the following Lemma 13 provides a normalization algorithm that converts every conjunction of R_1-literals into an equivalent disjunction of canonical conjunctions of R_1-literals.

Lemma 13 *Each conjunction of R_1-literals can be written as an equivalent R_1-formula.*

Proof. Consider an arbitrary, not-necessarily canonical, conjunction F of R_1-literals. We show how to transform F into an equivalent disjunction of *canonical* R_1-literals. The idea is to transform each conjunction into a disjunction of multiple conjunctions to ensure that all properties in Definition 10 are satisfied. We perform the following transformations as long as some property of Definition 10 is violated.
Property 1. If both $\exists x.P(x)$ and $\neg\exists x.P(x)$ occur, use the rule $Q \wedge \neg Q \rightarrow \mathsf{false}$ and eliminate the entire conjunction from the disjunction of conjunctions. If none of $\exists x.P(x)$ and $\neg\exists x.P(x)$ occur, use the rule $\mathsf{true} \rightarrow Q \vee \neg Q$ to introduce the missing $P(x)$, and then distribute the disjunction to the top level of the formula.
Property 2. First ensure that Property 1 holds. If the resulting conjunction contains no conjuncts of the form $\exists x.P(x)$, then the conjunction contains a conjunct $\neg\exists x.P(x)$ for every $P(x)$ a cube over $\mathcal{A}_1$. Therefore, the entire conjunction is false and can be eliminated from the disjunction of conjunctions.
Property 3. First ensure that Property 1 holds. Then, if the literal $\neg\exists x.P(x)$ occurs in the conjunction, remove from the conjunction all literals containing $P(x)$. Such literals are of the form $\neg\exists x.P(x) \wedge F_1(x)$ for some $F_1(x)$, $\neg\exists x\exists y.P(x) \wedge F_1(x,y)$, for some $F_2(x,y)$, or $\neg\exists x\exists y.P(y) \wedge F_1(x,y)$, for some $F_1(x,y)$; all these literals are implied by $\neg\exists x.P(x)$ so removing them yields an equivalent formula.
Property 4. If both conjuncts $\neg\exists x.P(x) \wedge A(x)$ and $\neg\exists x.P(x) \wedge \neg A(x)$ occur, replace them with the equivalent conjunct $\neg\exists x.P(x)$.
Property 5. If both

$$\neg\exists x\exists y.\ P_1(x) \wedge P_2(y) \wedge f(x,y)$$

and

$$\neg\exists x\exists y.\ P_1(x) \wedge P_2(y) \wedge \neg f(x,y)$$

occur, replace them with

$$(\neg\exists x.P_1(x)) \ \vee\ (\neg\exists y.P_2(y)),$$

then propagate the disjunction to the top level of the formula. ∎

Lemma 13 implies that R_2-formulas, although a syntactically a richer class, are no more expressive than R_1-formulas, hence Corollary 14.

Corollary 14 $\mathsf{models}[R_2] = \mathsf{models}[R_1]$

Proof. $\mathsf{models}[R_1] \subseteq \mathsf{models}[R_2]$ because R_2 is a richer class of formulas. Conversely, let $\mathcal{S}^\sharp \in \mathsf{models}[R_2]$. Then $\mathcal{S}^\sharp = \gamma^*_{\mathrm{F}}(F)$ for some R_2-formula F. By Lemma 13, let F' be an R_1-formula obtained by transforming conjunctions of F into disjunctions of canonical conjunctions of R_1-literals. F' is an R_1 formula equivalent to F. Therefore, $\mathcal{S}^\sharp = \gamma^*_{\mathrm{F}}(F')$, and $\mathcal{S}^\sharp \in \mathsf{models}[R_1]$. ∎

Definition 15 (Positive Boolean Combination) *If $B(p_1,\ldots,p_n)$ is a formula built from $p_1,\ldots,p_n$ using $\wedge,\vee,\neg$, we say that p_i (for $1 \le i \le n$) occurs positively in $B(p_1,\ldots,p_n)$ iff p_i occurs under an even number of $\neg$ signs. We say that $B(p_1,\ldots,p_n)$ is a positive boolean combination iff each of $p_1,\ldots,p_n$ occur positively in $B(p_1,\ldots,p_n)$.*

Definition 16 (R_3-formulas) *An R_3-formula is a positive boolean combination of R_1-literals.*

Lemma 17 states that R_2-formulas are simply the disjunctive normal forms of R_3-formulas.

Lemma 17 *Every R_3-formula is equivalent to an R_2-formula.*

Proof. Let F be an R_3-formula. Then the disjunctive normal form of F is an R_2-formula. ∎

Corollary 18 $\mathsf{models}[R_3] = \mathsf{models}[R_2]$

Proof. By Lemma 17. ∎

In the sequel we observe that replacing cubes over $\mathcal{A}_1$ in the definition of R_1-literals with boolean combinations over $\mathcal{A}_1$ does not change the set of expressible sets of two-valued structures. Definition 19 generalizes Definition 9.

Definition 19 (R_4-literals) *Let $B_1(x), B_2(y)$ range over arbitrary boolean combinations of elements of $\mathcal{A}_1$, let $Q(x)$ range over disjunctions of literals of the form $A(x)$ and the form $\neg A(x)$ for $A \in \mathcal{A} \setminus \mathcal{A}_1$, and let $g(x,y)$ range over disjunctions of literals of the form $f(x,y)$ and $\neg f(x,y)$ where $f \in \mathcal{F}$.*

An R_4-literal is a formula of one of the following forms:

1. $\exists x.B_1(x)$
2. $\neg\exists x.B_1(x) \wedge Q(x)$
3. $\neg\exists x\exists y.B_1(x) \wedge B_2(y) \wedge g(x,y)$

Definition 20 *An R_4-formula is a positive boolean combination of R_4-literals.*

Lemma 21 $\mathsf{models}[R_4] = \mathsf{models}[R_3]$

$$\exists x.B_1(x) \vee B_2(x) \;\rightarrow\; (\exists x.B_1(x)) \vee (\exists x.B_2(x))$$

$$\neg\exists x.B_1(x) \vee B_2(x) \;\rightarrow\; (\neg\exists x.B_1(x)) \wedge (\neg\exists x.B_2(x))$$

$$\neg\exists x.\ (B_1(x) \vee B_2(x)) \wedge Q(x) \;\rightarrow$$
$$(\neg\exists x.B_1(x) \wedge Q(x)) \wedge (\neg\exists x.B_2(x) \wedge Q(x))$$

$$\neg\exists x.\ B_1(x) \wedge (Q_1(x) \vee Q_2(x)) \;\rightarrow$$
$$(\neg\exists x.B_1(x) \wedge Q_1(x)) \wedge (\neg\exists x.B_1(x) \wedge Q_2(x))$$

$$\neg\exists x\exists y.\ (B_{11}(x) \vee B_{12}(x)) \wedge B_2(y) \wedge g(x,y) \rightarrow$$
$$\neg\exists x\exists y.\ B_{11}(x) \wedge B_2(y) \wedge g(x,y)\ \wedge$$
$$\neg\exists x\exists y.\ B_{12}(x) \wedge B_2(y) \wedge g(x,y)$$

$$\neg\exists x\exists y.\ B_1(x) \wedge (B_{21}(y) \vee B_{22}(y)) \wedge g(x,y) \rightarrow$$
$$\neg\exists x\exists y.\ B_1(x) \wedge B_{21}(y) \wedge g(x,y)\ \wedge$$
$$\neg\exists x\exists y.\ B_1(x) \wedge B_{22}(y) \wedge g(x,y)$$

$$\neg\exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge (g_1(x,y) \vee g_2(x,y)) \rightarrow$$
$$\neg\exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge g_1(x,y)\ \wedge$$
$$\neg\exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge g_2(x,y)$$

Figure 1: Transforming R_4-literals into R_1-literals

Proof. Note that a formula of the form $\neg\exists x.B_1(x)$ is equivalent to the formula $\neg\exists x.B_1(x) \wedge (A(x) \vee \neg A(x))$, which is of the form $\neg\exists x.B_1(x) \wedge Q(x)$. Therefore, R_4 is a richer class than R_3, so $\mathsf{models}[R_4] \supseteq \mathsf{models}[R_3]$. To show the converse, we transform each R_4-literal into a positive boolean combination of R_1-literals.

First, transform each boolean combination $B(x)$ (and $B(y)$) of $\mathcal{A}_1$ predicates into canonical disjunctive normal form, so that each $B(x)$ is a disjunction of cubes. Then apply rules in Figure 1 to decompose R_4-literals into R_1-literals. ∎

By eliminating the top-level negation from R_4-literals we obtain R_5-literals, which use universal quantifiers.

Definition 22 (R_5-literal) *Let $B_1(x), B_2(y)$ be variables denoting arbitrary boolean combinations of elements of $\mathcal{A}_1$, let $Q_P(x)$ denote* conjunctions *of literals of the form $A(x)$ and of the form $\neg A(x)$ for $A \in \mathcal{A} \setminus \mathcal{A}_1$, and let $g_P(x,y)$ denote* conjunctions *of literals of the form $f(x,y)$ and of the form $\neg f(x,y)$ for $f \in \mathcal{F}$.*

An R_5-literal is a formula of one of the following forms:

1. $\exists x.B_1(x)$
2. $\forall x.\ B_1(x) \Rightarrow Q_P(x)$
3. $\forall x\forall y.\ B_1(x) \wedge B_2(y) \Rightarrow g_P(x,y)$

Definition 23 *An R_5-formula is a positive boolean combination of R_5-literals.*

Lemma 24 $\mathsf{models}[R_5] = \mathsf{models}[R_4]$

Proof. $\forall x.\ B_1(x) \Rightarrow Q_P(x)$ corresponds to $\neg\exists x.B_1(x) \wedge Q(x)$ with $Q_P = \neg Q$, whereas $\forall x\forall y.\ B_1(x) \wedge B_2(y) \Rightarrow g_P(x,y)$ corresponds to $\neg\exists x\exists y.B_1(x) \wedge B_2(y) \wedge g(x,y)$ with $g_P = \neg g$. ∎

In the end we introduce R_6-formulas. Like heap abstractions based on may-edges, R_6-formulas implicitly indicate the absence of edges by specifying the set of possible endpoints for each edge.

Definition 25 (R_6-literals) *Let $B_1(x), B_2(y)$ denote arbitrary boolean combinations of elements of $\mathcal{A}_1$, let $Q_P(x)$ denote conjunctions of literals $A(x)$ and $\neg A(x)$ for $A \in \mathcal{A}\setminus\mathcal{A}_1$, and let f denote elements of $\mathcal{F}$.*

An R_6-literal is a formula of one of the following forms:

1. $\exists x.B_1(x)$ *(node existence)*
2. $\forall x.\ B_1(x) \Rightarrow Q_P(x)$ *(node properties)*
3. $\forall x\forall y.\ B_1(x) \wedge f(x,y) \Rightarrow B_2(y)$ *(may-edges)*
4. $\forall x\forall y.\ B_1(x) \wedge B_2(y) \Rightarrow f(x,y)$ *(must-edges)*

Definition 26 *An R_6-formula is a positive boolean combination of R_6-literals.*

Lemma 27 $\mathsf{models}[R_6] = \mathsf{models}[R_5]$

Proof. Observe first that the may-edge literal

$$\forall x\forall y.\ B_1(x) \wedge f(x,y) \Rightarrow B_2(y)$$

is equivalent to

$$\forall x\forall y.\ B_1(x) \wedge \neg B_2(y) \Rightarrow \neg f(x,y) \tag{9}$$

which is an R_5-literal. Conversely, every R_5 literal can be shown to be equivalent to a conjunction of may-edge and must-edge R_6-literals using the transformation

$$\begin{aligned}&\forall x\forall y.\ B_1(x) \wedge \neg B_2(y) \Rightarrow g_1(x,y) \wedge g_2(x,y) \rightarrow\\ &\quad \forall x\forall y.\ B_1(x) \wedge \neg B_2(y) \Rightarrow g_1(x,y)\ \wedge\\ &\quad \forall x\forall y.\ B_1(x) \wedge \neg B_2(y) \Rightarrow g_2(x,y)\end{aligned} \tag{10}$$

∎

The following Corollary 28 summarizes the results on different representations of constraints corresponding to three-valued structures.

Corollary 28

$$\mathsf{models}[T_1] =$$
$$\mathsf{models}[R_1] = \mathsf{models}[R_2] = \mathsf{models}[R_3] =$$
$$\mathsf{models}[R_4] = \mathsf{models}[R_5] = \mathsf{models}[R_6]$$

3.1 Closure under Disjunction and Conjunction

By definition, the syntactic class of R_3-formulas is closed under disjunction and conjunction. As the Corollary 29 below observes, this provides a way to compute the (disjunction and) conjunction of three-valued structures.

Corollary 29 *The family of sets* $\mathsf{models}[T_1]$ *is closed under union and intersection.*

Proof. The closure under union is trivial because union of sets of three-valued structures corresponds to the union of their models. For the closure under intersection, consider two sets of three-valued structures $\mathcal{S}_1$ and $\mathcal{S}_2$. Let F_1 be an R_3 formula such that $\gamma^*_{\mathrm{F}}(F_1) = \gamma^*(\mathcal{S}_1)$ and F_2 an R_3 formula such that $\gamma^*_{\mathrm{F}}(F_2) = \gamma^*(\mathcal{S}_2)$. Then $F_1 \wedge F_2$ is also an R_3 formula, and the set of three-valued structures corresponding to $F_1 \wedge F_2$ denotes the desired intersection. ∎

4 Three-Valued Structures with Tight Concretization

This section examines the constraints that arise from the meaning of sets of three-valued structures under *tight* concretization. These constraints are slightly more expressive than constraints in Section 3, as Section 4.2 shows. Interestingly, the added expressive power is just enough to make the constraints in this section closed under *all* boolean operations (Section 4.1). These closure properties are in contrast to the properties of constraints in Section 3, which are closed only under union and intersection. The closure under boolean operations allows, for example, reducing the implication of constraints to the satisfiability of constraints.

The structure of this section mirrors the structure of Section 3. We start by defining the interpretation of three-valued structures under tight concretization.

The following definition corresponds to [48, Definition 3.6], [52, Chapter 7]. Compared to our Definition 4 of Section 3, the only difference is the use of "=" instead of "$\supseteq$" in the condition on *1.* on $\iota(A)$ and the condition *2.* on $\iota(f)$.

Definition 30 (Tight Concretization) *Let $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$ be a two-valued structure, let $S = \langle U, \iota \rangle$ be a three-valued structure, and let $h : U^\sharp \to U$ be a surjective total function. We write $S^\sharp \sqsubseteq_T^h S$ iff*

1. *for every $A \in \mathcal{A}$ and $u \in U$:*

$$\iota(A)(u) = \{\iota^\sharp(A)(u^\sharp) \mid h(u^\sharp) = u\}$$

2. *for every $f \in \mathcal{F}$ and $u_1, u_2 \in U$:*

$$\iota(f)(u_1, u_2) = \{ \ \iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) \mid h({u_1}^\sharp) = u_1 \ \wedge \ h({u_2}^\sharp) = u_2 \}$$

We write $S^\sharp \sqsubseteq_T S$ iff there exists a surjective total function h such that $S^\sharp \sqsubseteq_T^h S$, and in that case we call h a homomorphism. The tight concretization of a three-valued structure S, is given by:

$$\gamma_T(S) = \{S^\sharp \mid S^\sharp \sqsubseteq_T S\}$$

We extend γ_T to γ_T^* that acts on sets of three-valued structures so that the set denotes a disjunction:

$$\gamma_T^*(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} \gamma_T(S)$$

Definition 31 (Tight Concretization Definability) *The set of sets of two-valued structures definable via three-valued structure with tight concretization is defined by:*

$$\mathsf{models}[T_2] = \{\gamma_T^*(\mathcal{S}) \mid \mathcal{S} \text{ a finite set of } \mathcal{A}_1\text{-bounded three-valued structures}\}$$

TR_1-literals are used to build formulas that characterize $\mathsf{models}[T_2]$.

Definition 32 (TR_1-literal) *Let $P_1(x), P_2(x)$ range over cubes over $\mathcal{A}_1$, let A range over elements of $\mathcal{A} \setminus \mathcal{A}_1$, and let f range over $\mathcal{F}$. A TR_1-atomic-formula is a formula of one of the following forms:*

$\exists x.\ P_1(x)$
$\exists x.\ P_1(x) \wedge A(x)$
$\exists x.\ P_1(x) \wedge \neg A(x)$
$\exists x \exists y.\ P_1(x) \wedge P_2(y) \wedge f(x,y)$
$\exists x \exists y.\ P_1(x) \wedge P_2(y) \wedge \neg f(x,y)$

A TR_1-literal is a TR_1-atomic-formula or its negation.

TR_1-formulas satisfy syntactic invariants that make them isomorphic to three-valued structures under tight concretization.

Definition 33 (TR_1-formulas) *Let $P(x)$, $P_1(x)$, $P_2(y)$ denote cubes over $\mathcal{A}_1$. A* canonical conjunction *of TR_1 literals is a conjunction of TR_1-literals that satisfies the following conditions.*

1. *for each $P(x)$ a cube over $\mathcal{A}_1$, exactly one of the conjuncts $\exists x.P(x)$ and $\neg\exists x.P(x)$ occurs;*

2. *there is at least one cube $P(x)$ such that the conjunct $\exists x.P(x)$ occurs in the conjunction;*

3. *if the conjunct $\neg\exists x.P(x)$ occurs, then this conjunct is the only occurrence of the cube $P(x)$ (and the cube $P(y)$) in the conjunction;*

4. *for each cube $P(x)$ and $A \in \mathcal{A} \setminus \mathcal{A}_1$ such that $\exists x.P(x)$ occurs, exactly one of the following three conditions holds:*
   - *(a) $\neg\exists x.\ P(x) \wedge A(x)$ occurs in the conjunction,*
   - *(b) $\neg\exists x.\ P(x) \wedge \neg A(x)$ occurs in the conjunction,*
   - *(c) both $\exists x.\ P(x) \wedge A(x)$ and $\exists x.\ P(x) \wedge \neg A(x)$ occur in the conjunction;*

5. *for every two cubes $P_1(x)$ and $P_2(y)$ such that the conjuncts $\exists x.P_1(x)$ and $\exists x.P_2(x)$ occur, and every $f \in \mathcal{F}$, exactly one one of the following three conditions holds:*
   - *(a) $\neg\exists x\exists y.\ P_1(x) \wedge P_2(y) \wedge f(x,y)$ occurs in the conjunction;*
   - *(b) $\neg\exists x\exists y.\ P_1(x) \wedge P_2(y) \wedge \neg f(x,y)$ occurs in the conjunction;*
   - *(c) both $\exists x\exists y.\ P_1(x) \wedge P_2(y) \wedge f(x,y)$ and $\exists x\exists y.\ P_1(x) \wedge P_2(y) \wedge \neg f(x,y)$ occur in the conjunction.*

A TR_1-formula is a disjunction of canonical conjunctions of TR_1-literals.

The following Proposition 34 shows that TR_1 formulas capture precisely the meaning of three-valued structures under tight concretization. The proof of Proposition 34 is similar to the proof of Proposition 11, and is similarly straightforward.

Proposition 34 $\mathsf{models}[TR_1] = \mathsf{models}[T_2]$

Proof. The idea of this proof is similar to the idea of the proof of Proposition 11: the meaning of each bounded three-valued structure under the tight concretization is equal to the meaning of some canonical conjunction of TR_1-formulas, and conversely. Therefore, disjunctions of canonical conjunctions of TR_1-literals correspond to sets of bounded three-valued structures under the tight concretization.

We next give a function μ mapping each bounded three-valued structure S to a canonical conjunction of TR_1-literals $\mu(S)$. We show that S under tight concretization and $\mu(S)$ represent the same set of two-valued structures. Moreover, μ is surjective.

Let $S = \langle U, \iota \rangle$ be an $\mathcal{A}_1$-bounded three-valued structure. Define the formula $\mu(S)$ as the conjunction of the following TR_1-literals. Define π as in the proof of Proposition 11.

Node existence. For each $u \in U$, introduce the TR_1-literal

$$\exists x.\pi(u)(x) \tag{11}$$

For each remaining $\mathcal{A}_1$-cube $P(x)$, that is, for each cube $P(x)$ such that $\pi(u)(x) \neq P(x)$ for all $u \in U$, introduce the TR_1-literal

$$\neg\exists x.P(x) \tag{12}$$

Node properties. Let $u \in U$ and $A \in \mathcal{A} \setminus \mathcal{A}_1$. If $\iota(A)(u) = \{1\}$, introduce the TR_1-literal

$$\neg\exists x.\ \pi(u)(x) \wedge \neg A(x) \tag{13}$$

If $\iota(A)(u) = \{0\}$, introduce the literal

$$\neg\exists x.\ \pi(u)(x) \wedge A(x) \tag{14}$$

If $\iota(A)(u) = \{0, 1\}$, introduce the following two TR_1-literals

$$\begin{aligned} &\exists x.\ \pi(u)(x) \wedge A(x) \\ &\exists x.\ \pi(u)(x) \wedge \neg A(x) \end{aligned} \tag{15}$$

Edges. Let $u_1, u_2 \in U$ (we allow $u_1 = u_2$) and let $f \in \mathcal{F}$. If $\iota(f)(u_1, u_2) = \{1\}$, introduce the TR_1-literal

$$\neg\exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x, y) \tag{16}$$

If $\iota(f)(u_1, u_2) = \{0\}$, introduce the TR_1-literal

$$\neg\exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x, y) \tag{17}$$

If $\iota(f)(u_1, u_2) = \{0, 1\}$, introduce the following two TR_1-literals

$$\begin{aligned} &\exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x, y) \\ &\exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x, y) \end{aligned} \tag{18}$$

Define formula $\mu(S)$ as the conjunction of all formulas (11), (12), (13), (14), (15), (16), (17), (18), introduced as described above. We next show $\gamma^*_{\mathrm{F}}(\mu(S)) = \gamma^*(S)$. As in the proof of Proposition 11, we establish and then use the property (7) for a homomorphism h from $S^\sharp$ to S.

Direction $\gamma^*_{\mathrm{F}}(\mu(S)) \supseteq \gamma^*(S)$. Let $S^\sharp \in \gamma^*(S)$. Then $S^\sharp \sqsubseteq^h S$ for some homomorphism h. We establish that (7) holds for h. For $A \in \mathcal{A}_1$, we have $\{\iota^\sharp(u^\sharp)(A)\} = \iota(u)(A)$. Then $\models A^{\alpha(A)}(u^\sharp)$. Therefore, the conjunction $\models \pi(u)(u^\sharp)$, which establishes (7). We next show $S^\sharp \models C$ for each conjunct C of $\mu(S)$.

1) Consider $C \equiv \exists x.\pi(u)(x)$ for some u. Because h is a surjection, $h(u^\sharp) = u$ for some $u^\sharp$, so $\models \pi(u)(u^\sharp)$, and the conjunct $\models C$.

2) Consider $C \equiv \neg\exists x.P(x)$ for the cube $P(x)$ distinct from all cubes $\pi(u)(x)$. Consider any $u^\sharp \in U^\sharp$. Then $\models \pi(h(u^\sharp)((^\sharp u))$, and $P(x)$ and $\pi(h(u^\sharp))(x)$ are distinct cubes, so $\neg \models P(u^\sharp)$. Therefore, $\models C$.

3) Consider $C \equiv \neg\exists x.\pi(u)(x) \wedge \neg A(x)$ for some $A \in \mathcal{A} \setminus \mathcal{A}_1$. This means that $\iota(A)(u) = \{1\}$. Consider any $u^\sharp$. If $\neg \models \pi(u)(u^\sharp)$, then $\models C$. If $\models \pi(u)(u^\sharp)$, then $h(u^\sharp) = u$ by (7), and because h is a homomorphism, $\iota^\sharp(u^\sharp) = 1$, so $\neg \models \neg A(u^\sharp)$. Hence $\models C$.

4) Consider $\neg\exists x.\pi(u)(x) \wedge A(x)$ for some $A \in \mathcal{A} \setminus \mathcal{A}_1$. Analogously to the previous case, $\iota(A)(u) = \{0\}$. Consider any $u^\sharp$. If $\pi(u)(u^\sharp)$, then $h(u^\sharp) = u$, and because h is a homomorphism, $\iota^\sharp(u^\sharp) = 0$, so $\neg \models A(u^\sharp)$. Hence $\models C$.

5) Consider conjuncts (15). These conjuncts occur only when $\iota(A)(u) = \{0, 1\}$. By the definition of tight concretization, there exists $u^\sharp \in U^\sharp$ such that $h(u^\sharp) = \pi(u)$ and $\iota^\sharp(A)(u^\sharp) = 1$. By property (7) of h, $\models \pi(u)(u^\sharp)$ and thus $\models \exists x.\pi(u)(x) \wedge A(x, y)$. Analogously, by the definition of tight concretization there exist $v^\sharp \in U^\sharp$ such that $h(v^\sharp) = u$, and $\iota^\sharp(A)(v^\sharp) = 0$, so $\models \exists x.\ \pi(u)(x) \wedge \neg A(x)$.

6) Consider $C \equiv \neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x, y)$. Then $\iota(f)(u_1, u_2) = \{1\}$. Consider any ${u_1}^\sharp, {u_2}^\sharp \in U^\sharp$. If $\neg \models \pi(u_1)({u_1}^\sharp)$ or $\neg \models \pi(u_1)({u_1}^\sharp)$, we have $\models C$. Suppose $\pi(u_1)({u_1}^\sharp)$ and $\pi(u_2)({u_2}^\sharp)$. Then $h({u_1}^\sharp) = u_1$ and $h({u_2}^\sharp) = u_2$; h is a homomorphism so $\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) = 1$, $\neg \models \neg f({u_1}^\sharp, {u_2}^\sharp)$ and $\models C$.

7) Consider $C \equiv \neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x, y)$. Analogously to the previous case, $\iota(f)(u_1, u_2) = 0$; for any $u^\sharp \in U^\sharp$, if $\models \pi(u_1)({u_1}^\sharp)$ and $\models \pi(u_2)({u_2}^\sharp)$, then $h({u_1}^\sharp) = u_1$ and $h({u_2}^\sharp) = u_2$, so $\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) = 0$, $\neg \models f({u_1}^\sharp, {u_2}^\sharp)$ and $\models C$.

8) Consider conjuncts (18). These conjuncts occur only when $\iota(f)(u_1, u_2) = \{0, 1\}$. By the definition of tight concretization, there exist $u_1^\sharp, u_2^\sharp \in U^\sharp$ such that $h(u_1^\sharp) = u_1$, $h(u_2^\sharp) = u_2$, and $\iota^\sharp(f)(u_1^\sharp, u_2^\sharp) = 1$. By property (7) of h, $\models \pi(u_1)(u_1^\sharp)$ and $\models \pi(u_2)(u_2^\sharp)$, and thus $\models \exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x, y)$. Analogously, by the definition of tight concretization, there exist $v_1^\sharp, v_2^\sharp \in U^\sharp$ such that $h(v_1^\sharp) = u_1$, $h(v_2^\sharp) = u_2$, and $\iota^\sharp(f)(v_1^\sharp, v_2^\sharp) = 0$, so $\models \exists x\exists y.\ \pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x, y)$.

Direction $\gamma^*_{\mathrm{F}}(\mu(S)) \subseteq \gamma^*(S)$. Let $S^\sharp \in \gamma^*_{\mathrm{F}}(()\mu(S))$, then all conjuncts of $\mu(S)$ are true in $S^\sharp$. We show that $S^\sharp \sqsubseteq^h S$ where h is defined in the same way as in the proof of the Proposition 11, so (7) holds and is surjective. We show that the homomorphism conditions of Definition 30 are satisfied for h.

1) Let us show

$$\{\iota^\sharp(A)(u^\sharp) \mid h(u^\sharp) = u\} = \iota(A)(u)$$

for all $A \in \mathcal{A}$ and for all $u \in U$. Consider $A \in \mathcal{A}_1$ and any $u^\sharp$ such that $h(u^\sharp) = u$. Then $\pi(u)(u^\sharp)$, so $A^{\alpha(A)}(u^\sharp)$, which implies $\{\iota^\sharp(A)(u^\sharp)\} = \iota(A)(u)$. Moreover, because $\exists x.\pi(u)(x)$ holds, the left-hand side side is a non-empty set, so it is equal to $\iota(A)(u)$.

Next, consider $A \in \mathcal{A} \setminus \mathcal{A}_1$. Consider first the case $\iota(A)(u) = \{1\}$. Then $\neg\exists x.\pi(u)(x) \wedge \neg A(x)$ occurs in $\mu(S)$. Therefore, if $h(u^\sharp) = u$, then $A(u^\sharp)$ holds, otherwise the conjunct would be false. Therefore, $\iota^\sharp(u^\sharp) = 1$. The left-hand side is non-empty so the equality holds.

The case $\iota(A)(u) = \{0\}$ is analogous: $\neg\exists x.\pi(u)(x) \wedge A(x)$ occurs in $\mu(S)$, so if $h(u^\sharp) = u$ then $A(u^\sharp)$ holds, so $\iota^\sharp(u^\sharp) = 0$ and the left-hand side is non-empty so the equality holds.

If $\iota(A)(u) = \{0, 1\}$ then the conjuncts (15) hold. Therefore, there exists a node $u^\sharp \in U^\sharp$ such that $h(u^\sharp) = u$ and $\iota^\sharp(A)(u^\sharp) = 1$, and there exists a node $v^\sharp \in U^\sharp$ such that $h(v^\sharp) = u$ and $\iota^\sharp(A)(u^\sharp) = 0$. The left hand-side is a set containing both 0 and 1, so it is equal to $\{0, 1\}$.

2) Let us show

$$\{\iota^\sharp(f)({u_1}^\sharp, {u_2}^\sharp) \mid h(u^\sharp) = u_1 \wedge h({u_2}^\sharp) = u_2\} = \iota(f)(u_1, u_2) \tag{19}$$

for all $f \in \mathcal{F}$ and $u_1, u_2 \in U$. This is similar to 1).

Consider first the case $\iota(f)(u_1, u_2) = \{1\}$. Then $\neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge \neg f(x, y)$ occurs in $\mu(S)$. Suppose that $h({u_1}^\sharp) = u_1$ and ${u_2}^\sharp = u_2$. Then $\pi(u_1)({u_1}^\sharp)$ and $\pi(u_2)({u_2}^\sharp)$ hold, so $f(u_1, u_2)$ must hold as well, otherwise the conjunct would be false. Therefore, $\iota^\sharp({u_1}^\sharp, {u_2}^\sharp) = 1$. Moreover, because of the conjunct $\exists x.\pi(u_1)(x)$ and the conjunct $\exists x.\pi(u_2)(x)$, the left-hand side is non-empty, so it is equal to $\{1\}$.

The case $\iota(f)(u_1, u_2) = \{0\}$ is analogous: $\neg\exists x\exists y.\pi(u_1)(x) \wedge \pi(u_2)(y) \wedge f(x, y)$ occurs in $\mu(S)$, so if $h({u_1}^\sharp) = u_1$ and ${u_2}^\sharp = u_2$, then $\pi(u_1)({u_1}^\sharp)$ and $\pi(u_2)({u_2}^\sharp)$ so $f(u_1, u_2)$ is false and $\iota^\sharp({u_1}^\sharp, {u_2}^\sharp) = 0$. Moreover, because of the conjunct $\exists x.\pi(u_1)(x)$ and the conjunct $\exists x.\pi(u_2)(x)$, the left-hand side is non-empty, so it is equal to $\{0\}$.

Finally, consider the case $\iota(A)(u) = \{0, 1\}$. Then the conjuncts (18) hold in $S^\sharp$. Because the first conjunct holds, there exist $u_1^\sharp$ and $u_2^\sharp$ such that $h(u_1^\sharp) = u_1$, $h(u_2^\sharp) = u_2$ hold and and $\sharp\iota(f)(u_1^\sharp, u_2^\sharp) = 1$. Therefore, 1 belongs to the left-hand side of 19. Similarly, because the second conjunct holds, 0 belongs to the right-hand side of of 19. Therefore (19) holds.

We conclude that $S^\sharp \sqsubseteq_T^h S$. Because every structure S has a corresponding equivalent formula $\mu(S)$, we have $\mathsf{models}[T_2] \subseteq \mathsf{models}[TR_1]$. To conclude $\mathsf{models}[T_2] \supseteq \mathsf{models}[TR_1]$, we show that μ is surjective.

Let F be a canonical conjunction of TR_1-literals. For each cube $P(x)$ such that $\exists x.P(x)$ occurs in F, let $u_{P(x)}$ be a distinct element. Let U be the set of all such elements $u_{P(x)}$. Property 2 of Definition 33 ensures that U is non-empty. Let α be such that $P(x) = \bigwedge_{A \in \mathcal{A}_1} A(x)^{\alpha(A)}$. Then define $\iota(A)(u_{P(x)}) = \{\alpha(A)\}$ for all $A \in \mathcal{A}_1$. For $A \in \mathcal{A} \setminus \mathcal{A}_1$, define $\iota(A)(u_{P(x)})$ as $\{1\}$ if $\neg\exists x.P(x) \wedge \neg A(x)$ occurs in F, as $\{0\}$ if $\neg\exists x.P(x) \wedge A(x)$ occurs in F, and as $\{0, 1\}$ otherwise. Such definition of $\iota(u_{P(x)})(A)$ is possible because of the Property 4 of Definition 33. Analogously, using Property 5 of Definition 33, for each $f \in \mathcal{F}$, define $\iota(f)(u_{P_1(x)}, u_{P_2(x)})$ as $\{1\}$ if $\neg\exists xy.P_1(x) \wedge P_2(y) \wedge \neg f(x, y)$ occurs in F, as $\{0\}$ if $\neg\exists xy.P_1(x) \wedge P_2(y) \wedge f(x, y)$ occurs in F, and as $\{0, 1\}$ otherwise. Let $S = \langle U, \iota\rangle$. To show $F = \mu(S)$, it suffices to show that F and $\mu(S)$ contain the same set of conjuncts. It is easy to see that each conjunct of $\mu(S)$ occurs in F. The converse is also straightforward by Definition 33.

We conclude that μ is surjective, and $\mathsf{models}[T_2] \supseteq \mathsf{models}[TR_1]$, which completes the proof.

It is easy to see that μ is, in fact, a bijection between the set 3-STRUCT and the set of canonical conjunctions of TR_1-literals. ■

As in Section 3, we proceed to show that we can permit a richer syntactic structure without changing the expressive power of constraints.

Definition 35 (TR_2-formulas) *A TR_2-formula is a disjunction of conjunctions of TR_1-literals.*

The following Lemma 36 is analogous to Lemma 13; it shows that any conjunction of TR_1 literals can be transformed into an equivalent disjunction of canonical conjunctions of TR_1 literals.

Lemma 36 *Each conjunction of TR_1-literals can be written as an equivalent TR_1-formula.*

Proof. Consider an arbitrary, not-necessarily canonical, conjunction F of TR_1-literals. We show how to transform F into an equivalent disjunction of *canonical* conjunctions of TR_1-literals. The idea is to transform conjunctions into disjunctions of multiple conjunctions to ensure that all properties in the Definition 33 are satisfied. We perform the following transformations as long as any of the properties in Definition 33 is violated.

Property 1. If both $\exists x.P(x)$ and $\neg\exists x.P(x)$ occur, the entire conjunction is false and we eliminate it from the disjunction of conjunctions. If none of the $\exists x.P(x)$ and $\neg\exists x.P(x)$ use the rule $\mathsf{true} \to (\exists x.P(x)) \vee (\neg\exists x.P(x))$ and then distribute the disjunction to the top level of the formula.

Property 2. First ensure that Property 1 holds. If the resulting conjunction contains no conjuncts of the form $\exists x.P(x)$, then the conjunction contains a conjunct $\neg\exists x.P(x)$ for every $P(x)$ a cube over $\mathcal{A}_1$. Therefore, the entire conjunction is false and can be eliminated from the disjunction of conjunctions.

Property 3. Suppose that the literal $\neg\exists x.P(x)$ occurs in the conjunction. If the conjunction contains a literal of one of the forms $\exists x.P(x) \wedge Q(x)$, $\exists x\exists y.P(x) \wedge Q(x, y)$, or $\exists x\exists y.P(y) \wedge Q(x, y)$, then the entire conjunction is contradictory and may be omitted from the disjunction of conjunctions. If there are no such conjunctions, then (as in the proof of Lemma 13) remove all , literals of forms $\neg\exists x.P(x) \wedge Q(x)$, $\neg\exists x\exists y.P(x) \wedge Q(x, y)$, and $\neg\exists x\exists y.P(y) \wedge Q(x, y)$. because they are implied by $\neg\exists x.P(x)$.

Property 4. If both a literal and its negation occur in the conjunction, the entire conjunction is false. Hence, we can assume that (a) and (c) do not occur simultaneously and (b) and (c) do not occur simultaneously. To ensure that (a) and (b) do not occur simultaneously, use the replacement rule

$$(\neg\exists x.P(x) \wedge A(x)) \wedge (\neg\exists x.P(x) \wedge \neg A(x)) \;\to\; \neg\exists x.P(x)$$

and then ensure again the Property 3. We have thus shown how to ensure that no two of the cases (a), (b), (c) hold simultaneously. To ensure that at least one of the cases (a), (b), (c), holds, use the fact that $\neg p \vee \neg q \vee (p \wedge q)$ is a tautology, and apply the rule

$$\begin{aligned} \mathsf{true} \to\; & (\neg\exists x.P(x) \wedge A(x)) \vee \\ & (\neg\exists x.P(x) \wedge \neg A(x)) \vee \\ & (\exists x.P(x) \wedge A(x)) \wedge (\exists x.P(x) \wedge \neg A(x)) \end{aligned}$$

Then propagate the disjunction to the top level of the formula. Then ensure that no two cases apply simultaneously, as described previously.

Property 5. Ensuring Property 5 is analogous to ensuring Property 4. If both a literal and its negation occur in the conjunction, the entire conjunction is false. Hence, we can assume that (a) and (c) do not occur simultaneously and (b) and (c) do not occur simultaneously. To ensure that (a) and (b) do not occur simultaneously, use the replacement rule

$$\begin{gathered} (\neg\exists x\exists y.P_1(x) \wedge P_2(y) \wedge f(x, y)) \wedge \\ (\neg\exists x\exists y.P_1(x) \wedge P_2(y) \wedge \neg f(x, y)) \;\to \\ (\neg\exists x.P_1(x)) \vee (\neg\exists y.P_2(y)) \end{gathered}$$

and then ensure again Property 3. We have thus shown how to ensure that no two of the cases (a), (b), (c) hold simultaneously. To ensure that at least one of the cases (a), (b), (c), holds, use the fact that $\neg p \vee \neg q \vee (p \wedge q)$ is a tautology, and apply the rule

$$\begin{aligned}\mathsf{true} \to\ & (\neg\exists x\exists y.P_1(x) \wedge f(x,y)) \vee \\ & (\neg\exists x\exists y.P_1(x) \wedge P_2(y) \wedge \neg f(x,y)) \vee \\ & (\exists x\exists y.P_1(x) \wedge f(x,y)) \wedge \\ & (\exists x\exists y.P_1(x) \wedge P_2(y) \wedge \neg f(x,y))\end{aligned}$$

Then propagate the disjunction to the top level of the formula. Then ensure that no two cases apply simultaneously, as described previously. ■

Corollary 37 $\mathsf{models}[TR_2] = \mathsf{models}[TR_1]$

Proof. Every TR_1-formula is a TR_2-formula. Conversely, let F be a TR_2-formula. Then F is a disjunction of conjunctions of TR_1. By Lemma 36, transform each conjunction of F into a disjunction of canonical conjunctions of TR_1 literals. The result is a TR_1-formula. ■

TR_3-formulas remove the disjunctive normal form requirement on TR_2-formulas.

Definition 38 (TR_3-formulas) *TR_3-formula is a boolean combination of TR_1-atomic-formulas.*

TR_2-formulas are the disjunctive normal forms of TR_3-formulas.

Lemma 39 *Every TR_3 formula is equivalent to a TR_2 formula.*

Proof. Let F be a TR_3 formula. Then the disjunctive normal form of F is a TR_2 formula. ■

Corollary 40 $\mathsf{models}[TR_3] = \mathsf{models}[TR_2]$

Proof. Every TR_2 formula is a TR_3 formula, so $\mathsf{models}[TR_3] \supseteq \mathsf{models}[TR_2]$. The converse $\mathsf{models}[TR_3] \subseteq \mathsf{models}[TR_2]$ follows from Lemma 39. ■

Analogously to R_4 formulas in Section 3, we introduce TR_4 formulas that allow using boolean combinations of more complex atomic formulas.

Definition 41 (TR_4-formulas) *Let $B_1(x), B_2(y)$ be range over arbitrary boolean combinations of elements of $\mathcal{A}_1$, let $Q(x)$ range over disjunctions of literals of form $A(x)$ and $\neg A(x)$ where $A \in \mathcal{A} \setminus \mathcal{A}_1$, and let $g(x,y)$ range over disjunctions of literals of the form $f(x,y)$ and $\neg f(x,y)$ where $f \in \mathcal{F}$.*

A TR_4-atomic-formula is a formula of one of the following forms:

1. $\exists x.\ B_1(x)$
2. $\exists x.\ B_1(x) \wedge Q(x)$
3. $\exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge g(x,y)$

A TR_4-literal is a TR_4-atomic-formula or its negation. A TR_4-formula is a boolean combination of TR_4-atomic-formulas.

Lemma 42 $\mathsf{models}[TR_4] = \mathsf{models}[TR_3]$

$$\begin{aligned}
& \exists x.B_1(x) \vee B_2(x) \ \to\ (\exists x.B_1(x)) \vee (\exists x.B_2(x)) \\
& \exists x.\ (B_1(x) \vee B_2(x)) \wedge Q(x) \ \to \\
& \quad (\exists x.B_1(x) \wedge Q(x)) \vee (\exists x.B_2(x) \wedge Q(x)) \\
& \exists x.\ B_1(x) \wedge (Q_1(x) \vee Q_2(x)) \ \to \\
& (\exists x.B_1(x) \wedge Q_1(x)) \vee (\exists x.B_1(x) \wedge Q_2(x)) \\
& \exists x\exists y.\ (B_{11}(x) \vee B_{12}(x)) \wedge B_2(y) \wedge g(x,y) \to \\
& \quad \exists x\exists y.\ B_{11}(x) \wedge B_2(y) \wedge g(x,y) \vee \\
& \quad \exists x\exists y.\ B_{12}(x) \wedge B_2(y) \wedge g(x,y) \\
& \exists x\exists y.\ B_1(x) \wedge (B_{21}(y) \vee B_{22}(y)) \wedge g(x,y) \to \\
& \quad \exists x\exists y.\ B_1(x) \wedge B_{21}(y) \wedge g(x,y) \vee \\
& \quad \exists x\exists y.\ B_1(x) \wedge B_{22}(y) \wedge g(x,y) \\
& \exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge (g_1(x,y) \vee g_2(x,y)) \to \\
& \quad \exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge g_1(x,y) \wedge \\
& \quad \exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge g_2(x,y)
\end{aligned}$$

Figure 2: Transforming TR_4-literals into TR_1-literals

Proof. Each formula of the form $\neg\exists x.B_1(x)$ is equivalent to the formula $\neg\exists x.B_1(x) \wedge (A(x) \vee \neg A(x))$, which is of the form $\neg\exists x.B_1(x) \wedge Q(x)$. Therefore, TR_4 is a richer class than TR_3, so $\mathsf{models}[TR_4] \supseteq \mathsf{models}[TR_3]$. To show the converse, transform each TR_4-literal into a boolean combination of TR_1-literals.

First, transform each boolean combination $B(x)$ (and $B(y)$) of $\mathcal{A}_1$ predicates into canonical disjunctive normal form, so that each $B(x)$ is a disjunction of cubes. Then apply rules in Figure 2 to decompose TR_4-literals into TR_1-literals. ■

Instead of existential quantifiers, we may use atomic formulas that contain universal quantifiers.

Definition 43 (TR_5-formulas) *Let $B_1(x), B_2(y)$ denote arbitrary boolean combinations of elements of $\mathcal{A}_1$, let $Q_P(x)$ denote conjunctions of literals of the form $A(x)$ and $\neg A(x)$ for $A \in \mathcal{A} \setminus \mathcal{A}_1$, and let $g_P(x,y)$ denote conjunctions of literals $f(x,y)$ and $\neg f(x,y)$ for $f \in \mathcal{F}$.*

An TR_5-atomic-formula is a formula of one of the following forms:

1. $\forall x.B_1(x)$
2. $\forall x.B_1(x) \Rightarrow Q_P(x)$
3. $\forall x.\ B_1(x) \Rightarrow \forall y.\ B_2(y) \Rightarrow g_P(x,y)$

A TR_5-formula is a boolean combination of TR_5-atomic-formulas.

Lemma 44 $\mathsf{models}[TR_5] = \mathsf{models}[TR_4]$

Proof. For each TR_5-atomic-formula there exists a corresponding equivalent TR_4-formula, and for each TR_4-atomic-formula there exists a corresponding equivalent TR_5-formula.

The mapping from TR_5-atomic-formulas to TR_4-formulas is in Figure 3, the mapping of TR_4-atomic-formulas

TR_5 − atomic formula	TR_4 − formula
$\forall x.\ B_1(x)$	$\neg\exists x.\ \neg B_1(x)$
$\forall x.\ B_1(x) \Rightarrow (L_1(x) \wedge \ldots \wedge L_k(x))$	$\neg\exists x.\ B_1(x) \wedge (\overline{L}_1(x) \vee \ldots \vee \overline{L}_k(x))$
$\forall x.\ B_1(x) \Rightarrow \forall y.\ B_2(y) \Rightarrow (L_1(x,y) \wedge \ldots L_k(x,y))$	$\neg\exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge (\overline{L}_1(x,y) \vee \ldots \overline{L}_k(x,y))$

Figure 3: Mapping TR_5-atomic-formulas to TR_4-formulas

TR_4 − atomic formula	TR_5 − formula
$\exists x.\ B_1(x)$	$\neg\forall x. \neg B_1(x)$
$\exists x.\ B_1(x) \wedge (L_1(x) \vee \ldots L_k(x))$	$\neg\forall x.\ B_1(x) \wedge (\overline{L}_1(x) \wedge \ldots \wedge \overline{L}_k(x))$
$\exists x\exists y.\ B_1(x) \wedge B_2(y) \wedge (L_1(x,y) \wedge \ldots \wedge L_k(x,y))$	$\neg\forall x.\ B_1(x) \Rightarrow \forall y.\ B_2(y) \Rightarrow (\overline{L}_1(x,y) \vee \ldots \vee \overline{L}_k(x,y))$

Figure 4: Mapping TR_4-atomic-formulas to TR_5-formulas

to TR_5-formulas is in Figure 4. We use the notation $\overline{L}$ to denote L_1 if L is of the form $\neg L_1$ for some L_1, and $\neg L$ if L is not of the form $\neg L_1$ for some L_1. ∎

The following Corollary 45 summarizes the results on different representations of constraints corresponding to three-valued structures with tight concretization.

Corollary 45

$$\mathsf{models}[T_2] = \mathsf{models}[TR_1] = \mathsf{models}[TR_2] =$$
$$\mathsf{models}[TR_3] = \mathsf{models}[TR_4] = \mathsf{models}[TR_5]$$

Definition 46 (Boolean Shape Analysis Constraints) *We call the set of sets* $\mathsf{models}[T_2]$ boolean shape analysis constraints.

4.1 Closure under Boolean Operations

By definition, TR_3-formulas are closed under all boolean operations.

Corollary 47 *The family of sets* $\mathsf{models}[T_2]$ *forms a boolean algebra of sets which is a subalgebra of the boolean algebra of all subsets of* 2-STRUCT.

As an example consequence of closure under all boolean set operations we obtain the following proposition.

Proposition 48 *There is an algorithm that constructs, given two finite sets of bounded three-valued structures $\mathcal{S}_1$ and $\mathcal{S}_2$, a finite set of bounded three-valued structures $\mathcal{S}_3$ such that:*

$$\gamma^*_T(\mathcal{S}_1) \subseteq \gamma^*_T(\mathcal{S}_2) \ \ \textit{iff} \ \ \gamma^*_T(\mathcal{S}_3) = \emptyset$$

Similarly, the equivalence of two three-valued structures reduces to the satisfiability.

Proposition 49 *There is an algorithm that constructs, given two finite sets of bounded three-valued structures $\mathcal{S}_1$ and $\mathcal{S}_2$, a finite set of bounded three-valued structures $\mathcal{S}_3$ such that:*

$$\gamma^*_T(\mathcal{S}_1) = \gamma^*_T(\mathcal{S}_2) \ \ \textit{iff} \ \ \gamma^*_T(\mathcal{S}_3) = \emptyset$$

4.2 Relationship with Non-Tight Concretization

In Proposition 50 below we observe that three-valued structures with *tight concretization* (Definition 30) are at least as expressive as three-valued structures with *concretization* (Definition 4).

Proposition 50 $\mathsf{models}[T_2] \supseteq \mathsf{models}[T_1]$.

Proof. By definition, every R_4-formula is a TR_4-formula, so $\mathsf{models}[TR_4] \supseteq \mathsf{models}[R_4]$. Therefore,

$$\mathsf{models}[T_2] = \mathsf{models}[TR_4] \supseteq \mathsf{models}[R_4] = \mathsf{models}[T_1]$$

∎

Proposition 50 implies that, even if we work with the interpretation of three-valued structures under concretization, we can convert three-valued structures into boolean shape analysis constraints and check for entailment or equivalence of the original constraints via satisfiability of the, richer, boolean shape analysis constraints. In fact, the Proposition 51 below shows that boolean shape analysis constraints $\mathsf{models}[T_2]$ are the smallest extension of the constraints $\mathsf{models}[T_1]$ which have this desirable property.

Proposition 51 $\mathsf{models}[T_2]$ *is the smallest superset of* $\mathsf{models}[T_1]$ *that is closed under all boolean operations.*

Proof. $\mathsf{models}[T_2] \supseteq \mathsf{models}[T_1]$ by Proposition 50, and $\mathsf{models}[T_2]$ is closed under all boolean operations by Corollary 47. Because $\mathsf{models}[T_2] = \mathsf{models}[TR_4]$ and $\mathsf{models}[T_1] = \mathsf{models}[R_4]$, it remains to show that every TR_4 formula is (equivalent to) a boolean combination of some R_4-literals. By definition, TR_4-formulas are boolean combinations of TR_4 atomic formulas, so it suffices to show that each TR_4 atomic formula is a boolean combination of R_4 literals. That is certainly true, in fact, it suffices to use at most one negation of an R_4 literal to obtain any TR_4 literal. ∎

4.3 Node Splitting

Given a three-valued structure $S = \langle U, \iota \rangle$, it is desirable if $\iota(A)(u) \in \{\{0\},\{1\}\}$ for all $u \in U$ and $A \in \mathcal{A}$. This property holds if all predicates are abstraction predicates, that is, if $\mathcal{A}_1 = \mathcal{A}$. The following Proposition 52 shows that we can always assume that $\mathcal{A}_1 = \mathcal{A}$ if the syntactic class of formulas is sufficiently rich.

Proposition 52 *Every TR_4-formula with the set of abstraction predicates $\mathcal{A}_1 \subseteq \mathcal{A}$ is also a TR_4 formula with the set of abstraction predicates $\mathcal{A}_1 = \mathcal{A}$.*

Proof. Observe that, in the atomic formula $F_1(x) \equiv \exists x.\ B_1(x) \wedge Q(x)$ of the Definition 41, the subformula $B_2(x) \equiv B_1(x) \wedge Q(x)$ is a boolean combination of predicates from $\mathcal{A}$, so $F_1(x)$ is of the form $\exists x.B_2(x)$ for a boolean combination of predicates from $\mathcal{A}$. ∎

Note that the converse of Proposition 52 is not true. For example, if $A, A' \in \mathcal{A} \setminus \mathcal{A}_1$ then the property $\neg\exists x.\ A(x) \wedge \neg A'(x)$, which correlates two non-abstraction predicates is a TR_4-formula with the set of abstraction predicates $\mathcal{A}$, but is not equivalent to any TR_4-formula with the set of abstraction predicates $\mathcal{A}_1$.

Definition 53 (Split Form) *Let F_1 be a TR_4-formula with the set of abstraction predicates $\mathcal{A}_1 \subseteq \mathcal{A}$. By Proposition 52 and Corollary 45, let F_2 be a TR_1-formula with the set of abstraction predicates $\mathcal{A}$ such that F_2 is equivalent to F_1. We call F_2 the* split form *of F_1.*

Letting $\mathcal{A}_1 = \mathcal{A}$ in Definition 41, we obtain the following Corollary 54.

Corollary 54 (Split Form Formulas) *The set of split forms of TR_4 formulas is precisely the set of boolean combinations of formulas of the form*

1. $\exists x.\ B_1(x)$
2. $\exists x \exists y.\ B_1(x) \wedge B_2(y) \wedge g(x,y)$

where $B_1(x)$, $B_2(y)$ are boolean combinations of literals of the form $A(x)$ and $A(y)$ for $A \in \mathcal{A}$, and $g(x,y)$ ranges over disjunctions of literals of the form $f(x,y)$ and $\neg f(x,y)$ for $f \in \mathcal{F}$.

5 Decidability of Independent Predicates

In this section we present decidability results for constraints expressed by three-valued structures under tight concretization. We show that satisfiability, entailment and equivalence of boolean shape analysis constraints are all decidable. Boolean shape analysis constraints (TR_1-formulas) are more expressive than R_1-formulas by Proposition 50, so we obtain decidability results for R_1-formulas as well.

Formulation of Decidability Problems We assume finite sets $\mathcal{A}$ and $\mathcal{F}$ of predicates. As a result, the number of non-isomorphic bounded three-valued structures, and, therefore, the number of non-equivalent R_1-formulas, is finite. Therefore, for fixed $\mathcal{A}$ and $\mathcal{F}$, a problem of the form:

> Given a TR_1-formula F, is F satisfiable?

is essentially finite and therefore trivially decidable. However, we are interested in having a single algorithm that would give decidability for any number of unary and binary predicates. Therefore, the size of sets $\mathcal{A}$ and $\mathcal{F}$ is part of the input to the decision procedure we are looking for. For example, we are interested in the questions of the form:

> Given sets $\mathcal{A}$ and $\mathcal{F}$ and a TR_1-formula F over predicates $\mathcal{A}$ and $\mathcal{F}$, is F satisfiable?

In this section we study such decidability questions for independent predicates, when the three-valued structures are interpreted over the entire set 2-STRUCT. Section 6.4 addresses the more general case where some of the predicates are defined using first-order formulas, which means that formulas are interpreted over a *subset* of 2-STRUCT.

Satisfiability of TR_1-formulas over 2-STRUCT is decidable. In fact, the proof of the following Lemma 55 shows that every disjunct of a TR_1-formula has a small model in 2-STRUCT.

Lemma 55 *Let F be a canonical conjunction of TR_1-literals and let the number of cubes $P(x)$ over $\mathcal{A}_1$ such that $\exists x.P(x)$ occurs in F be n. Then there exists a two-valued structure $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$ such that $|U^\sharp| = 2n$ and F is true in $S^\sharp$.*

Proof. Let $S = \langle U, \iota \rangle$ be the structure that corresponds to F by the proof of Proposition 34. Let $U = \{u_1, \ldots, u_n\}$. Define $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$ as follows. Let $U^\sharp = \{u_1^\sharp, u_2^\sharp, \ldots, u_{2n}^\sharp\}$. In the sequel we define $\iota^\sharp$ so that $S^\sharp \sqsubseteq_T^h S$ where h is given by

$$h(u_{2i-1}^\sharp) = u_i$$
$$h(u_{2i}^\sharp) = u_i$$

for $1 \le i \le n$. By definition, h is surjective.

Define $\iota^\sharp(A)$ for $A \in \mathcal{A}_1$ as follows. Let $1 \le i \le n$. Then $\iota(u_i) = \{0\}$ or $\iota(u_i) = \{1\}$. If $\iota(u_i) = \{0\}$, define

$$\iota^\sharp(A)(u_{2i-1}^\sharp) = \iota^\sharp(A)(u_{2i}^\sharp) = 0$$

If $\iota(u_i) = \{1\}$, define

$$\iota^\sharp(A)(u_{2i-1}^\sharp) = \iota^\sharp(A)(u_{2i}^\sharp) = 1$$

Define $\iota^\sharp(A)$ for $A \in \mathcal{A} \setminus \mathcal{A}_1$ as follows. Let $1 \le i \le n$. If $\iota(u_i) = \{0\}$, define

$$\iota^\sharp(A)(u_{2i-1}^\sharp) = \iota^\sharp(A)(u_{2i}^\sharp) = 0$$

If $\iota(u_i) = \{1\}$, define

$$\iota^\sharp(A)(u_{2i-1}^\sharp) = \iota^\sharp(A)(u_{2i}^\sharp) = 1$$

If $\iota(u_i) = \{0, 1\}$, define

$$\iota^\sharp(A)(u_{2i-1}^\sharp) = 0$$
$$\iota^\sharp(A)(u_{2i}^\sharp) = 1$$

Define $\iota^\sharp(f)$ for $f \in \mathcal{F}$ as follows. Let $1 \le i, j \le n$. If $\iota(f)(u_i, u_j) = \{0\}$, define

$$\iota^\sharp(f)(u_k^\sharp, u_l^\sharp) = 0$$

for $k \in \{2i-1, 2i\}$ and $l \in \{2j-1, 2j\}$. If $\iota(f)(u_i, u_j) = \{1\}$, define

$$\iota^\sharp(f)(u_k^\sharp, u_l^\sharp) = 1$$

for $k \in \{2i-1, 2i\}$ and $l \in \{2j-1, 2j\}$. If $\iota(f)(u_i, u_j) = \{0, 1\}$, define

$$\iota^\sharp(f)(u_k^\sharp, u_{2j-1}^\sharp) = 0$$
$$\iota^\sharp(f)(u_k^\sharp, u_{2j}^\sharp) = 1$$

for $k \in \{2i-1, 2i\}$.

It is straightforward to show $S^\sharp \sqsubseteq_T^h S$. Therefore, F holds in $S^\sharp$. ∎

Corollary 56 $\gamma_T^*(\mathcal{S}) = \emptyset$ *iff* $\mathcal{S} = \emptyset$.

Proof. By Lemma 55. ∎

We note that the construction of the model in Lemma 55 becomes even simpler if we assume that the formula F is in the split form. Corollary 56 then follows from the observation that if F has at least one disjunct then split form of F has at least one disjunct.

Corollary 57 *The following questions are decidable for sets* $\mathcal{S}_1, \mathcal{S}_2$ *of three-valued structures:*

1. $\gamma_T^*(\mathcal{S}_1) \subseteq \gamma_T^*(\mathcal{S}_2)$;
2. $\gamma_T^*(\mathcal{S}_1) = \gamma_T^*(\mathcal{S}_2)$.

Proof. By Corollary 56, Proposition 48, and Proposition 49. ∎

6 Structures with Defined Predicates

In this section we introduce the notion of a three-valued structure with defined predicates. Previous sections interpret three-valued structures and formulas over the set 2-STRUCT of all two-valued structures. In general, it is useful to interpret three-valued structures and formulas over some subset 2-CSTRUCT $\subseteq$ 2-STRUCT of *compatible* two-valued structures [49, Page 268].

6.1 Compatible Structures

We view structures with defined predicates as a way of defining a subset 2-CSTRUCT $\subseteq$ 2-STRUCT.

Definition 58 (Compatible Structures) *Let* $\mathcal{A}_2 \subseteq \mathcal{A}$ *be a set of* defined unary predicates *and* $\mathcal{F}_2 \subseteq \mathcal{F}$ *be a set of* defined binary predicates. *Let* 2-SEM-STRUCT $\subseteq$ 2-STRUCT *be the set of two-valued structures that satisfy the constraints of the semantics of the programming language. Next, for each* $A \in \mathcal{A}_2$*, and each two-valued structure* $S^\sharp \in$ 2-STRUCT *where* $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$*, let* $d_A(S^\sharp) : U^\sharp \to \{0,1\}$ *be a unary predicate. For each* $f \in \mathcal{A}_2$*, and each two-valued structure* $S^\sharp \in$ 2-STRUCT *where* $S^\sharp = \langle U^\sharp, \iota^\sharp \rangle$*, let* $d_f(S^\sharp) : (U^\sharp)^2 \to \{0,1\}$ *be a binary predicate. Define*

$$
\begin{aligned}
&\text{2-CSTRUCT} = \{S^\sharp = \langle U^\sharp, \iota^\sharp \rangle \mid \\
&\quad S^\sharp \in \text{2-SEM-STRUCT} \land \\
&\quad \bigwedge_{A \in \mathcal{A}_2} \forall u^\sharp \in U^\sharp.\ \iota^\sharp(A)(u^\sharp) = d_A(S^\sharp)(u^\sharp) \land \\
&\quad \bigwedge_{f \in \mathcal{F}_2} \forall {u_1}^\sharp, {u_2}^\sharp \in U^\sharp.\ \iota(f)({u_1}^\sharp, {u_2}^\sharp) = d_f(S^\sharp)({u_1}^\sharp, {u_2}^\sharp)\}
\end{aligned}
\tag{20}
$$

Definition 59 below introduces tight concretization with respect to compatible structures, in the natural way.

Definition 59 (Compatible Tight Concretization) *If* $\mathcal{S} \subseteq$ 3-STRUCT *is a set of three-valued structures, define*

$$c\gamma_T^*(\mathcal{S}) = \gamma_T^*(\mathcal{S}) \cap \text{2-CSTRUCT}$$

We then use $c\gamma_T^*$ to define the class of definable sets $\mathsf{models}[cT_2]$. With the results of Section 4.3 in mind, we let $\mathcal{A}_1 = \mathcal{A}$.

Definition 60 *The set of sets of compatible two-valued structures definable via three-valued structure with tight concretization is defined by:*

$$\mathsf{models}[cT_2] = \{c\gamma_T^*(\mathcal{S}) \mid \mathcal{S} \textit{ a finite set of } \mathcal{A}_1\textit{-bounded three-valued structures}\}$$

Lemma 61

$$\mathsf{models}[cT_2] = \{\mathcal{S}^\sharp \cap \text{2-CSTRUCT} \mid \mathcal{S}^\sharp \in \mathsf{models}[T_2]\}$$

Proof. Immediate by Definition 60 and Definition 59. ∎

6.2 Formulas for Compatible Structures

In Section 4 we have characterized sets of two-valued structures using formulas. We now characterize sets of *compatible* two-valued structures by conjoining the formulas with the *compatibility formula*.

Definition 62 (Compatibility Formula) *Let* ψ_0 *be a sentence that axiomatizes the set* 2-SEM-STRUCT*, so that:*

$$\mathsf{models}[\{\psi_0\}] = \text{2-SEM-STRUCT}$$

Let the value of each predicate $d_A(S^\sharp)$ *for* $A \in \mathcal{A}_2$ *be equal to the Tarskian semantics of some formula* $\psi_A(x)$ *in the structure* $S^\sharp$*:*

$$d_A(S^\sharp)(u^\sharp) = [\![\psi_A(x)]\!]^{S^\sharp} [x \mapsto u^\sharp]$$

and let the value of each predicate $d_f(S^\sharp)$ *for* $f \in \mathcal{F}_2$ *be equal to the Tarskian semantics of some formula* $\psi_f(x,y)$ *in the structure* $S^\sharp$*:*

$$d_f(S^\sharp)({u_1}^\sharp, {u_2}^\sharp) = [\![\psi_A(x)]\!]^{S^\sharp} [x \mapsto {u_1}^\sharp, y \mapsto {u_2}^\sharp]$$

Define the compatibility formula F_ψ *by:*

$$
\begin{aligned}
F_\psi \equiv\ & \psi_0 \land \\
& \bigwedge_{A \in \mathcal{A}_2} \forall x.\ A(x) \iff \psi_A(x) \land \\
& \bigwedge_{f \in \mathcal{F}_2} \forall x\, y.\ f(x,y) \iff \psi_f(x,y)
\end{aligned}
$$

For each class of formulas TR_i we introduce the corresponding class cTR_i by conjoining the formulas with F_ψ.

Definition 63 (Formulas for Compatible Structures) *For each* i *where* $1 \le i \le 5$*, let the set of* cTR_i *formulas be the set of all formulas* $B \land F_\psi$ *for* B *a* TR_i *formula.*

Lemma 64 below shows that compatibility formula defines precisely the subset of compatible two-valued structures.

Lemma 64 (Compatibility Formula is Correct)

$$\text{2-CSTRUCT} = \{S^\sharp \in \text{2-STRUCT} \mid [\![F_\psi]\!]^{S^\sharp} = 1\}$$

Proof. Immediate by Definition 58 and Definition 62. ∎

As a result, we obtain the following characterization of the constraints expressible using cTR_i formulas.

Lemma 65 *For each* i *where* $1 \le i \le 5$,

$$\mathsf{models}[cTR_i] = \{\mathcal{S}^\sharp \cap \text{2-CSTRUCT} \mid \mathcal{S}^\sharp \in \mathsf{models}[TR_i]\}$$

Proof. By Definition 63 and Lemma 64. ■

The following Corollary 66 states the desired correspondence between formulas and three-valued structures with defined predicates.

Corollary 66

$$\mathsf{models}[cT_2] = \mathsf{models}[cTR_1] = \mathsf{models}[cTR_2] =$$
$$\mathsf{models}[cTR_3] = \mathsf{models}[cTR_4] = \mathsf{models}[cTR_5]$$

Proof. From Lemma 61, Lemma 65, and Corollary 45. ■

6.3 Closure under Boolean Operations

We next show that, even in the presence of defined predicates, we can reduce the entailment and the equivalence of constraints to the satisfiability problem. This results follows from the closure under boolean operations. The results below generalize the results of Section 4.1.

Corollary 67 *The family of sets* $\mathsf{models}[cT_2]$ *forms a boolean algebra of sets which is a subalgebra of the boolean algebra of all subsets of* 2-CSTRUCT.

Proof. From Lemma 61, Lemma 65, and Corollary 47. ■

Proposition 68 *There is an algorithm that constructs, given two finite sets of three-valued structures* $\mathcal{S}_1$ *and* $\mathcal{S}_2$*, a finite set of three-valued structures* $\mathcal{S}_3$ *such that:*

$$c\gamma_T^*(\mathcal{S}_1) \subseteq c\gamma_T^*(\mathcal{S}_2) \quad \textit{iff} \quad c\gamma_T^*(\mathcal{S}_3) = \emptyset$$

Proposition 69 *There is an algorithm that constructs, given two finite sets of three-valued structures* $\mathcal{S}_1$ *and* $\mathcal{S}_2$*, a finite set of three-valued structures* $\mathcal{S}_3$ *such that:*

$$c\gamma_T^*(\mathcal{S}_1) = c\gamma_T^*(\mathcal{S}_2) \quad \textit{iff} \quad c\gamma_T^*(\mathcal{S}_3) = \emptyset$$

6.4 Decidability Properties

The following conditional result generalizes the idea of Section 5.

Corollary 70 *Let* $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$ *range over finite sets of three-valued structures with defined predicates. Assume that the question* $c\gamma_T^*(\mathcal{S}) = \emptyset$ *is decidable. Then the following questions are decidable as well:*

1. $c\gamma_T^*(\mathcal{S}_1) \subseteq c\gamma_T^*(\mathcal{S}_2)$;
2. $c\gamma_T^*(\mathcal{S}_1) = c\gamma_T^*(\mathcal{S}_2)$.

Proof. By Proposition 68 and Proposition 69. ■

We present an example of constraints for which the satisfiability question is decidable in [36]; other examples of decidable constraints can be formulated based on the techniques of logic L_r of [4] or based on monadic second-order logic of trees which is in the heart of the graph types approach [17, 28–31, 43].

7 Related Work

A parametric framework for shape analysis is presented in [49]. A systematic presentation of three-valued logic with equality is given in [44]. A description of three-valued logic analyzer is in [38], an extension to interprocedural analysis is in [47] and the use of shape analysis for program verification is demonstrated in [39]. Other shape analysis techniques include [16, 20, 21, 25, 33, 37, 43].

Our paper presents a contribution to the characterization of heap summaries by formulas, which is a promising direction of shape analysis that has been initiated in [34, 35, 46, 52]. Shape analysis constraints differ from regular graph constraints [34, 35] because shape analysis constraints characterize sets of objects by defining predicates, instead of using existential quantification over sets of objects. Logic L_r in [4] allows specifying reachability properties between local variables and is therefore appropriate for expressing certain classes of shape graphs. What L_r does not allow is defining a set of nodes A using some predicate and then stating further properties of objects in the set A, which is one of the main expressive features of three-valued structures.

Our work follows the line of shape analysis approaches which view program as transforming concrete graph structures [20, 21, 25, 33, 37, 43, 49]. An alternative approach is to identify each heap object using the set of paths that lead to the object [8, 16, 23]. Other notations for reasoning about the heap include spatial logic [10, 11, 26, 45] and alias types [50, 51].

It is possible to apply predicate abstraction techniques [2, 3, 22] to perform shape analysis; the view of three-valued structures as boolean combinations of constraints of certain form may be beneficial for this direction of work and enable easier application of representations such as binary decision diagrams [5, 41, 42].

A shape analysis tool must ultimately take into account the definitions of instrumentation predicates, which requires some form of theorem proving or decision procedures. [49, Page 272] uses rules based on Horn clauses for such reasoning, whereas [46] proposes the use of theorem provers. In this paper we have identified one component of the problem that is always decidable and useful: it is always possible to reduce entailment and equivalence problems to the satisfiability problem. In [36], we report a concrete example of constraints for which the satisfiability is decidable, the results in the present paper then imply that the entailment and the equivalence are decidable as well.

Researchers have proposed several program checking techniques based on dataflow analysis, symbolic execution, and abstract interpretation [6, 9, 12, 18, 19, 24, 40]. The primary strength of the shape analysis approach compared to the alternative approaches is the ability to perform sound and precise reasoning about dynamically allocated data structures.

The boolean algebra of state predicates and predicate transformers has been used successfully as the foundation of refinement calculus [1]. In this paper we have identified a particular subalgebra of the boolean algebra of all state predicates; we view this boolean algebra as providing the foundation of shape analysis.

8 Conclusions

We have characterized constraints used as dataflow facts of parametric shape analysis based on three-valued logic. Our characterization represents these dataflow facts as boolean combinations of formulas. The usual concretization semantics yields only positive boolean combinations. On the other hand, the tight concretization yields boolean shape analysis constraints, which are closed under all boolean combinations. Among the useful consequences of the closure of boolean shape analysis constraints under all boolean operations is the fact that the entailment and the equivalence of constraints is reducible to the satisfiability of constraints.

We view the results of this paper as a step in further understanding of the foundations of shape analysis. To make the connection with [49], this paper starts with three-valued structures and proceeds to characterize the structures using formulas. An alternative approach is to start with canonical formulas that express the desired properties and then explore efficient ways of representing and manipulating these formulas. We believe that the entire framework [49] can be reformulated using canonical forms of formulas instead of three-valued structures. We also expect that the idea of viewing dataflow facts as canonical forms of formulas is methodologically useful in general, especially for the analyses that verify complex program properties.

Acknowledgements The results of this paper were inspired in part by the discussions with Patrick Lam, Andreas Podelski, Thomas Reps, Mooly Sagiv, Greta Yorsh, and David Schmidt. We thank Patrick Lam for useful discussions and implementation of an early prototype for role analysis based on canonical forms of formulas in Fall 2001. We thank Andreas Podelski for discussions on using canonical formulas for shape analysis at MIT in Fall 2002, at the Dagstuhl seminar on Shape in Spring 2003, and at the visit of first author to Max-Planck Institute for Computer Science in Spring 2003. We thank Thomas Reps and Mooly Sagiv for numerous discussions of various aspects of shape analysis. We thank David Schmidt for discussions on shape analysis at the Dagstuhl Seminar on Shape and at MIT.

We thank Greta Yorsh for discussions on shape analysis at the Dagstuhl seminar on Shape as well as for providing many useful comments on an earlier version of this report.

References

[1] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus.* Springer-Verlag, 1998.

[2] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. ACM PLDI*, 2001.

[3] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Relative completeness of abstraction refinement for software model checking. In *TACAS'02*, volume 2280 of *LNCS*, page 158, 2002.

[4] Michael Benedikt, Thomas Reps, and Mooly Sagiv. A decidable logic for linked data structures. In *Proc. 8th ESOP*, 1999.

[5] Marc Berndl, Ondrej Lhoták, Feng Qian, Laurie Hendren, and Navindra Umanee. Points-to analysis using BDDs. In *PLDI 2003*, 2003.

[6] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A Static Analyzer for Large Safety-Critical Software. In *ACM PLDI*, San Diego, California, June 2003. ACM.

[7] Egon Börger and Robert Stärk. *Abstract State Machines.* Springer-Verlag, 2003.

[8] Marius Bozga, Radu Iosif, and Yassine Laknech. Storeless semantics and alias logic. In *ACM PEPM'03*, pages 55–65. ACM Press, 2003.

[9] William R. Bush, Jonathan D. Pincus, and David J. Sielaff. A static analyzer for finding dynamic programming errors. *Software—Practice & Experience*, 30(7):775–802, 2000.

[10] Cristiano Calcagno, Luca Cardelli, and Andrew D. Gordon. Deciding validity in a spatial logic for trees. In *ACM TLDI'02*, 2002.

[11] Cristiano Calcagno, Samin Ishtiaq, and Peter W. O'Hearn. Semantic analysis of pointer aliasing, allocation and disposal in hoare logic. In *Proc. 2nd International Conference on Principles and Practice of Declarative Programming*, 2000.

[12] Lori Clarke and Debra Richardson. Symbolic evaluation methods for program analysis. In *Program Flow Analysis: Theory and Applications*, chapter 9. Prentice-Hall, Inc., 1981.

[13] Edgar F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, 1970.

[14] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th POPL*, 1977.

[15] Chris J. Date. *An Introduction to Database Systems.* Addison-Wesley, 6 edition, 1995.

[16] Alain Deutsch. Interprocedural may-alias analysis for pointers: Beyond k-limiting. In *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, pages 230–241. ACM Press, 1994.

[17] Jacob Elgaard, Anders Møller, and Michael I. Schwartzbach. Compile-time debugging of C programs working on trees. In *Proc. 9th ESOP*, 2000.

[18] Dawson Engler, Benjamin Chelf, Andy Chou, and Seth Hallem. Checking system rules using system-specific, programmer-written compiler extensions. In *Proc. 4th USENIX OSDI*, 2000.

[19] Cormac Flanagan, K. Rustan M. Leino, Mark Lilibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended Static Checking for Java. In *Proc. ACM PLDI*, 2002.

[20] Pascal Fradet and Daniel Le Métayer. Shape types. In *Proc. 24th ACM POPL*, 1997.

[21] Rakesh Ghiya and Laurie Hendren. Is it a tree, a DAG, or a cyclic graph? In *Proc. 23rd ACM POPL*, 1996.

[22] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with pvs. In *Proc. 9th CAV*, pages 72–83, 1997.

[23] C. A. R. Hoare and He Jifeng. A trace model for pointers and objects. In *Proc. 13th ECOOP*, volume 1628 of *LNCS*, 1999.

[24] Gerard J. Holzmann. Static source code checking for user-defined properties. In *Proc. IDPT 2002, Pasadena, CA*, June 2002.

[25] Joseph Hummel, Laurie J. Hendren, and Alexandru Nicolau. A general data dependence test for dynamic, pointer-based data structures. In *Proc. ACM PLDI*, 1994.

[26] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. 28th ACM POPL*, 2001.

[27] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM TOSEM*, 11(2):256–290, 2002.

[28] Jacob L. Jensen, Michael E. Jørgensen, Nils Klarlund, and Michael I. Schwartzbach. Automatic verification of pointer programs using monadic second order logic. In *Proc. ACM PLDI*, Las Vegas, NV, 1997.

[29] Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. MONA implementation secrets. In *Proc. 5th International Conference on Implementation and Application of Automata*. LNCS, 2000.

[30] Nils Klarlund and Michael I. Schwartzbach. Graph types. In *Proc. 20th ACM POPL*, Charleston, SC, 1993.

[31] Nils Klarlund and Michael I. Schwartzbach. Graphs and decidable transductions based on edge constraints. In *Proc. 19th Colloquium on Trees and Algebra in Programming*, number 787 in LNCS, 1994.

[32] Stephen Cole Kleene. *Introduction to Metamathematics*. D. Van Nostrand Company, Inc., Princeton, New Jersey, 1952. fifth reprint, 1967.

[33] Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Proc. 29th POPL*, 2002.

[34] Viktor Kuncak and Martin Rinard. Typestate checking and regular graph constraints. Technical Report 863, MIT Laboratory for Computer Science, 2002.

[35] Viktor Kuncak and Martin Rinard. Existential heap abstraction entailment is undecidable. In *10th Annual International Static Analysis Symposium (SAS 2003)*, San Diego, California, June 11-13 2003.

[36] Viktor Kuncak and Martin Rinard. On cardinality constraints in shape analysis. Technical report, MIT CSAIL, August 2003.

[37] James R. Larus and Paul N. Hilfinger. Detecting conflicts between structure accesses. In *Proc. ACM PLDI*, Atlanta, GA, June 1988.

[38] Tal Lev-Ami. TVLA: A framework for kleene based logic static analyses. Master's thesis, Tel-Aviv University, Israel, 2000.

[39] Tal Lev-Ami, Thomas Reps, Mooly Sagiv, and Reinhard Wilhelm. Putting static analysis to work for verification: A case study. In *International Symposium on Software Testing and Analysis*, 2000.

[40] Francesco Logozzo. Class-level modular analysis for object oriented languages. In *Static Analysis, 10th International Symposium, SAS 2003, San Diego, CA, USA, June 11-13, 2003, Proceedings*, volume 2694 of *Lecture Notes in Computer Science*. Springer, 2003.

[41] Roman Manevich, G. Ramalingam, John Field, Deepak Goyal, and Mooly Sagiv. Compactly representing first-order structures for static analysis. In *Proc. 9th International Static Analysis Symposium*, pages 196–212, 2002.

[42] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer-Verlag (Berlin, Heidelberg, New York), 1998.

[43] Anders Møller and Michael I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.

[44] Flemming Nielson, Hanne Riis Nielson, and Mooly Sagiv. Kleene's logic with equality. *Information Processing Letters*, 80(3):131–137, 2001.

[45] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proc. CSL, Paris 2001*, volume 2142 of *LNCS*, 2001.

[46] Thomas Reps, Mooly Sagiv, and Greta Yorsh. Symbolic implementation of the best transformer. Technical Report TR-1468, University of Wisconsin, January 2003.

[47] Noam Rinetzky and Mooly Sagiv. Interprocedual shape analysis for recursive programs. In *Proc. 10th International Conference on Compiler Construction*, 2001.

[48] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. In *Proc. 26th ACM POPL*, 1999.

[49] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.

[50] Frederick Smith, David Walker, and Greg Morrisett. Alias types. In *Proc. 9th ESOP*, Berlin, Germany, March 2000.

[51] David Walker and Greg Morrisett. Alias types for recursive data structures. In *Workshop on Types in Compilation*, 2000.

[52] Greta Yorsh. Logical characterizations of heap abstractions. Master's thesis, Tel-Aviv University, March 2003.