Abstract Interpretation as Automated Deduction

Vijay D'Silva[1] and Caterina Urban[2]

[1] Google Inc., San Francisco
[2] École Normale Supérieure, Paris

Abstract. Algorithmic deduction and abstract interpretation are two widely used and successful approaches to implementing program verifiers. A major impediment to combining these approaches is that their mathematical foundations and implementation approaches are fundamentally different. This paper presents a new, logical perspective on abstract interpreters that perform reachability analysis using non-relational domains. We encode reachability of a location in a control-flow graph as satisfiability in a monadic, second-order logic parameterized by a first-order theory. We show that three components of an abstract interpreter, the lattice, transformers and iteration algorithm, represent a first-order, substructural theory, parametric deduction and abduction in that theory, and second-order constraint propagation.

1 Introduction

Two major approaches to automated reasoning about programs are those based on SAT and SMT solvers and those based on abstract interpretation. In the solver-based approaches, a property of a program is encoded by formulae in a logic or theory and a solver is used to check if the property holds. In abstract interpretation, a property of a program is expressed in terms of fixed points and fixed point approximation techniques are used to calculate and reason about fixed points [6]. The complementary strengths of these approaches has led to a decade of theoretical and practical effort to combine them.

The strengths of SMT solvers include efficient Boolean reasoning, complete reasoning in certain theories, theory combination, proof generation and interpolation. Recent research has demonstrated that deduction algorithms have applications in program analysis besides solving formulae. DPLL(T) and CDCL have been lifted to implement property-guided, path-sensitive analyses [9, 15]. Stålmarck's method has been used to refine abstract transformers [26], interpolants have been used to refine widening operators [12] and unification has been used to obtain complete reasoning about restricted families of programs [28]. The Nelson-Oppen procedure, though less general than reduced product [7, 8], works as an algorithmic domain combinator [13].

Conversely, the strengths of abstract interpreters are the use of approximation to overcome the theoretical undecidability and practical scalability issues in program verification and the use of widening operators to derive invariants. The

large number of abstract domains enables application-specific reasoning and the flexibility to choose the trade-off between precision and efficiency. Ideas from abstract interpretation have been incorporated in SMT and constraint solvers by using abstract domains for theory propagation [29, 19], joins for space-efficient representation [3], and widening for generalization [18]. Algorithms based on fixed points have been used to implement alternatives to DPLL(T) [4, 27].

Nonetheless, there remain obstacles to combining these two approaches. Conceptually, SMT algorithms are expressed in terms of models and proofs while abstract interpretation is presented in terms of lattices, transformers and fixed points. These mathematical differences translate into practical differences in the interfaces implemented by solvers and abstract interpreters and type of results they produce, leading to further impediments to combining the two approaches.

This paper presents a logical account of a family of reachability analyses based on abstract interpretation. We encode reachability as satisfiability in a weak, monadic, second-order logic. A classic result of Büchi shows that a formula in the weak monadic second-order theory of one successor (WS1S) is satisfiable exactly if the models of that formula form a regular language [5, 31, 30]. If an automaton is viewed as a finite-state program, Büchi's theorem encodes reachability as satisfiability in WS1S. We extend a part of this result to a logic WS1S(T) interpreted over finite sequences of first-order structures.

Much of this paper is concerned with logical characterizations of the components of an abstract interpreter. The lattice in an analyzer represents a substructural, first-order theory, with the proof system for the theory generating the partial order of the lattice. Transformers for conditionals implement deduction and abduction modulo the theory. The invariant map constructed by abstract interpreters is a strict generalization of partial assignments in SAT and SMT solvers and fixed point iteration is second-order constraint propagation. Due to space restrictions, we defer proofs of statements to the full version of the paper.

2 Reachability as Second-Order Satisfiability

The contribution of this section is the logic WS1S(T), which is an extension of Büchi's WS1S with a theory. To simplify reasoning about programs in this logic, we restrict the class of models that are usually considered for WS1S.

Notation. We use $\hat{=}$ for definition. Let $\mathcal{P}(S)$ denote the set of all subsets of S, called the powerset of S, and $\mathcal{F}(S)$ denote the finite subsets of S. Given a function $f : A \to B$, $f[a \mapsto b]$ denotes the function that maps a to b and maps c distinct from a to $f(c)$.

2.1 Weak Monadic Second Order Theories of One Successor

Our syntax contains first-order variables *Vars*, functions *Fun* and predicates *Pred*. The symbols x, y, z range over *Vars*, f, g, h range over *Fun* and P, Q, R range over *Pred*. We also use a set *Pos* of first-order *position variables* whose elements are i, j, k and a set *SVar* of *monadic second-order variables* denoted

X, Y, Z. Second-order variables are uninterpreted, unary predicates. We also use a unary successor function *suc* and a binary, successor predicate *Suc*.

Our logic consists of three families of formulae called state, transition and trace formulae, which are interpreted over first-order structures, pairs of first-order structures and finite sequences of first-order structures respectively. The formulae are named after how they are interpreted over programs.

$$
\begin{array}{rl}
t ::= & x \mid f(t_0, \ldots, t_n) \qquad \text{Term} \\
\varphi ::= & P(t_0, \ldots, t_n) \mid \varphi \wedge \varphi \mid \neg\varphi \qquad \text{State Formula} \\
\psi ::= & suc(x) = t \mid \psi \wedge \psi \mid \neg\psi \qquad \text{Transition Formula} \\
\Phi ::= & X(i) \mid Suc(i,j) \mid \varphi(i) \mid \psi(i) \\
& \mid \Phi \wedge \Phi \mid \neg\Phi \mid \exists i : Pos.\Phi \qquad \text{Trace formula}
\end{array}
$$

State formulae are interpreted with respect to a *theory* $\mathcal{T}$ given by a first-order interpretation (Val, I), which defines functions $I(f)$, relations $I(P)$, and equality $=_{\mathcal{T}}$ over values in *Val*. A *state* maps variables to values and $State \mathrel{\hat{=}} Vars \to Val$ is the set of states. The value $[\![t]\!]_s$ of a term t in a state s is defined as usual.

$$[\![x]\!]_s \mathrel{\hat{=}} s(x) \qquad\qquad [\![f(t_1, \ldots, t_k)]\!]_s \mathrel{\hat{=}} I(f)([\![t_0]\!]_s, \ldots, [\![t_n]\!]_s)$$

As is standard, $s \models_{\mathcal{T}} \varphi$ denotes that s is a model of φ in the theory $\mathcal{T}$.

$$
\begin{array}{l}
s \models_{\mathcal{T}} P(t_0, \ldots, t_n) \text{ if } ([\![t_0]\!]_s, \ldots, [\![t_n]\!]_s) \in I(P) \\
s \models_{\mathcal{T}} \varphi \wedge \psi \text{ if } s \models_{\mathcal{T}} \varphi \text{ and } s \models_{\mathcal{T}} \psi \qquad\qquad s \models_{\mathcal{T}} \neg\varphi \text{ if } s \not\models_{\mathcal{T}} \varphi
\end{array}
$$

The semantics of Boolean operators is defined analogously for transition and trace formulae, so we omit them in what follows. A *transition* is a pair of states (r, s) and a transition formula is interpreted at a transition.

$$
\begin{array}{l}
(r, s) \models P(t_0, \ldots, t_n) \text{ if } ([\![t_0]\!]_r, \ldots, [\![t_n]\!]_r) \in I(P) \\
(r, s) \models suc(x) = t \text{ if } [\![x]\!]_s =_{\mathcal{T}} [\![t]\!]_r
\end{array}
$$

A *trace of length* k is a sequence $\tau : [0, k-1] \to State$. We call $\tau(m)$ the *state at position* m, with the implicit qualifier $m < k$. A *k-assignment* $\sigma : (Pos \to \mathbb{N}) \cup (SVar \to \mathcal{F}(\mathbb{N}))$ maps position variables to $[0, k-1]$ and second-order variables to *finite subsets* of $[0, k-1]$. A k-assignment satisfies that $\{\sigma(X) \mid X \in SVar\}$ partitions the interval $[0, k-1]$. We explain the partition condition shortly. A WS1S(T) *structure* (τ, σ) consists of a trace τ of length k and a k-assignment σ. A trace formula is interpreted with respect to a WS1S(T) structure.

$$
\begin{array}{l}
(\tau, \sigma) \models X(i) \text{ if } \sigma(i) \text{ is in } \sigma(X) \\
(\tau, \sigma) \models \varphi(i) \text{ if } \tau(\sigma(i)) \models_{\mathcal{T}} \varphi \\
(\tau, \sigma) \models \psi(i) \text{ if } \sigma(i) < k-1 \text{ and } (\tau(\sigma(i)), \tau(\sigma(i)+1)) \models \psi \\
(\tau, \sigma) \models Suc(i,j) \text{ if } \sigma(i) + 1 = \sigma(j) \\
(\tau, \sigma) \models \exists i : Pos.\Phi \text{ if } (\tau, \sigma[i \mapsto n]) \models \Phi \text{ for some } n \text{ in } \mathbb{N}
\end{array}
$$

Note that $\varphi(i)$ is interpreted at the state at position $\sigma(i)$ and $\psi(i)$ at the transition from $\sigma(i)$. The semantics of $\psi(i)$ is only defined if $\sigma(i)$ is not the last position on τ. A trace formula Φ is *satisfiable* if there exists a trace τ and assignment σ such that $(\tau, \sigma) \models \Phi$. We assume standard shorthands for $\vee$ and $\Rightarrow$ and write $\Phi \models \Psi$ for $\models \Phi \Rightarrow \Psi$.

Example 1. The WS1S formula $\mathit{First}(i) \mathrel{\hat{=}} \forall j.\neg \mathit{Suc}(j, i)$ is true at the first position on a trace and $\mathit{Last}(i) \mathrel{\hat{=}} \forall j.\neg \mathit{Suc}(i, j)$ is true at the last position. See [31, 30] for more examples. WS1S(T) has no second-order quantification so the encoding of transitive closure in WS1S does not carry over. Transitive closure may be encoded if the underlying theory is powerful enough. $\lhd$

2.2 Encoding Reachability in WS1S(T)

Büchi showed that the models of a WS1S formula form a regular language and vice-versa. The modern proof of this statement [31, 30] encodes the structure of a finite automaton using second-order variables. We now extend this construction to encode a control-flow graph (CFG) by a WS1S(T) formula.

A *command* is an assignment $x := t$ of a term t to a first-order variable x, or is a condition $[\varphi]$, where φ is a state formula. A CFG $G = (\mathit{Loc}, E, \mathtt{in}, \mathit{Ex}, \mathit{stmt})$ consists of a finite set of locations Loc including an initial location $\mathtt{in}$, a set of exit locations Ex, edges $E \subseteq \mathit{Loc} \times \mathit{Loc}$, and a labelling $\mathit{stmt} : E \to \mathit{Cmd}$ of edges with commands. To assist the presentation, we require that every location is reachable from $\mathtt{in}$, and that exit locations have no successors. This requirement is not fundamental to our results.

We define an execution semantics for CFGs. We assume that terms in commands are interpreted over the same first-order structure as state formulae. The formula $\mathit{Same}_V \mathrel{\hat{=}} \bigwedge_{x \in V} \mathit{succ}(x) = x$ expresses that variables in the set V are not modified in a transition and Trans_c is the *transition formula* for a command.

$$\mathit{Trans}_c \mathrel{\hat{=}} \begin{cases} b \implies \mathit{Same}_{\mathit{Vars}} & \text{if } c = [b] \\ \mathit{suc}(x) = t \wedge \mathit{Same}_{\mathit{Vars} \setminus \{x\}} & \text{if } c = x := t \end{cases}$$

A *transition relation* for a command c is the set of models Rel_c of Trans_c. We write Trans_e and Rel_e for the transition formula and relation of the command $\mathit{stmt}(e)$. An *execution of length* k is a sequence $\rho = (m_0, s_0), \ldots, (m_{k-1}, s_{k-1})$ of location and state pairs in which each $e = (m_i, m_{i+1})$ is an edge in E and the pair of states (s_i, s_{i+1}) is in the transition relation Rel_e. A location m is *reachable* if there is an execution ρ of some length k such that $\rho(k-1) = (m, s)$ for some state s.

The safety properties checked by abstract interpreters are usually encoded as reachability of locations in a CFG. The formula $\mathit{Reach}_{G,L}$ below encodes reachability of a set of locations L in a CFG G as satisfiability in WS1S(T). The first line below is an *initial constraint*, the second is a set of *transition constraints*

indexed by locations, and the third line encodes *final constraints*.

$$\begin{aligned} \mathit{Reach}_{G,L} \mathrel{\hat{=}} & \forall i.\mathit{First}(i) \implies X_{\mathtt{in}}(i) \\ & \wedge \bigwedge_{v \in \mathit{Loc}} \forall i.\forall j.X_v(j) \wedge \mathit{Suc}(i,j) \implies \bigvee_{(u,v)\in E} \mathit{Trans}_{(u,v)}(i) \wedge X_u(i) \\ & \wedge \forall j.\mathit{Last}(j) \implies \bigvee_{u \in L} X_u(j) \end{aligned}$$

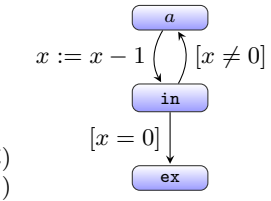
We explain this formula in terms of a structure (τ, σ). The trace τ contains valuations of variables but has no information about locations. A second-order variable X_v represents the location v and $\sigma(X_v)$ represents the points in τ when control is at v. The initial constraint ensures that execution begins in $\mathtt{in}$. The final constraint ensures that execution ends in one of the locations in L. In a transition constraint, $X_v(j) \wedge \mathit{Suc}(i,j)$ expresses that the state $\tau(j)$ is visited at location v and its consequent expresses that the state $\tau(i)$ must have been visited at a location u that precedes v in the CFG and that $(\tau(i), \tau(j))$ must be in the transition relation (u, v).

Theorem 1. *Some location in a set L in a* CFG *G is reachable if and only if the formula $\mathit{Reach}_{G,L}$ is satisfiable.*

Proof. [$\Leftarrow$] If a location $w \in L$ is reachable, there is an execution $\rho \mathrel{\hat{=}} (u_0, s_0), \ldots, (u_{k-1}, s_{k-1})$ with $u_0 = \mathtt{in}$ and $u_{k-1} = w$. Define the structure (τ, σ) with $\tau \mathrel{\hat{=}} s_0, \ldots, s_{k-1}$ and $\sigma \mathrel{\hat{=}} \{X_u \mapsto \{i \mid \rho(i) = (u, s), s \in \mathit{State}\} \mid u \in \mathit{Loc}\}$. We show that (τ, σ) is a model of $\mathit{Reach}_{G,L}$. Since $u_0 = \mathtt{in}$ and $u_{k-1} = w$, the initial and final constraints are satisfied. In the transition constraint, if $X_v(j)$ holds, there is some $(u_i, s_i), (u_{i+1}, s_{i+1})$ in ρ with $u_{i+1} = v$. Thus, the transition (s_i, s_{i+1}) satisfies the transition formula $\mathit{Trans}_{(u_i,v)}$.
[$\Rightarrow$]Assume (τ, σ) is a model of $\mathit{Reach}_{G,L}$. Define a sequence ρ with $\rho(i) \mathrel{\hat{=}} (u, \tau(i))$ where $i \in \sigma(X_u)$. As σ induces a partition, there is a unique u with i in $\sigma(X_u)$. We show that ρ is an execution reaching L. The initial constraint guarantees that $\rho(0)$ is at $\mathtt{in}$ and the final constraints guarantee that ρ ends in L. The transition constraints ensure that every step in the execution traverses an edge in G and respects the transition relation of the edge. □

We believe this is a simple yet novel encoding of reachability, a property widely checked by abstract interpreters, in a minor extension of a well-known logic. The translation from second-order logics is at the heart of the automata-based verification, so we believe this work connects abstract interpretation to the automata-theoretic approach to program verification in a fundamental, yet novel way. In other second-order characterizations of correctness [2, 11], it is invariants and not executions that are encoded by satisfying assignments. Moreover, those encodings do not connect to the automata-theoretic approach.

Example 2. A CFG G and the formula $\mathit{Reach}_{G,Ex}$ for a program with a variable x of type $\mathbb{Z}$ are shown in Fig. 1. Executions that start with a strictly negative value of x neither terminate nor reach $\mathtt{ex}$. For brevity, we write a state as the value



$$
\begin{array}{r}
(\forall i.\mathit{First}(i) \Rightarrow X_{\mathtt{in}}(i)) \wedge (\forall i.\mathit{Last}(i) \Rightarrow X_{\mathtt{ex}}(i)) \\
\wedge\ \forall i.\forall j.X_{\mathtt{in}}(j) \wedge \mathit{Suc}(i,j) \Rightarrow (\mathit{suc}(x) = x - 1)(i) \wedge X_a(i) \\
\wedge\ \forall i.\forall j.X_a(j) \wedge \mathit{Suc}(i,j) \Rightarrow ((x \neq 0 \Rightarrow \mathit{suc}(x) = x)(i) \wedge X_{\mathtt{in}}(i) \\
\wedge\ \forall i.\forall j.X_{\mathtt{ex}}(j) \wedge \mathit{Suc}(i,j) \Rightarrow (x = 0 \Rightarrow \mathit{suc}(x) = x)(i) \wedge X_{\mathtt{in}}(i))
\end{array}
$$

Fig. 1. A CFG for a program with non-terminating executions and a WS1S(T) formula over the theory of integer arithmetic encoding the reachability of `ex`.

of x. The execution $(\mathtt{in}, 1), (a, 1), (\mathtt{in}, 0), (\mathtt{ex}, 0)$ reaches `ex`. It is encoded by the model (τ, σ), with $\sigma \mathrel{\hat{=}} \{X_{\mathtt{in}} \mapsto \{0, 2\}, X_a \mapsto \{1\}, X_{\mathtt{ex}} \mathrel{\hat{=}} \{3\}\}$ and $\tau = 1, 1, 0, 0$. Note that σ partitions SVar because each position on the trace corresponds to a unique location. No structure (τ, σ) in which x is strictly negative in $\tau(0)$ satisfies $\mathit{Reach}_{G,Ex}$. ◁

Note that a program invariant would include all reachable states, but a model of $\mathit{Reach}_{G,L}$ only involves states that occur on a single execution. We emphasise that we are not considering arbitrary formulae in WS1S(T). The formula $\mathit{Reach}_{G,L}$ is a conjunction of constraints in which the initial, final and transition constraints have a fixed structure. The second-order variables and first-order program variables are free, but the first-order position variables are bound.

3 Lattices and Substructural First-Order Theories

The contribution of this section is to relate first-order substructural theories with the lattices in abstract domains. We show that certain lattices used in practice are Lindenbaum-Tarski algebras of theories that we introduce here.

3.1 First-Order Substructural Theories

For this section, assume a set of variables *Vars* and a first-order theory of integer arithmetic with the standard functions and relations and let $\models_{\mathbb{Z}}$ define the semantics of quantifier-free first-order formulae. A *logical language* $(\mathcal{L}, \vdash_{\mathcal{L}})$ consists of a set of formulae and a proof system. The grammar below defines a set of formulae in terms of atomic formulae, logical constants and connectives.

We introduce formulae and calculi for a *sign logic*, a *constant logic* and an *interval logic*, with the names deriving from the abstract domains being modelled. The formulae in our logics are closed under conjunction but not under disjunction or negation. There are only three atomic formulae in $\mathcal{S}$. The infinitely many atomic formulae in $\mathcal{C}$ are equalities between a variable and an integer, and atomic formulae in $\mathcal{I}$-formulae involve upper bounds and lower bounds on variables. The three logics contain the logical constant `tt`, denoting true, but instead

The core calculus $\vdash_{\text{CORE}}$

$$\frac{}{\varphi \vdash \varphi}\ \text{I} \qquad \frac{}{\mathsf{ff}_x \vdash \varphi(x)}\ \mathsf{ff}_\text{L} \qquad \frac{\Gamma \vdash \varphi \qquad \varphi, \Sigma \vdash \psi}{\Gamma, \Sigma \vdash \psi}\ \text{CUT} \qquad \frac{}{\Gamma \vdash \mathsf{tt}}\ \mathsf{tt}_\text{R}$$

$$\frac{\Gamma \vdash \psi}{\Gamma, \varphi \vdash \psi}\ \text{WL} \qquad \frac{\Gamma, \varphi, \varphi \vdash \psi}{\Gamma, \varphi \vdash \psi}\ \text{CL} \qquad \frac{\Gamma, \varphi, \psi \vdash \theta}{\Gamma, \psi, \varphi \vdash \theta}\ \text{PL}$$

$$\frac{\Gamma, \varphi \vdash \theta}{\Gamma, \varphi \wedge \psi \vdash \theta}\ \wedge\text{L}_1 \qquad \frac{\Gamma, \psi \vdash \theta}{\Gamma, \varphi \wedge \psi \vdash \theta}\ \wedge\text{L}_2 \qquad \frac{\Gamma \vdash \varphi \qquad \Sigma \vdash \psi}{\Gamma, \Sigma \vdash \varphi \wedge \psi}\ \wedge\text{R}$$

Table 1. Proof rules for the core calculus and its extensions. The core calculus $\vdash_{\text{CORE}}$ contains rules for introduction (I), cut (CUT), weakening (WL), contraction (CL) and permutation (PL) on the left, conjunction ($\wedge$L$_1$, $\wedge$L$_2$, $\wedge$R), false (ffL), in which $\varphi(x)$ has only one free variable x, and true (ttR).

of a constant for false, we have a family ff_x parameterized by variables.

$$\varphi ::= x < 0 \mid x = 0 \mid x > 0 \mid \mathsf{ff}_x \mid \mathsf{tt} \mid \varphi \wedge \varphi \qquad \mathcal{S}$$
$$\varphi ::= x = k \mid \mathsf{ff}_x \mid \mathsf{tt} \mid \varphi \wedge \varphi \qquad \mathcal{C}$$
$$\varphi ::= x \leq k \mid x \geq k \mid \mathsf{ff}_x \mid \mathsf{tt} \mid \varphi \wedge \varphi \qquad \mathcal{I}$$

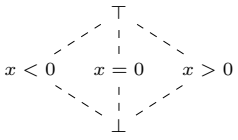
A calculus $\vdash_{\text{CORE}}$ for the logical core of these logics is shown in Table 1. We use sequents of the form $\Gamma, \Sigma \vdash_\mathcal{L} \varphi$, where the *antecedents* Γ and Σ are sequences of formulae, and the *consequent* φ is a single, first-order formula. We write $\bigwedge \Gamma$ for the conjunction of the sequence elements in Γ. A calculus $\vdash_\mathcal{L}$ is *sound* if every derivable sequent $\Gamma \vdash_\mathcal{L} \psi$ satisfies that $\models_\mathbb{Z} \bigwedge \Gamma \Rightarrow \psi$. The semantics of ff_x in $\models_\mathbb{Z}$ is ff. Two formulae are *inter-derivable* if the sequents $\varphi \vdash_\mathcal{L} \psi$ and $\psi \vdash_\mathcal{L} \varphi$ are both derivable.

Sequent calculi usually contain structural, logical and cut rules, and in the case of theories also theory rules. Our logics are *substructural* because the sequents have a restricted structure, lack right structural rules, and lack rules for disjunction, negation and implication. Our non-standard treatment of false is influenced by the way abstract domains reason about contradictions.

We review the theory rules for our logics. The reader should be warned that these logics have a restricted syntax and weak proof systems so the set of derivations is limited. We claim no responsibility for any disappointment arising from how uninteresting the derivable theorems are. The calculus $\vdash_\mathcal{S}$, in Fig. 2, extends $\vdash_{\text{CORE}}$ with rules for deriving ff_x from conjunctions of atomic formulae. The calculus for $\mathcal{C}$ is similar to that for $\mathcal{S}$ with the theory rule below instead. The calculus $\mathcal{I}$ in Fig. 4 contains rules for modifying upper and lower bounds.

$$[m \neq_\mathbb{Z} n]\ \frac{}{\Gamma, x = m \wedge x = n \vdash \mathsf{ff}_x}\ \mathsf{ff}_{\text{R}_4}$$

Example 3. Fig. 3 contains a derivation of $x < 0 \vdash_\mathcal{S} x < 0 \wedge \mathsf{tt}$. The converse $x < 0 \wedge \mathsf{tt} \vdash_\mathcal{S} x < 0$ is derivable with I and $\wedge$L$_1$, showing that $x < 0$ and $x < 0 \wedge \mathsf{tt}$ are inter-derivable.



$$\vdash_{\mathcal{S}}$$

$$\vdash_{\text{CORE}}$$

$$\frac{}{\Gamma, x < 0 \wedge x = 0 \vdash \mathsf{ff}_x}\ \mathsf{ff}_{\mathrm{R}_1}$$

$$\frac{}{\Gamma, x = 0 \wedge x > 0 \vdash \mathsf{ff}_x}\ \mathsf{ff}_{\mathrm{R}_2}$$

$$\frac{}{\Gamma, x < 0 \wedge x > 0 \vdash \mathsf{ff}_x}\ \mathsf{ff}_{\mathrm{R}_3}$$

Fig. 2. A lattice of signs and a calculus that generates it.

$$\dfrac{\dfrac{\dfrac{}{x < 0 \vdash_{\mathcal{S}_1} x < 0}\ \mathrm{I} \qquad \dfrac{}{x < 0 \vdash_{\mathcal{S}_1} \mathsf{tt}}\ \mathsf{tt}_{\mathrm{R}}}{x < 0, x < 0 \vdash_{\mathcal{S}_1} x < 0 \wedge \mathsf{tt}}\ \wedge_{\mathrm{R}}}{x < 0 \vdash_{\mathcal{S}_1} x < 0 \wedge \mathsf{tt}}\ \mathrm{CL}$$

Fig. 3. A derivation in the sign calculus $\vdash_{\mathcal{S}}$.

An abstract interpreter reasoning about variable values can derive a sequent $y \leq 0, x \leq 5 \wedge x \geq 7 \vdash_{\mathcal{I}} \mathsf{ff}_x \wedge y \leq 3$ showing that the inconsistency arises from x or $x \leq 2, y \leq 0 \wedge y \geq 3 \vdash_{\mathcal{I}} \mathsf{ff}_y \wedge x \leq 3$ showing an inconsistency from y. $\lhd$

Theorem 2. *The calculi $\vdash_{\mathcal{S}}$, $\vdash_{\mathcal{C}}$, and $\vdash_{\mathcal{I}}$ are sound.*

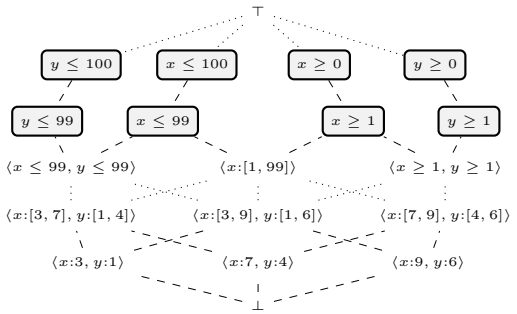
The proof is by induction on the structure of a derivation. This soundness theorem is used to show an isomorphism between the lattices generated by these calculi and the lattices they model.

3.2 Lattices from Substructural Theories

We recall elementary lattice theory. A lattice $(A, \sqsubseteq, \sqcap, \sqcup)$ is a partially ordered set (poset) with a binary greatest lower bound $\sqcap$, called the *meet*, and a binary least upper bound $\sqcup$, called the *join*. A poset with only a meet is called a meet-semi-lattice. A lattice is *bounded* if it has a greatest element $\top$, called *top*, and a least element $\bot$ called *bottom*. The notion of isomorphism for lattices is standard.

Pointwise lifting is an operation that lifts the order and operations of a lattice to functions on the lattice. Consider the functions $f, g : S \to A$, where S is a set and A a lattice as above. The pointwise order $f \sqsubseteq g$ holds if $f(x) \sqsubseteq g(x)$ for all x, while the pointwise meet $f \sqcap g$ maps x in S to $f(x) \sqcap g(x)$. The pointwise lift of other relations and operations on A is similarly defined.

Tarski related logic and lattices by extending a construction of Lindenbaum to generate Boolean algebras from propositional calculi and first-order sentences. We use a generalization of this construction to formulae with free variables [20]. We write $[\varphi]_{\mathcal{L}}$ for the equivalence class of φ with respect to an equivalence relation $\equiv_{\mathcal{L}}$. A logic $(\mathcal{L}, \vdash_{\mathcal{L}})$ that is closed under conjunction generates the



$$\vdash_{\mathcal{I}}$$

$$\vdash_{\text{CORE}}$$

$$[m \leq n]\ \frac{\Gamma, x \leq n \vdash \varphi}{\Gamma, x \leq m \vdash \varphi}\ \text{UB-L} \qquad [m \leq n]\ \frac{\Gamma \vdash x \leq m}{\Gamma \vdash x \leq n}\ \text{UB-R}$$

$$[m \leq n]\ \frac{\Gamma, x \geq m \vdash \varphi}{\Gamma, x \geq n \vdash \varphi}\ \text{LB-L} \qquad [m \leq n]\ \frac{\Gamma \vdash x \geq n}{\Gamma \vdash x \geq m}\ \text{LB-R}$$

$$[m < n]\ \frac{}{\Gamma, x \leq m \wedge x \geq n \vdash \mathsf{ff}_x}\ \mathsf{ff}\text{R}_5$$

Fig. 4. The domain of intervals over two variables and a calculus for interval logic.

Lindenbaum-Tarski algebra $(\mathcal{L}/\equiv_{\mathcal{L}}, \preccurlyeq, \curlywedge)$ below.

$$\varphi \equiv_{\mathcal{L}} \psi \text{ if } \varphi \vdash_{\mathcal{L}} \psi \text{ and } \psi \vdash_{\mathcal{L}} \varphi.$$
$$[\varphi]_{\mathcal{L}} \preccurlyeq [\psi]_{\mathcal{L}} \text{ if } \theta_1 \vdash_{\mathcal{L}} \theta_2 \text{ for some } \theta_1 \in [\varphi]_{\mathcal{L}}, \text{ and } \theta_2 \in [\psi]_{\mathcal{L}}.$$
$$[\varphi]_{\mathcal{L}} \curlywedge [\psi]_{\mathcal{L}} \mathrel{\hat{=}} [\theta_1 \wedge \theta_2]_{\mathcal{L}} \text{ where } \theta_1 \in [\varphi]_{\mathcal{L}}, \text{ and } \theta_2 \in [\psi]_{\mathcal{L}}.$$

The relation $\equiv_{\mathcal{L}}$, defined by inter-derivability, is an equivalence whose classes form the carrier set of the algebra. Logical connectives generate operators. Though Lindenbaum-Tarski algebras of standard logics have been studied in depth, the algebras for the substructural theories we consider have not. To prove the lemma below, we show that derivability induces a partial order on the equivalence classes of $\equiv_{\mathcal{L}}$ and that conjunction induces a greatest lower bound.

Lemma 1. *Let* $(\mathcal{L}, \vdash)$ *be a quantifer-free first-order language closed under conjunction and* $\vdash$ *be a sound calculus that extends* $\vdash_{\text{CORE}}$*. The Lindenbaum-Tarski algebra of* $\mathcal{L}$ *is a meet-semi-lattice.*

We now recall certain lattices used in abstract interpretation and show that they are isomorphic to the Lindenbaum-Tarski algebras of the logics we introduced. The lattice of signs $(\mathit{Sign}, \sqsubseteq)$ is shown in Fig. 2. The lattice of integer constants $(\mathit{Const}, \sqsubseteq)$ consists of the elements $\mathbb{Z} \cup \{\bot, \top\}$, with $\bot$ and $\top$ as bottom and top, and all other elements being incomparable. The lattice of integer intervals $(\mathit{Itv}, \sqsubseteq)$, consists of the set $\{[a, b] \mid a \leq b, a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{\infty\}\}$ and a special element $\bot$ denoting the empty interval. The partial order is standard and $[-\infty, \infty]$ is the top element.

An *abstract environment* is a function $\varepsilon : \mathit{Vars} \to D$, from program variables to a lattice D that represents approximations of variable values. A lattice of abstract environments is derived from a lattice D by pointwise lifting.

Theorem 3. *The Lindenbaum-Tarski algebra of each of the logics* $\mathcal{S}$*,* $\mathcal{C}$*,* $\mathcal{I}$ *over a set of variables Vars, is isomorphic to the pointwise lift of each of the lattices Sign, Const, and Itv to abstract environments over Vars.*

To provide intuition for the proof, we detail here the case for the logic $\mathcal{S}$ and the lattice *Sign* over one variable. To prove that the Lindenbaum-Tarski algebra of $\mathcal{S}$ over a variable x is isomorphic to *Sign* we have to show that there are five equivalence classes, and that $\preccurlyeq$ and $\curlywedge$ are as in Fig. 2. The five candidate equivalence classes are $\{[\mathsf{ff}_x]_{\mathcal{S}}, [\mathsf{tt}]_{\mathcal{S}}, [x < 0]_{\mathcal{S}}, [x = 0]_{\mathcal{S}}, [x > 0]_{\mathcal{S}}\}$. The proof that there are at most five equivalence classes is by induction on the structure of formulae. The proof argument is that every conjunct in $\mathcal{S}$ is inter-derivable from a formula in one of these classes. The proof that there are at least five equivalence classes relies on the soundness of $\vdash_{\mathcal{S}}$. If there are fewer than five equivalence classes, there are consequences derivable in $\vdash_{\mathcal{S}}$ that do not hold semantically. Observe that this proof argument holds only because every lattice element represents a different set of structures. In abstract interpretation parlance, this argument only applies to abstractions in which the concretization function is injective.

Next, we define a function $h : \mathcal{S}/\equiv_{\mathcal{S}} \to \mathit{Sign}$ that maps equivalence classes to corresponding lattice elements. To show that h is an isomorphism we argue by induction on formula structure for comparable equivalence classes and by appeal to soundness for incomparable equivalence classes. This argument generalizes to a finite number of variables because all the logics we have considered only involve one-place predicates. The shaded elements in Fig. 4 are the images of the formulae shown under the isomorphism.

4 Abstract Transformers, Deduction and Abduction

The constant and interval domains are used in practice even though, as shown in the previous section, they have weak proof systems. In this section, we adapt Tarski's notion of consequence operators to logically model abstract transformers for conditionals. Consequence operators provided an approach to algebraically modelling deduction. These transformers can be viewed as enriching a weak proof system $\vdash_{\mathcal{L}}$ with the ability to reason about formulae that are not definable in $\mathcal{L}$.

We consider again a quantifier-free first-order theory $\mathcal{T}$ with semantics $\models_{\mathcal{T}}$ and a logical language $(\mathcal{L}, \vdash_{\mathcal{L}})$, where $\mathcal{L} \subseteq \mathcal{T}$. Deduction in $\mathcal{L}$ *with respect to a formula* $\varphi \in \mathcal{T}$ is formalized by a *deduction function* $ded_{\varphi} : \mathcal{F}(\mathcal{L}) \to \mathcal{F}(\mathcal{L})$ between finite sets of formulae in $\mathcal{L}$. A deduction function is sound if for finite $\Theta \subseteq \mathcal{L}$, and $\theta \in ded_{\varphi}(\Theta)$, $\varphi \wedge \bigwedge \Theta \models_{\mathcal{T}} \theta$. That is, the deduced formulae are consequences of the arguments and, crucially, the parameter φ. The formula φ acts as an external hint to boost the capabilities of a weak deductive system. The parameter φ may not exist in $\mathcal{L}$, so there may not be a rule of the form $\varphi, \Gamma \vdash_{\mathcal{L}} \theta$ corresponding to an application of the deduction function.

Similarly, we model abduction by a function that generates antecedents given consequents. An abduction function $abd_{\varphi} : \mathcal{F}(\mathcal{L}) \to \mathcal{F}(\mathcal{L})$ derives antecedents in $\mathcal{L}$ with respect to a parameter φ. An abduction function is sound if for all Θ, and $\theta \in abd_{\varphi}(\Theta)$, $\varphi \wedge \theta \models_T \bigwedge \Theta$.

Example 4. This examples illustrates how deduction with respect to a formula enables reasoning that is not possible in the lattice itself. Let $\varphi \mathrel{\hat{=}} 3y - 1 >$

$0 \wedge x = -y$ be a formula in a theory. We define one possible sound deduction function ded_φ for consequences in $\mathcal{S}$.

$$ded_\varphi(\{\mathsf{tt}\}) = ded_\varphi(\{y > 0\}) = ded_\varphi(\{x < 0\}) = \{y > 0, x < 0\}$$
$$ded_\varphi(\{\mathsf{ff}_x\}) = ded_\varphi(\{x = 0\}) = ded_\varphi(\{x > 0\}) = ded_\varphi(\{y = 0\}) = \{\mathsf{ff}_x, \mathsf{ff}_y\}$$

The results of applying ded_φ shown above are the only two possibilities, even for sets of formulae not shown above. ◁

The difference between ded_φ and classical consequence operators is that we make fewer assumptions on properties of ded_φ in the same way our lattices make fewer structural assumptions than classical logics. Recall that a set of formulae C is *consequence-closed* with respect to $\vdash_{\mathcal{L}}$ if for all φ in C, if $\varphi \vdash_{\mathcal{L}} \theta$, then θ is in C. The *consequence closure* of C is the smallest consequence-closed set containing C. If $\Gamma \vdash_{\mathcal{L}} \theta$, the consequence closure of Γ contains the consequence closure of θ. A deduction function inverts this relationship, because it strengthens its argument using φ. That is, the consequence closure of $ded_\varphi(\Theta)$ is a superset of the consequence closure of Θ because it contains formulae derived using φ.

Deduction functions, when factored through the Lindenbaum-Tarski equivalence relation, give rise to sound transformers for conditionals. To make this precise, we require the notion of a concretization function from abstract interpretation. Let $(A, \sqsubseteq, \sqcap)$ be a bounded lattice and $(\mathcal{P}(State), \subseteq, \cap)$ be the powerset of states with the subset order. We say that A is an *abstraction of* $\mathcal{P}(State)$ if there is a monotone function $\gamma : A \to \mathcal{P}(State)$ satisfying that $\gamma(\top) = State$ and $\gamma(\bot) = \emptyset$. Requiring that $\bot$ has an empty concretization is non-standard but is required for a logical treatment of false.

Recall that $Rel_{[\varphi]}$ is the transition relation for a conditional. A function $post_{[\varphi]} : A \to A$ is a *sound successor transformer* for the conditional $[\varphi]$ if the set of structures obtained by applying $post_{[\varphi]}$ overapproximates the structures obtained by applying the transition relation: $Rel_{[\varphi]}(\gamma(a)) \subseteq \gamma(post_{[\varphi]}(a))$. Dually, a function $\tilde{pre}_{[\varphi]} : A \to A$ is a *sound predecessor transformer* for the conditional $[\varphi]$ if the set of structures obtained by applying $\tilde{pre}_{[\varphi]}$ underapproximates the structures obtained by applying the transition relation backwards: $\gamma(\tilde{pre}_{[\varphi]}(a)) \subseteq \{s \mid Rel_{[\varphi]}(\{s\}) \subseteq \gamma(a)\}$.

To relate these transformers to deduction and abduction functions, we lift the functions above to operate on the Lindenbaum-Tarski algebra. We write $\equiv$ instead of $\equiv_{\mathcal{L}}$ for brevity.

$$ded^{\equiv}_\varphi : \mathcal{L}/{\equiv} \to \mathcal{L}/{\equiv} \qquad ded^{\equiv}_\varphi([\theta]_\equiv) \mathrel{\hat{=}} \bigwedge \{[\psi]_\equiv \mid \psi \in ded_\varphi([\theta]_\equiv)\}$$
$$abd^{\equiv}_\varphi : \mathcal{L}/{\equiv} \to \mathcal{L}/{\equiv} \qquad abd^{\equiv}_\varphi([\theta]_\equiv) \mathrel{\hat{=}} [\psi]_\equiv \text{ for some } \psi \in abd_\varphi([\theta]_\equiv)$$

The result of deduction on the Lindenbaum-Tarski algebra is a meet of equivalence classes of formulae in order to obtain the strongest consequence possible. Assuming that an equivalence class consists of only finitely many formulae, this result is well-defined. If an equivalence class is not finite, a finite number of representatives can be used instead. The lift of abduction is not the dual of deduction. Instead, the result of lifting abduction is the equivalence class of one of

the formulae that result from abduction. This is because we want the weakest possible abduction but our logics lack disjunction. Using the lattice-theoretic join in algebras where the join exists may lead to unsound abduction.

Theorem 4. *Let ded_φ and abd_φ be sound deduction and abduction transformers and $(\mathcal{L}, \vdash_\mathcal{L})$ be a logical language closed under conjunction with a calculus that extends $\vdash_{\text{CORE}}$. Then, the lifted functions $ded^{\equiv}_\varphi$ and $abd^{\equiv}_\varphi$ are sound successor and predecessor transformers for the conditional $[\varphi]$.*

We have not modelled transformers for assignment because we have not identified a satisfying treatment of substitution and quantification that factors through the Lindenbaum-Tarski construction.

5 Abstract Interpreters as Second-Order Solvers

An abstract interpreter for reachability analysis combines a lattice with transformers to derive program invariants. We have shown that lattices approximate state formulae, and that deduction and abduction functions approximate transition formulae. We now show that the steps in fixed point iteration can be understood as second-order propagation. Logically, a fixed point iterator can be viewed as an SMT solver for trace formulae.

We introduce *abstract assignments* to model approximations of trace formulae. We have chosen this term to emphasise the similarity to partial assignments in SAT solvers. Let $(A, \sqsubseteq, \sqcap)$ be a lattice that is an abstraction of the lattice of states $(\mathcal{P}(\mathit{State}), \subseteq, \cap)$. Recall that SVar is the set of second-order variables. The *lattice of abstract assignments* is $(\mathit{Asg}_A, \sqsubseteq, \sqcap)$, where $\mathit{Asg}_A \mathrel{\hat{=}} \mathit{SVar} \to A$ is the set of abstract assignments and the order and meet are defined pointwise.

Let Struct be the set of pairs (τ, σ) of WS1S(T) structures. We show that the lattice of abstract assignments is an abstraction of $(\mathcal{P}(\mathit{Struct}), \subseteq, \cup)$. An abstract assignment represents sets of WS1S(T) structures analogous to the way a partial assignment in a DPLL-based SAT solver represents all assignments that extend to undefined variables. The set of WS1S(T) structures represented by an abstract assignment is given by the concretization $\mathit{conc} : \mathit{Asg}_A \to \mathcal{P}(\mathit{Struct})$ below.

$$\mathit{conc}(\mathit{asg}) \mathrel{\hat{=}} \{(\tau, \sigma) \mid \text{for all } X \in \mathit{SVar}.\, \{\tau(i) \mid i \in \sigma(X)\} \subseteq \gamma(\mathit{asg}(X))\}$$

Explained in terms of states, an abstract assignment represents structures by the set of states at each program location but forgets the order between states.

We present the run of an abstract interpreter as a solver for $\mathit{Reach}_{G,L}$. An abstract interpreter begins with the variable map $\lambda Y.\top$ indicating that nothing is known about the satisfiability of $\mathit{Reach}_{G,L}$, so every structure is potentially a model of $\mathit{Reach}_{G,L}$. An abstract assignment is updated using a propagation rule. If a location is not reachable, the formula is unsatisfiable, as deduced by the conflict rule.

$$\mathit{asg} \rightsquigarrow \mathit{asg}[X_v \mapsto d], \quad \text{where } d = \bigsqcup_{(u,v)\in E} \left\{ \mathit{post}_{(u,v)}(\mathit{asg}(X_v)) \right\} \qquad \textsf{Propagate}$$

$$\mathit{asg} \rightsquigarrow \textsf{unsat} \qquad \text{if } \mathit{asg}(X_v) = \bot, \text{ for some } v \in L \qquad \textsf{Conflict}$$

Propagation modifies an abstract assignment similar to the way Boolean constraint propagation (BCP) updates a partial assignment with two key differences. One is that rather than values, the assignment is updated with elements of a lattice. The second is that in BCP, before decisions are made, every value that is undefined becomes tt or ff, becoming strictly more precise. With abstract assignments, the assignments to X_v within an SCC with more than one node, will, in general, get weaker. We have not modelled termination concerns, which are addressed with widening and narrowing operators. The theorem below expresses the soundness of fixed point iteration without widening and narrowing in terms of satisfiability.

Theorem 5. *If the repeated application of the propagation and conflict rules leads to* unsat*, the formula* $Reach_{G,L}$ *is unsatisfiable.*

6 Related Work, Discussion and Conclusion

The development of novel combinations of automated deduction and abstract interpretation is a driving force behind much current research, which we surveyed in the introduction. Consult the dissertations [14, 22] and Dagstuhl seminar notes [17] for a detailed treatment of this research. Such work has been applied to design new SMT solvers [4], program analyzers [10, 23], and has helped automate the construction of program analyzers [27, 24, 25].

However, our experience has been that crucial aspects of solvers such as branching and conflict analysis heuristics are difficult to characterize lattice-theoretically due to their combinatorial nature. In this work, we have initiated a complementary research programme by giving logical characterizations of instances of abstract interpretation. To relate logics to lattices, we have combined ideas from substructural logic with the Lindenbaum-Tarski construction [20] and Tarski's algebraic treatment of deduction.

A more abstract approach would be to use the framework of Stone duality, which uses category theory to relate lattices, topological spaces and logics. Stone duality was extended to programs by Abramsky [1] who related domains in semantics to intuitionistic, modal, fixed point logic. Jensen [16] applied Abramsky's work to extract a logic from a specific abstract interpretation: strictness analysis.

In this paper, we have modelled logics that lack disjunction and have weaker proof systems than those considered in approaches based on Stone duality. The closest study to ours is by Schmidt [21], who articulated the idea that the partial order of an abstract domain defines its proof theory. In terms of algebraic logic, Schmidt's work can be understood as identifying logical characterization of families of lattices in abstract interpretation as free algebras of the Lindenbaum-Tarski construction. In comparison, our work has focused on characterizing specific lattices as theories.

Conclusion. This work is a first step towards a logical description of the internals of an abstract interpreter in the mathematical and algorithmic vocabulary of SAT

and SMT solvers. The results in this paper make precise widespread folk intuition about the logical basis of certain abstract interpreters. Though our results are unsurprising, we believe the techniques we have used are novel and connect ideas from substructural logic, algebraic logic and satisfiability research. In using Büchi's construction, we have also connected abstract interpretation with the automata-theoretic approach to logic and verification. Folk knowledge asserts that transformers for assignments provide a form of quantifier elimination. We have not modelled these transformers here because we are missing a rigorous treatment that integrates with the Lindenbaum-Tarski construction.

In terms of solver architecture, the simple abstract interpreter we have considered can be viewed as a second-order theory solver that only updates assignments. This view provides a direct route to integrating branching heuristics, conflict analysis, and variable selection. We have begun these investigations and hope that this exposition enables the automated deduction community to participate in the same.

References

1. S. Abramsky. *Domain Theory and the Logic of Observable Properties*. PhD thesis, University of London, 1987.
2. A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35:79–111, November 1999.
3. N. Bjørner, B. Duterte, and L. de Moura. Accelerating lemma learning using joins – DPLL(⊔). In *LPAR*, 2008.
4. M. Brain, V. D'silva, A. Griggio, L. Haller, and D. Kroening. Deciding floating-point logic with abstract conflict driven clause learning. *Formal Methods in Systems Design*, 45(2):213–245, Oct. 2014.
5. J. R. Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford Univ. Press, 1960.
6. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM Press, 1977.
7. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282. ACM Press, 1979.
8. P. Cousot, R. Cousot, and L. Mauborgne. Theories, solvers and static analysis by abstract interpretation. *J. ACM*, 59(6):31:1–31:56, Jan. 2013.
9. V. D'Silva, L. Haller, and D. Kroening. Abstract conflict driven learning. In *POPL*, pages 143–154. ACM Press, 2013.
10. V. D'Silva, L. Haller, D. Kroening, and M. Tautschnig. Numeric bounds analysis with conflict-driven learning. In *TACAS*, pages 48–63. Springer, 2012.
11. S. Grebenshchikov, N. P. Lopes, C. Popeea, and A. Rybalchenko. Synthesizing software verifiers from proof rules. In *PLDI*, pages 405–416. ACM Press, 2012.
12. B. S. Gulavani, S. Chakraborty, A. V. Nori, and S. K. Rajamani. Automatically refining abstract interpretations. In *TACAS*, volume 4963 of *LNCS*, pages 443–458. Springer, 2008.
13. S. Gulwani and A. Tiwari. Combining abstract interpreters. In *PLDI*, pages 376–386. ACM Press, 2006.
14. L. C. R. Haller. *Abstract Satisfaction*. PhD thesis, University of Oxford, 2014.

15. W. R. Harris, S. Sankaranarayanan, F. Ivančić, and A. Gupta. Program analysis via satisfiability modulo path programs. In *POPL*, pages 71–82, 2010.
16. T. P. Jensen. Strictness analysis in logical form. In *FPCA*, pages 352–366. Springer, 1991.
17. D. Kroening, T. W. Reps, S. A. Seshia, and A. V. Thakur. Decision procedures and abstract interpretation (Dagstuhl seminar 14351). *Dagstuhl Reports*, 4(8):89–106, 2014.
18. K. R. M. Leino and F. Logozzo. Using widenings to infer loop invariants inside an SMT solver, or: A theorem prover as abstract domain. In *Workshop on Invariant Generation*, pages 70–84. RISC Report 07-07, 2007.
19. M. Pelleau, C. Truchet, and F. Benhamou. Octagonal domains for continuous constraints. In *CP*, pages 706–720, 2011.
20. H. Rasiowa and R. Sikorski. *The mathematics of metamathematics.* Polish Academy of Science, Warsaw, 1963.
21. D. A. Schmidt. Internal and external logics of abstract interpretations. In *VMCAI*, pages 263–278, Berlin, Heidelberg, 2008. Springer-Verlag.
22. A. V. Thakur. *Symbolic Abstraction: Algorithms and Applications.* PhD thesis, The University of Wisconsin - Madison, 2014.
23. A. V. Thakur, J. Breck, and T. W. Reps. Satisfiability modulo abstraction for separation logic with linked lists. In *SPIN*, pages 58–67, 2014.
24. A. V. Thakur, M. Elder, and T. W. Reps. Bilateral algorithms for symbolic abstraction. In *SAS*, pages 111–128, 2012.
25. A. V. Thakur, A. Lal, J. Lim, and T. W. Reps. Posthat and all that: Automating abstract interpretation. *Electric Notes in Theoretical Computer Science*, 311:15–32, 2015.
26. A. V. Thakur and T. Reps. A generalization of Stålmarck's method. In *SAS*. Springer, 2012.
27. A. V. Thakur and T. W. Reps. A method for symbolic computation of abstract operations. In *CAV*, 2012.
28. A. Tiwari and S. Gulwani. Logical interpretation: Static program analysis using theorem proving. In *CADE*, pages 147–166, 2007.
29. C. Truchet, M. Pelleau, and F. Benhamou. Abstract domains for constraint programming, with the example of octagons. In *Symbolic and Numeric Algorithms for Scientific Computing*, pages 72–79, 2010.
30. S. van den Elsen. Weak monadic second-order theory of one successor. Seminar: Decision Procedures, 2012.
31. M. Y. Vardi and T. Wilke. Automata: from logics to algorithms. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].*, pages 629–736, 2008.