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

Abstract. The parametric shape analysis framework of Sagiv, Reps, and Wilhelm [45,46] uses three-valued structures as dataflow lattice elements to represent sets of states at different program points. The recent work of Yorsh, Reps, Sagiv, Wilhelm [48,50] introduces a family of formulas in (classical, two-valued) logic that are isomorphic to three-valued structures [46] and represent the same sets of concrete states.

In this paper we introduce a larger syntactic class of formulas that has the same expressive power as the formulas in [48]. The formulas in [48] can be viewed as a normal form of the formulas in our syntactic class; we give an algorithm for transforming our formulas to this normal form. Our formulas make it obvious that the constraints are closed under all boolean operations and therefore form a boolean algebra. Our algorithm also gives a reduction of the entailment and the equivalence problems for these constraints to the satisfiability problem.

Keywords: Shape Analysis, Program Verification, Abstract Interpretation, Boolean Algebra, First-Order Logic, Model Checking

1 Introduction

Background. Shape analysis [9,12,15,16,20,22,27,32,37,46] is a technique for statically analyzing programs that manipulate dynamically allocated data structures, and is important for precise reasoning about programs written in modern imperative programming languages. Parametric shape analysis [45,46] is a framework that can be instantiated to provide a variety of precise shape analyses. We can describe this approach informally as follows. The concrete program state is a *two-valued structure*, that is, a finite relational structure $\langle U^\sharp, \iota^\sharp \rangle$, which maps, for example, a binary relation symbol r to a binary relation $\iota^\sharp(r) : U^\sharp \times U^\sharp \to \{0, 1\}$. To represent a potentially infinite set of concrete program states, [46] uses finite *three-valued structures*, which are relational structures in three-valued logic [24,38]. A three-valued structure $\langle U, \iota \rangle$ maps a binary relation symbol r

[*] 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.

[**] VMCAI'04. Version of November 7, 2003, 4:30pm

to a three-valued relation $\iota(r) : U \times U \to \{\{0\}, \{1\}, \{0,1\}\}$. Three-valued structures generalize the graphs used in several previous shape analyses [9, 22, 44]. The elements of the domain U of a three-valued structure $\langle U, \iota \rangle$ represent disjoint non-empty sets of objects. Given two such sets A and B, we can compute the three-valued relation by $\iota(r)(A, B) = \{\iota^\sharp(r)(a, b) \mid a \in A \wedge b \in B\}$. As observed in [48, 50], the fact $\iota(r)(A, B) = \{0\}$ means that the formula $\neg\exists x \exists y. A(x) \wedge B(y) \wedge r(x, y)$ holds on the two-valued structure $\langle U^\sharp, \iota^\sharp \rangle$. Similarly, the fact $\iota(r)(A, B) = \{1\}$ means that $\neg\exists x \exists y. A(x) \wedge B(y) \wedge \neg r(x, y)$ holds, whereas $\iota(r)(A, B) = \{0, 1\}$ means that both $\exists x \exists y. A(x) \wedge B(y) \wedge r(x, y)$ and $\exists x \exists y. A(x) \wedge B(y) \wedge \neg r(x, y)$ hold. As a result, any three-valued structure can be described by a corresponding formula in first-order logic [50]. In this paper we take a closer look at the class of formulas that arise when characterizing the meaning of three-valued structures. We characterize such formulas as the set of all boolean combinations of certain simple formulas, such as $\exists x \exists y. A(x) \wedge B(y) \wedge r(x, y)$ (see Definition 5). As a result, we establish that the meaning of three valued structures (under the tight concretization semantics [50, Chapter 7]) is closed under all boolean operations and therefore forms a boolean algebra.

Characterizing structures using formulas. The characterization of three-valued structures using formulas in first-order logic is presented for the first time in [48,50]. Section 3.1 of [48] explains that the semantics of general three-valued structures can represent the existence of graph coloring. As a result, first-order structures in general are not definable using first-order logic, but require the use of monadic second-order logic [48, Section 4]. However, an interesting class of three-valued structures can be represented using first-order logic [48, Section 3.2], in particular, this is the case for *bounded structures*. Two versions of the semantics for three-valued structures are of interest: the *standard concretization* [45, Definition 3.5], [48, Chapter 3] and the *tight concretization* [48, Chapter 7] (the later corresponding to the *canonical abstraction* [45, Definition 3.6]). One can view the *characteristic formulas for canonical abstraction* of [48, Chapter 7] as the starting point for the class of formulas in this paper: we show how to allow a richer syntactic class of formulas, and give an algorithm for converting these formulas to the characteristic formulas for canonical abstraction.

We have previously studied *regular graph constraints* [28,29], inspired by the semantics of role analysis [25–27]. Regular graph constraints abstract the notion of graph summaries where nodes do not have a unique abstraction criterion. In [28,29] we observe that such constraints can be equivalently characterized using graphs summaries and using existential monadic second-order logic formulas. Somewhat surprisingly, whereas the satisfiability of regular graph constraints is decidable [29, Section 2.4], the entailment and the equivalence of regular graph constraints are undecidable [29, Section 3], [28]. These properties of regular graph constraints are in contrast to the nice closure properties of the *boolean shape analysis constraints* of the present paper.

1.1 Contributions

Main result. The main result of this paper is a new syntactic class of formulas that characterize the meaning of three-valued structures under tight concretization. The new syntactic class is defined as the set of all boolean combinations of formulas of a certain form. The proof of the equivalence of the new syntactic class and previously introduced characteristic formulas for canonical abstraction [48] is a normalization algorithm that transforms formulas in our syntactic class to the characteristic formulas for canonical abstraction (which are isomorphic to three-valued structures). Our characterization immediately implies that the constraints expressible as the meaning of three-valued structures are closed under all boolean operations, we thus call them "boolean shape analysis constraints".

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 computations in dataflow analysis and can easily be computed even for three-valued structures (by taking the union of sets of three-valued structures). 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 $d \land c$ in the `then` branch, and $d \land \neg c$ 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. The implication problem is important in compositional shape analysis which uses assume/guarantee reasoning to show that a procedure conforms to its specification.

Decidability of constraints. The closure of boolean shape analysis constraints under boolean operations holds in the presence of arbitrary instrumentation predicates [46, Section 5]. What the particular choice of instrumentation predicates determines is whether the satisfiability problem for the constraints is decidable. If the satisfiability problem for three-valued structures with a particular choice of instrumentation predicates is decidable, our normalization algorithm yields an algorithm for the satisfiability problem of formulas in the richer syntactic class, which, by closure under boolean operations, gives an algorithm for deciding the entailment and the equivalence of boolean shape analysis constraints.

Consequences for program annotations. The ability to write program annotations can greatly improve the effectiveness of static analysis, but the representation of program properties in the program analysis is often different from the representation of program properties that is appropriate for program annotations. On the one hand, to synthesize invariants using 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, for example, by providing a translation from a logic-based annotation language to the analysis representation. The translation from the full first-order logic to three-valued structures is equivalent to first-order theorem proving, and is therefore undecidable. Because we restrict our attention to formulas of a particular form, we are able to find a (complete and sound) decision procedure for generating three-valued structures that have the same meaning as these formulas.[1] The existence of this information-preserving translation algorithm indicates that our formulas have the same expressive power as three-valued structures. Nevertheless, our formulas are more flexible than the direct use of three-valued structures (or formulas isomorphic to three-valued structures). For example, our formulas may use sets that are potentially intersecting or empty, while the summary nodes of three-valued structures represent disjoint, non-empty sets of nodes.

In addition to the benefits for writing program annotations, the richer syntactic class of formulas is potentially useful for analysis representations. A set of three-valued structures corresponds to a disjunctive normal form; alternative representations for three-valued structures may be more appropriate in some cases.

2 Preliminaries

We mostly follow the setup of [46]. 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 typical element $f \in \mathcal{F}$). For simplicity, we consider only unary and binary relation symbols, which are usually sufficient for modelling dynamically allocated structures. 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 2-STRUCT be the set of all two-valued structures. 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 3-STRUCT denote the set of all three-valued structures. If $S^\sharp$ is a two-valued structure and F a closed formula in first-order logic, then $[\![F]\!]^{S^\sharp} \in \{0,1\}$ denotes the truth-value of F in $S^\sharp$, and $\gamma^*_{\mathrm{F}}(F) = \{S^\sharp \in \text{2-STRUCT} \mid [\![F]\!]^{S^\sharp} = 1\}$ is the set of

[1] An alternative approach proposes the use of theorem provers to synthesize three-valued structures from arbitrary first-order formulas [40, 41, 49, 49], [48, Chapter 6].

models of F. If C is a set of formulas, then $\mathsf{models}[C] = \{\gamma^*_{\mathrm{F}}(F) \mid F \in C\}$ is the set of sets of models of formulas from C. Let $\mathcal{A}_1 \subseteq \mathcal{A}$ be a finite subset of unary predicates. We call elements of $\mathcal{A}_1$ *abstraction predicates.* An $\mathcal{A}_1$-bounded structure is three-valued structure $\langle U, \iota \rangle$ for which 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 $A \in \mathcal{A}_1$. The following definition of tight concretization corresponds to [48, Chapter 7], [45, Definition 3.6].

Definition 1 (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 $S^\sharp \sqsubseteq^h_T S$ 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 $S^\sharp \sqsubseteq^h_T S$, and in that case we call h a homomorphism. The tight concretization of a three-valued structure S, denoted $\gamma_T(S)$, 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: $\gamma^*_T(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} \gamma_T(S)$. The set of sets of two-valued structures definable via three-valued structure with tight concretization is $\mathsf{models}[T_2] = \{\gamma^*_T(\mathcal{S}) \mid \mathcal{S}$ a finite set of $\mathcal{A}_1$-bounded three-valued structures$\}$. We call the set of sets $\mathsf{models}[T_2]$ boolean shape analysis constraints *(the results of this paper justify to the name).*

If $A \in \mathcal{A}$ and $\alpha \in \{0,1\}$ then A^α is defined by $A^1 = A$ and $A^0 = \neg A$. 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 $\alpha_1, \ldots, \alpha_q \in \{0,1\}$.

Definition 2 (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 correspond to formulas in [48, Chapter 7]. TR_1-formulas satisfy syntactic invariants that make them isomorphic to three-valued structures with tight-concretization semantics.

Definition 3 (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 all of the following properties:*

P1. 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;

P2. there is at least one cube $P(x)$ such that the conjunct $\exists x.P(x)$ occurs in the conjunction;

P3. 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;

P4. for each cube $P(x)$ such that $\exists x.P(x)$ occurs, and each $A \in \mathcal{A} \setminus \mathcal{A}_1$, 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;

P5. 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 for 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.

A small difference between TR_1-formulas and formulas in [48, Definition 7.3.3, Page 31] is that [48, Definition 7.3.3, Page 31] does not contain conjuncts of the form $\neg\exists x.P(x)$ stating the emptiness of each empty cube, but instead contains one conjunct of the form $\forall x. \bigvee_P P(x)$ where P ranges over all non-empty cubes.

The following Proposition 4 shows that TR_1 formulas capture precisely the meaning of three-valued structures under tight concretization. The proof of Proposition 4 was first presented in [48, Appendix B] (and reviewed in [31, Page 9]). The proof shows that TR_1 formulas and bounded three-valued structures can be viewed as different notations for the same mathematical structure.

Proposition 4. $\mathsf{models}[TR_1] = \mathsf{models}[T_2]$

3 A New Characterization of Three-Valued Structures

Definition 5 introduces the new syntactic class of formulas characterizing three-valued structures under tight concretization semantics. Theorem 6 gives a constructive proof of the correctness of the characterization.

Definition 5 (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. $\exists x.\ B_1(x) \wedge Q(x)$

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

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

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

$$\begin{aligned}\exists x \exists y.\ (B_{11}(x) \vee B_{12}(x)) \wedge B_2(y) \wedge g(x,y) \rightarrow\ & \exists x \exists y.\ B_{11}(x) \wedge B_2(y) \wedge g(x,y)\ \vee \\ & \exists x \exists y.\ B_{12}(x) \wedge B_2(y) \wedge g(x,y)\end{aligned}$$

$$\begin{aligned}\exists x \exists y.\ B_1(x) \wedge (B_{21}(y) \vee B_{22}(y)) \wedge g(x,y) \rightarrow\ & \exists x \exists y.\ B_1(x) \wedge B_{21}(y) \wedge g(x,y)\ \vee \\ & \exists x \exists y.\ B_1(x) \wedge B_{22}(y) \wedge g(x,y)\end{aligned}$$

$$\begin{aligned}\exists x \exists y.\ B_1(x) \wedge B_2(y) \wedge (g_1(x,y) \vee g_2(x,y)) \rightarrow\ & \exists x \exists y.\ B_1(x) \wedge B_2(y) \wedge g_1(x,y)\ \wedge \\ & \exists x \exists y.\ B_1(x) \wedge B_2(y) \wedge g_2(x,y)\end{aligned}$$

Fig. 1. Transforming TR_4-literals into TR_1-literals.

Ensure each of the Properties of Definition 3 by applying the appropriate rules:

P1. $(\exists x. P(x)) \wedge (\neg \exists x. P(x)) \rightarrow \mathsf{false}$
$\mathsf{true} \rightarrow (\exists x. P(x)) \vee (\neg \exists x. P(x))$

P2. $\bigwedge_{P \in \mathsf{cubes}} \neg \exists x. P(x) \rightarrow \mathsf{false}$

P3. $(\neg \exists x. P(x)) \wedge (\exists x. P(x) \wedge Q(x)) \rightarrow \mathsf{false}$
$(\neg \exists x. P(x)) \wedge (\exists x \exists y. P(x) \wedge Q(x,y)) \rightarrow \mathsf{false}$
$(\neg \exists x. P(x)) \wedge (\exists x \exists y. P(y) \wedge Q(x,y)) \rightarrow \mathsf{false}$
$(\neg \exists x. P(x)) \wedge (\neg \exists x. P(x) \wedge Q(x)) \rightarrow \neg \exists x. P(x)$
$(\neg \exists x. P(x)) \wedge (\neg \exists x \exists y. P(x) \wedge Q(x,y)) \rightarrow \neg \exists x. P(x)$
$(\neg \exists x. P(x)) \wedge (\neg \exists x \exists y. P(y) \wedge Q(x,y)) \rightarrow \neg \exists x. P(x)$

P4. $(\exists x. P(x) \wedge Q(x)) \wedge (\neg \exists x. P(x) \wedge Q(x)) \rightarrow \mathsf{false}$
$(\neg \exists x. P(x) \wedge A(x)) \wedge (\neg \exists x. P(x) \wedge \neg A(x)) \;\rightarrow\; \neg \exists x. P(x)$
$$\begin{aligned}\mathsf{true} \rightarrow\ & (\neg \exists x. P(x) \wedge A(x))\ \vee \\ & (\neg \exists x. P(x) \wedge \neg A(x))\ \vee \\ & (\exists x. P(x) \wedge A(x)) \wedge (\exists x. P(x) \wedge \neg A(x))\end{aligned}$$
+rules for $P3$

P5. $(\exists x \exists y. P_1(x) \wedge P_2(y) \wedge Q(x,y)) \wedge (\neg \exists x \exists y. P_1(x) \wedge P_2(y) \wedge Q(x,y)) \rightarrow \mathsf{false}$
$$(\neg \exists x \exists y. P_1(x) \wedge P_2(y) \wedge f(x,y)) \;\wedge\; (\neg \exists x \exists y. P_1(x) \wedge P_2(y) \wedge \neg f(x,y)) \;\rightarrow\; (\neg \exists x. P_1(x)) \vee (\neg \exists y. P_2(y))$$
$$\begin{aligned}\mathsf{true} \rightarrow\ & (\neg \exists x \exists y. P_1(x) \wedge P_2(y) \wedge f(x,y))\ \vee \\ & (\neg \exists x \exists y. P_1(x) \wedge P_2(y) \wedge \neg f(x,y))\ \vee \\ & (\exists x \exists y. P_1(x) \wedge P_2(y) \wedge f(x,y)) \;\wedge\; (\exists x \exists y. P_1(x) \wedge P_2(y) \wedge \neg f(x,y))\end{aligned}$$
+rules for $P3$

Fig. 2. Transforming a conjunction of TR_1-literals into a canonical conjunction of TR_1-literals.

1. apply the rules in Figure 1 to transform the TR_4-formula into a boolean combination of TR_1-literals;
2. transform the formula into a disjunction of conjunctions of TR_1-literals;
3. apply the rules in Figure 2 to transform each conjunction of TR_1-literals into a canonical conjunction of TR_1-literals, while keeping the formula in disjunctive normal form.

Fig. 3. Normalization algorithm for transforming TR_4-formulas into TR_1-formulas.

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.

Theorem 6. *Algorithm sketched in Figures 3, 1, 2 converts a TR_4-formula into an equivalent TR_1-formula in a finite number of steps.*

Corollary 7. $\mathsf{models}[TR_4] = \mathsf{models}[TR_1] = \mathsf{models}[T_2]$.

By definition, TR_4-formulas are closed under all boolean operations.

Corollary 8. *1. 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.

2. 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 $\gamma_T^*(\mathcal{S}_1) \subseteq \gamma_T^*(\mathcal{S}_2)$ iff $\gamma_T^*(\mathcal{S}_3) = \emptyset$.

3. 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: $\gamma_T^*(\mathcal{S}_1) = \gamma_T^*(\mathcal{S}_2)$ iff $\gamma_T^*(\mathcal{S}_3) = \emptyset$.

Note. 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 $\mathcal{A}_1 = \mathcal{A}$. When $\mathcal{A} = \mathcal{A}_1$, then the class of TR_4-formulas can be defined simply as boolean combinations of formulas 1) $\exists x.\ B_1(x)$, and 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 $f \in \mathcal{F}$.

4 Decidability of Independent Predicates

We next examine the decidability of 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?" (Note that the sets $\mathcal{A}$ and $\mathcal{F}$ are part of the input to the decision procedure; for fixed finite sets $\mathcal{A}$ and $\mathcal{F}$ there are finitely many three-valued structures, so the decision problem would be trivial.) It turns out that satisfiability of TR_1-formulas over 2-STRUCT is decidable because the family of TR_1-formulas over 2-STRUCT has small model property. It is easy to construct a model $\langle U^\sharp, \iota^\sharp \rangle$ of a canonical conjunction of TR_1-literals by introducing at most two elements of the domain $U^\sharp$ for each non-empty cube.

Proposition 9. *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 $S^\sharp$.*

Corollary 10. $\gamma_T^*(\mathcal{S}) = \emptyset$ *iff* $\mathcal{S} = \emptyset$.

Corollary 11. *The following questions are decidable for sets $\mathcal{S}_1, \mathcal{S}_2$ of three-valued structures: 1) $\gamma_T^*(\mathcal{S}_1) = \emptyset$; 2) $\gamma_T^*(\mathcal{S}_1) \subseteq \gamma_T^*(\mathcal{S}_2)$; 3) $\gamma_T^*(\mathcal{S}_1) = \gamma_T^*(\mathcal{S}_2)$.*

5 Structures 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 ⊆ 2-STRUCT of *compatible* two-valued structures [46, Page 268]. The meaning of tight concretization with respect to 2-CSTRUCT is $c\gamma_T^*(\mathcal{S}) = \gamma_T^*(\mathcal{S}) \cap$ 2-CSTRUCT and we let $\mathsf{models}[cT_2]$ denote the set of all $c\gamma_T^*(\mathcal{S})$ for all finite sets of bounded three-valued structures. To characterize the meaning of three-valued structures over 2-CSTRUCT, for each class of formulas TR_i we introduce the corresponding class cTR_i by conjoining the formulas with a first-order formula F_ψ that characterizes the subset 2-CSTRUCT. It then follows $\mathsf{models}[cTR_i] = \{\mathcal{S}^\sharp \cap \text{2-CSTRUCT} \mid \mathcal{S}^\sharp \in \mathsf{models}[TR_i]\}$. Hence, $\mathsf{models}[cTR_4]$ is a subalgebra of the boolean algebra of subsets of 2-CSTRUCT, its sets are subsets of elements of $\mathsf{models}[TR_4]$, and the following generalization of Corollary 11 holds.

Corollary 12. *Assume that the satisfiability for three-valued structures interpreted over* 2-CSTRUCT *using tight concretization is decidable. Then the entailment and the equivalence of three-valued structures interpreted over* 2-CSTRUCT *using tight concretization are also decidable.*

6 Further Related Work

Researchers have proposed several program checking techniques based on dataflow analysis and symbolic execution [6, 8, 10, 13, 14, 19, 35]. The primary strength of shape analysis compared to the alternative approaches is the ability to perform sound and precise reasoning about dynamically allocated data structures.

Our work follows the line of shape analysis approaches which view the program as operating on concrete graph structures [9, 15, 16, 20, 22, 27, 32, 37, 46]. An alternative approach is to identify each heap object using the set of paths that lead to the object [7, 12, 18]. Other notations for reasoning about the heap include spatial logic [21] and alias types [47]. In the past we have seen a contrast between the approach to verification of dynamically allocated data structures based on Hoare logic [14, 21, 37], and the approach based on manipulation of graph summaries [9, 16, 22, 32]. The work [20] and especially [45] are important steps in bringing these two views together. Along with the recent work [40,48–50] our paper makes further contributions to unifying these two approaches.

The parametric framework for shape analysis is presented in [45, 46]. A systematic presentation of three-valued logic with equality is given in [38]. A description of a three-valued logic analyzer TVLA is in [33], an extension to interprocedural analysis is in [42, 43], and the use of shape analysis for program verification is demonstrated in [34]. A finite differencing approach for automatically computing transfer functions for analysis is presented in [39]. A shape analysis tool must ultimately take into account the definitions of instrumentation predicates, which requires some form of theorem proving or decision procedures. The original work [46, Page 272] uses rules based on Horn clauses for such reasoning, whereas [40, 48–50] (see Section 1) propose the use of theorem provers and decision procedures. 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. Of great importance for taking advantage of our result, as well as the results of [40, 48–50], are decidable logics that can express heap properties. Among the promising such logics are monadic second-order logic of trees [23], the logic L_r [4], and role logic [30].

It is possible to apply predicate abstraction techniques [2, 3, 17] 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 the easier application of representations such as binary decision diagrams [5, 11, 36]. 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.

7 Conclusions and Future Work

We have presented a new characterization of the constraints used as dataflow facts in parametric shape analysis. Our characterization represents these dataflow facts as boolean combinations of formulas. 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 these constraints is reducible to the satisfiability of the constraints.

In this paper we have focused on the tight-concretization semantics of three-valued structures. In the full version of the paper [31] we additionally show similar results for standard-concretization semantics of three-valued structures, with one important difference: the resulting constraints are closed under conjunction and disjunction, but not under negation. In fact, the least boolean algebra generated by those constraints is precisely the boolean algebra of boolean shape analysis 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 [46], this paper starts with three-valued structures and proceeds to characterize the structures using formulas. An alternative approach is to start with formulas that express the desired properties and then explore efficient ways of representing and manipulating these formulas. We believe that the entire framework [46] 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 Andreas Podelski, Thomas Reps, Mooly Sagiv, Greta Yorsh, and David Schmidt. We thank Greta Yorsh and Darko Marinov for useful comments on an earlier version of the paper.

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. Marius Bozga, Radu Iosif, and Yassine Laknech. Storeless semantics and alias logic. In *ACM PEPM'03*, pages 55–65. ACM Press, 2003.
8. 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.
9. David R. Chase, Mark Wegman, and F. Kenneth Zadeck. Analysis of pointers and structures. In *Proc. ACM PLDI*, 1990.
10. Lori Clarke and Debra Richardson. Symbolic evaluation methods for program analysis. In *Program Flow Analysis: Theory and Applications*, chapter 9. Prentice-Hall, Inc., 1981.
11. Dennis Dams and Kedar S. Namjoshi. Shape analysis through predicate abstraction and model checking. In *VMCAI 2003*, volume 2575 of *LNCS*, pages 310–323, 2003.

12. 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.
13. 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.
14. 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.
15. Pascal Fradet and Daniel Le Métayer. Shape types. In *Proc. 24th ACM POPL*, 1997.
16. Rakesh Ghiya and Laurie Hendren. Is it a tree, a DAG, or a cyclic graph? In *Proc. 23rd ACM POPL*, 1996.
17. Susanne Graf and Hassen Saidi. Construction of abstract state graphs with pvs. In *Proc. 9th CAV*, pages 72–83, 1997.
18. C. A. R. Hoare and He Jifeng. A trace model for pointers and objects. In *Proc. 13th ECOOP*, volume 1628 of *LNCS*, 1999.
19. Gerard J. Holzmann. Static source code checking for user-defined properties. In *Proc. IDPT 2002, Pasadena, CA*, June 2002.
20. Joseph Hummel, Laurie J. Hendren, and Alexandru Nicolau. A general data dependence test for dynamic, pointer-based data structures. In *Proc. ACM PLDI*, 1994.
21. Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. 28th ACM POPL*, 2001.
22. N. D. Jones and S. S. Muchnick. Flexible approach to interprocedural data flow analysis and programs with recursive data structures. In *Proc. 9th ACM POPL*, 1982.
23. Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. MONA implementation secrets. In *Proc. 5th International Conference on Implementation and Application of Automata*. LNCS, 2000.
24. Stephen Cole Kleene. *Introduction to Metamathematics*. D. Van Nostrand Company, Inc., Princeton, New Jersey, 1952. fifth reprint, 1967.
25. Viktor Kuncak, Patrick Lam, and Martin Rinard. A language for role specifications. In *Proceedings of the 14th Workshop on Languages and Compilers for Parallel Computing*, volume 2624 of *Lecture Notes in Computer Science, Springer*, 2001.
26. Viktor Kuncak, Patrick Lam, and Martin Rinard. Roles are really great! Technical Report 822, Laboratory for Computer Science, Massachusetts Institute of Technology, 2001.
27. Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Proc. 29th POPL*, 2002.
28. Viktor Kuncak and Martin Rinard. Typestate checking and regular graph constraints. Technical Report 863, MIT Laboratory for Computer Science, 2002.
29. 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.
30. Viktor Kuncak and Martin Rinard. On role logic. Technical Report 925, MIT CSAIL, 2003.
31. Viktor Kuncak and Martin Rinard. On the boolean algebra of shape analysis constraints. Technical report, MIT CSAIL, August 2003.

32. James R. Larus and Paul N. Hilfinger. Detecting conflicts between structure accesses. In *Proc. ACM PLDI*, Atlanta, GA, June 1988.
33. Tal Lev-Ami. TVLA: A framework for kleene based logic static analyses. Master's thesis, Tel-Aviv University, Israel, 2000.
34. 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.
35. 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.
36. 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.
37. Anders Møller and Michael I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.
38. Flemming Nielson, Hanne Riis Nielson, and Mooly Sagiv. Kleene's logic with equality. *Information Processing Letters*, 80(3):131–137, 2001.
39. Thomas Reps, Mooly Sagiv, and Alexey Loginov. Finite differencing of logical formulas for static analysis. In *Proc. 12th ESOP*, 2003.
40. Thomas Reps, Mooly Sagiv, and Greta Yorsh. Symbolic implementation of the best transformer. Technical Report TR-1468, University of Wisconsin, January 2003.
41. Thomas Reps, Mooly Sagiv, and Greta Yorsh. Symbolic implementation of the best transformer. In *VMCAI'04*, 2004.
42. Noam Rinetzky. Interprocedural shape analysis. Master's thesis, Technion - Israel Institute of Technology, 2000.
43. Noam Rinetzky and Mooly Sagiv. Interprocedual shape analysis for recursive programs. In *Proc. 10th International Conference on Compiler Construction*, 2001.
44. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM TOPLAS*, 20(1):1–50, 1998.
45. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. In *Proc. 26th ACM POPL*, 1999.
46. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
47. Frederick Smith, David Walker, and Greg Morrisett. Alias types. In *Proc. 9th ESOP*, Berlin, Germany, March 2000.
48. Greta Yorsh. Logical characterizations of heap abstractions. Master's thesis, Tel-Aviv University, March 2003.
49. Greta Yorsh, Thomas Reps, and Mooly Sagiv. Symbolically computing most-precise abstract operations for shape analysis. Technical report, Tel-Aviv University, September 2003. Forthcoming.
50. Greta Yorsh, Thomas Reps, Mooly Sagiv, and Reinhard Wilhelm. Logical characterizations of heap abstractions. Technical report, University of Wisconsin, Madison, January 2003.

Appendix: Example

Figure 4 illustrates the use of boolean shape analysis constraints and their closure properties. The left column introduces a set of constraints that provide a partial specification of an operation that removes some elements from a list. The right column shows that the conjunction of these constraints is a TR_4-formula. In this example $\mathcal{A}_1 = \mathcal{A} = \{A_l, A_r, A_{r0}\}$ and $\mathcal{F} = \{f\}$. The predicate A_l represents the object referenced by a local variable. The predicate A_{r0} denotes the set of nodes reachable from the local variable in the initial state, whereas the predicate A_r denotes the set of nodes reachable from the local variable in the final state of the operation. The binary relation f represents the value of the "next" pointer of the list in the final state. The meaning of the constraints is the following: C_1) the first element of the list has no incoming references; C_2) the list has at least two elements; C_3) the object referenced by local variable is reachable from the local variable in both pre- and post- state; C_4) following reachable nodes along the f field yields reachable nodes; C_5) all nodes are reachable; C_6) the data structure operation only removes elements from the set, it does not add any elements.

Consider the question whether the formula C_7 in Figure 4 is a consequence of the conjunction of constraints $\bigwedge_{i=1}^{6} C_i$. Transform the formula $\neg C_7 \wedge \bigwedge_{i=1}^{6} C_i$ into a TR_1-formula using our normalization algorithm in Figure 3. The result is the set of counterexamples in Figure 5 (represented as three-valued structures). These counterexamples show that the formula C_7 is *not* a consequence of the constraints in Figure 4, and show the set of scenarios in which the violation of formula C_7 can occur.

$C_1 : \forall y.A_l(y) \Rightarrow \neg\exists x.f(x,y)$	$\neg\exists x.\exists y.A_l(y) \wedge f(x,y)$
$C_2 : \exists x\exists y.A_l(x) \wedge \neg A_l(y) \wedge f(x,y)$	$\exists x\exists y.A_l(x) \wedge \neg A_l(y) \wedge f(x,y)$
$C_3 : \forall x.A_l(x) \Rightarrow A_r(x) \wedge A_{r0}(x)$	$\neg\exists x.\neg(A_l(x) \Rightarrow A_r(x) \wedge A_{r0}(x))$
$C_4 : \forall x\forall y.A_r(x) \wedge f(x,y) \Rightarrow A_r(y)$	$\neg\exists x\exists y.A_r(x) \wedge \neg A_r(y) \wedge f(x,y)$
$C_5 : \forall x.A_l(x) \vee A_r(x) \vee A_{r0}(x)$	$\neg\exists x.\neg(A_l(x) \vee A_r(x) \vee A_{r0}(x))$
$C_6 : \forall x.A_r(x) \Rightarrow A_{r0}(x)$	$\neg\exists x.\neg(A_r(x) \Rightarrow A_{r0}(x))$

Potential consequence:

C_7 : $\forall x\forall y.\neg A_l(x) \wedge \neg A_r(x) \wedge f(x,y) \Rightarrow \neg A_r(y)$

Fig. 4. Example verification of entailment of boolean shape analysis constraints.

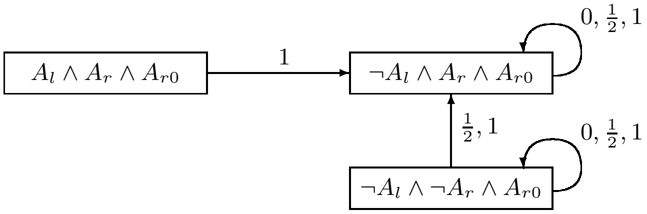
$A_l \wedge A_r \wedge A_{r0}$
1
$\neg A_l \wedge A_r \wedge A_{r0}$
$0, \frac{1}{2}, 1$
$\frac{1}{2}, 1$
$0, \frac{1}{2}, 1$
$\neg A_l \wedge \neg A_r \wedge A_{r0}$

Fig. 5. Counterexample structures for the entailment of constraints in Figure 4. There are $2 \cdot 3 \cdot 3$ counterexample three-valued structures, for different values of edges.