Journal of Automated Reasoning Special Issue on CADE-25 manuscript No.
(will be inserted by the editor)

Abstract Interpretation as Automated Deduction

Vijay D'Silva · **Caterina Urban**

Received: date / Accepted: date

Abstract Automata theory, algorithmic deduction and abstract interpretation provide the foundation behind three approaches to implementing program verifiers. This article is a first step towards a mathematical translation between these approaches. By extending Büchi's theorem, we show that reachability in a control flow graph can be encoded as satisfiability in an extension of the weak, monadic, second-order logic of one successor. Abstract interpreters are, in a precise sense, sound but incomplete solvers for such formulae. The three components of an abstract interpreter: the lattice, transformers and iteration algorithm, respectively represent a fragment of a first-order theory, deduction in that theory, and second-order constraint propagation. By inverting the Lindenbaum-Tarski construction, we show that lattices used in practice are subclassical first-order theories.

Keywords Abstract Interpretation · Deduction · Lindenbaum-Tarski Construction

1 Introduction

Verification with satisfiability solvers, the automata-theoretic approach and abstract interpretation provide three approaches for checking if an assertion in an imperative program may be violated. At a high level, each technique can be viewed as proving a statement of the form below.

$$\vdash \mathit{Exec}(P) \implies \neg \mathit{Err} \qquad \mathscr{L}(\mathscr{A}_P \times \mathscr{A}_{\mathit{Err}}) = \emptyset \qquad [\![P]\!]_A \sqcap [\![\mathit{Err}]\!]_A \sqsubseteq \bot$$

In solver-based approaches, bounded executions of a program P are encoded as a formula $\mathit{Exec}(P)$. An assertion is not violated if the formula $\mathit{Exec}(P) \implies \neg \mathit{Err}$ is

Vijay D'Silva
Google Inc., San Francisco
E-mail: vijay.dsilva@gmail.com

Caterina Urban
ETH Zürich
E-mail: caterina.urban@inf.ethz.ch

true, which is determined by checking if $Exec(P) \wedge Err$ is satisfiable [Bjørner and de Moura, 2014]. In the automata-theoretic approach, executions of a program and erroneous executions are represented using automata. The assertion is not violated if the language of the product automaton $\mathscr{L}(\mathscr{A}_p \times \mathscr{A}_{Err})$ is empty [Vardi and Wolper, 1994]. In abstract interpretation, an assertion is verified by computing an invariant and checking if the invariant contains an error state. The invariant and error states are represented as elements of a lattice A. The assertion is not violated if the meet $[\![P]\!]_A \sqcap [\![Err]\!]_A$ is the bottom element of the lattice A [Cousot and Cousot, 1977].

These approaches have complementary strengths, which we now review. The strengths of SMT solvers include efficient Boolean reasoning, complete reasoning in certain theories, theory combination, proof generation and interpolation. The strengths of the automata-theoretic approach are the use of automata to represent infinitary behaviour and the use of graph algorithms to reason about temporal properties. 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 to generalize from partial information about a program to invariants.

The complementary strengths of these approaches has led to multiple theoretical and practical effort to combine them. D'antoni and Veanes [2015] and Heizmann et al [2013] describe two different efforts that combine automata and SMT solvers. The notes of the Dagstuhl seminar by Kroening et al [2014], provide an overview of research combining abstract interpretation and satisfiability, while the work of Dalla Preda et al [2015] generalizes automata to operate on abstract domains. Despite these advances, a major impediment to combining these approaches is that they are formulated in terms of different mathematical objects. These mathematical differences translate into practical differences in the interfaces implemented by tools using each approach, which leads to further impediments to combining approaches. Conceptually, solver algorithms are formulated in terms of models and proofs, automata-theoretic algorithms in terms of graphs, and abstract interpretation is presented in terms of lattices and fixed points.

Content and Contribution. This paper applies classical results in logic to relate abstract interpreters for reachability analysis with the automata-theoretic approach and the satisfiability-based approach. One conceptual consequence of our work is to show that abstract interpreters are, in a precise sense, solvers for satisfiability of a special family of formulae in monadic, second-order logic. A second, conceptual consequence of our work is in showing that certain lattices used in abstract interpreters are subclassical fragments of first-order theories.

Fig. 1 summarises the technical concepts in the paper.Büchi [1960] showed that the sets definable by the weak, monadic, second-order logic of one successor (WS1S) are regular languages. The modern proof of this statement involves an encoding of an automaton in WS1S, and a compilation of a WS1S formula into an automaton. Intuitively, the language of an automaton $\mathscr{A}$ is an element $\mathscr{L}(\mathscr{A})$ of the lattice of languages and the set $mod(\varphi)$ of models of a formula φ is an element of lattice of subsets of structures over which formulae are interpreted. In § 2 we adapt the translation of automata to WS1S to encode erroneous executions in a control-flow graphs (CFGs) as a satisfiability problem.

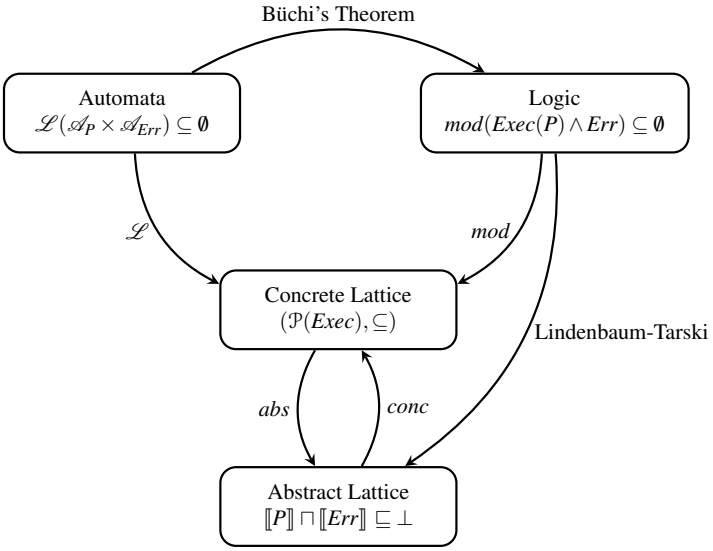


Fig. 1 To check if an error location *Err* is reachable in a program *P*, one can check if the language of an automaton $\mathscr{L}(\mathscr{A})$ is empty, if a formula has no models $mod(\varphi)$, or if an element of a lattice is bottom. Büchi showed how to translate directly between automata and WS1S and by applying his construction, we obtain a logic for describing erroneous executions in a control-flow graph. Abstract interpreters solve such formulae using approximations of the lattice of sets of executions. The Lindenbaum-Tarski construction allows for generating a lattice from a logic and by inverting it, we identify logics and proof systems corresponding to lattices in abstract interpretation.

Abstract Interpretation		Logic	
Lattice element	a	$[\varphi]_{\equiv}$	Equivalence class
Partial order	$\sqsubseteq$	$\vdash / \equiv$	Proof system
Lattice	A	$\mathscr{L}/\equiv$	Set of equivalence classes
Concretization	γ	$\models^{-1}$	Satisfaction

Table 1 A logical view of a lattice in abstract interpretation.

Our second contribution, in § 3, is to show how a simple abstract interpreter is, in a precise sense, a solver for satisfiability of WS1S(T) formulae. The main components of an abstract interpreter are a lattice, monotone functions called transformers, and an invariant map that associates lattice elements with control locations. We show that these components are, respectively, approximations of first-order structures, of relations between first-order structures, and sequences of first-order structures. An abstract interpreter performs constraint propagation using assignments to second-order variables, similar to propagation techniques in SAT solvers.

Our third contribution, summarized in Table 1 with details in § 4 and § 5 is to give a logical account of certain lattices used in abstract interpretation. A lattice can be viewed as a logic in which the concretization function defines the model-theoretic semantics and the partial order defines the proof-theoretic semantics. We use the

Lindenbaum-Tarski construction [Surma, 1982] to show that the proof systems we identify characterize existing lattices up to isomorphism. In particular, we give logical characterizations of the lattices of signs, constants and intervals, all of which are commonly used and studied in abstract interpretation.

Note. This article extends preliminary results announced in [D'Silva and Urban, 2015a] with new formalization, results and complete proofs. The characterization of an analyzer as a solver in § 3 adds a formalization and proof of the soundness of propagation and clarifies the connection to propagation in SAT solvers. § 4 and § 5 provide complete proofs of the proof-theoretic material and add a model-theoretic justification for the logics we choose.

2 Reachability as Second-Order Satisfiability

In this section, we introduce a new logic in which one can encode reachability of a control location as a satisfiability problem without an apriori bound on the length of an execution. We show how the logical encoding of reachability follows from a straightforward extension of Büchi's theorem.

2.1 Weak Monadic Second Order Theories of One Successor

Notation. We use $\triangleq$ 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 $f(c)$.

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 as such because when modelling programs, first-order structures model states, pairs of first-order structures model transitions, and sequences of first-order structures model program executions.

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

The formula $suc(x) = t$ expresses that the value of the first-order variable x after a transition equals the value of the term t before the transition. The formula $Suc(i, j)$ expresses that the position j on the trace occurs immediately after position i. The formulae $\varphi(i)$ and $\psi(i)$ express that the state formula φ and the transition formula ψ hold at position i on a trace. Though similar, WS1S(T) and WS1S are incomparable, because WS1S(T) contains first-order variables and terms, which WS1S does not, but WS1S allows for second-order quantification, which WS1S(T) does not.

State formulae are interpreted with respect to a theory $\mathscr{T}$ given by a first-order interpretation (Val, I), which defines functions $I(f)$, relations $I(P)$, and equality $=_{\mathscr{T}}$ 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, $s \models_{\mathscr{T}} \varphi$ denotes that s is a model of φ in the theory $\mathscr{T}$.

$$s \models_{\mathscr{T}} P(t_0,\ldots,t_n) \text{ if } ([\![t_0]\!]_s,\ldots,[\![t_n]\!]_s) \in I(P)$$
$$s \models_{\mathscr{T}} \varphi_1 \wedge \varphi_2 \text{ if } s \models_{\mathscr{T}} \varphi_1 \text{ and } s \models_{\mathscr{T}} \varphi_2 \qquad\qquad s \models_{\mathscr{T}} \neg\varphi \text{ if } s \not\models_{\mathscr{T}} \varphi$$

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

$$(r, s) \models suc(x) = t \text{ if } [\![x]\!]_s =_{\mathscr{T}} [\![t]\!]_r$$

A trace is a finite sequence of states and a position assignment associates position variables and second order variables with positions on a trace. Formally, 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, as defined below.

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

A structure (τ, σ) satisfies $X(i)$ if the position i is in the set of positions associated with X. Note that the semantics of a transition formula $\psi(i)$ 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 $\vee$, $\Rightarrow$ and $\forall$, and write $\Phi \models \Psi$ for $\models \Phi \Rightarrow \Psi$.

Example 1 We give examples of WS1S(T) formulae, which are also WS1S formulae and which we will use later in the paper. The WS1S formula $First(i) \triangleq \forall j.\neg Suc(j,i)$ is true at the first position on a trace and $Last(i) \triangleq \forall j.\neg Suc(i,j)$ is true at the last position. See [Vardi and Wilke, 2008; van den Elsen, 2012] for more examples. $\triangleleft$

The standard encoding of transitive closure of the successor relation in WS1S involves second-order quantification, so this encoding does not carry over. There may be other ways to encode transitive closure, depending on the underlying theory, but we do not explore this direction here because second-order quantification is not required for the specific class of formulae that we consider.

2.2 Encoding Reachability in WS1S(T)

Büchi showed that a language is regular if and only if it arises as the set of models of a WS1S formula. The modern proof that a regular language is definable in WS1S [Vardi and Wilke, 2008; van den Elsen, 2012] encodes the structure and acceptance condition of a finite automaton using second-order variables. We now extend this construction to encode the set of executions that reach a location in a control-flow graph (CFG) as the models of a WS1S(T) formula. In the next section, we will show how an abstract interpreter is, in a precise sense, a solver for this 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 $G = (Loc, E, \mathtt{in}, Ex, stmt)$ consists of a finite set of locations Loc including an initial location $\mathtt{in}$, a set of exit locations Ex, edges $E \subseteq Loc \times Loc$, and a labelling $stmt : E \to Cmd$ of edges with commands. To simplify the presentation, we require that every location is reachable from $\mathtt{in}$, and that exit locations have no successors.

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 $Same_V$ below expresses that variables in the set V are not modified in a transition and $Trans_c$ is the *transition formula* for a command c.

$$Same_V \triangleq \bigwedge_{x \in V} suc(x) = x \qquad Trans_c \triangleq \begin{cases} b \implies Same_{Vars} & \text{if } c = [b] \\ suc(x) = t \wedge Same_{Vars \setminus \{x\}} & \text{if } c = x := t \end{cases}$$

The *transition relation* of 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 $stmt(e)$. 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 $Reach_{G,L}$ 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*.

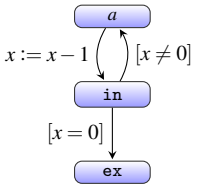
$$\begin{aligned} \mathit{Reach}_{G,L} \triangleq\ & \forall i.\mathit{First}(i) \implies X_{\mathtt{in}}(i) \\ & \wedge \left(\bigwedge_{v \in \mathit{Loc}} \forall i.\forall j.X_v(j) \wedge \mathit{Suc}(i,j) \implies \bigvee_{(u,v)\in E} \mathit{Trans}_{(u,v)}(i) \wedge X_u(i) \right) \\ & \wedge \left(\forall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

In intuitive terms, in a model (τ,σ) of the formula above, σ describes control-flow and τ describes data-flow. The trace τ contains states but not 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 the first location of an execution is `in`. The final constraint ensures that execution ends a location 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 [$\Rightarrow$] If a location $w \in L$ in the CFG G is reachable, there is an execution $\rho \triangleq (u_0,s_0),\ldots,(u_{k-1},s_{k-1})$ with $u_0 = \mathtt{in}$ and $u_{k-1} = w$. Define the structure (τ,σ) with $\tau \triangleq s_0,\ldots,s_{k-1}$ and $\sigma \triangleq \{X_u \mapsto \{i \mid \rho(i) = (u,s), s \in \mathit{State}\} \mid u \in \mathit{Loc}\}$. There are no first-order position variables in the domain of σ because all such variables are bound in $\mathit{Reach}_{G,L}$. 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 and j is the successor of i, it must be that $j = i+1$ and 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 $\mathit{Trans}_{(u_i,v)}$.
[$\Leftarrow$]Assume (τ,σ) is a model of $\mathit{Reach}_{G,L}$, where τ is a trace of length k. Define a sequence ρ with $\rho(i) \triangleq (u,\tau(i))$ where $i \in \sigma(X_u)$. As σ induces a partition of $[0,k-1]$, 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 `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 variation of a known logic. By viewing the problem of reachability in a CFG in terms of satisfiability of $\mathit{Reach}_{G,L}$, we can connect the abstract interpretation approach with the automata-theoretic approach and with satisfiability-based approaches. Abstract interpreters operate on CFGs, which can be viewed as generalizations of automata. In addition, as we show in subsequent sections, abstract interpreters can be viewed as solving $\mathit{Reach}_{G,L}$ using deductive techniques. Thus, at a conceptual level, abstract interpreters use a hybrid of the automata-theoretic and logical approaches.



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

Fig. 2 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`.

Example 2 A CFG G and the formula $\mathit{Reach}_{G,Ex}$ for a program with an integer variable x are shown in Fig. 2. Executions that start with a strictly negative value of x neither terminate nor reach `ex`. 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 $\tau = (x{:}1),(x{:}1),(x{:}0),(x{:}0)$. 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 $\mathit{Reach}_{G,Ex}$. ◁

This example highlights important differences between WS1S(T) and encodings of program correctness in terms of set constraints [Aiken, 1999] or second-order Horn clauses [Grebenshchikov et al, 2012]. Invariants, which are solutions to constraints generated in these approaches, are formulae whose models include all reachable states. In contrast, a model of $\mathit{Reach}_{G,L}$ only involves states that occur on a single execution. Note that other formalisms allow for encoding a broader range of problems. Our intent with WS1S(T) though is to give a logical account of abstract interpreters, and not to solve arbitrary formulae.

3 Abstract Interpreters as Second-Order Solvers

Three crucial components of an abstract interpreter are a lattice, transformers, which are monotone functions on the lattice, and an invariant map, which maps locations in a CFG to lattice elements. An abstract interpreter updates the invariant map by applying transformers to lattice elements. We now show how an abstract interpreter performs second-order constraint propagation.

Lattice Theory. We recall elements of lattice theory. A lattice $(A,\sqsubseteq,\sqcap,\sqcup)$ is a set A equipped with a partial order $\sqsubseteq$, 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*.

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 $f \sqcap g$ is a function that maps x in S to $f(x) \sqcap g(x)$.

Let $(\mathcal{P}(S),\subseteq)$ be the lattice of all subsets of S ordered by inclusion. A bounded lattice $(A,\sqsubseteq)$ is an *abstraction* of $(\mathcal{P}(S),\subseteq)$ if there exists a monotone *concretization*

function $\gamma : A \to \mathcal{P}(S)$ satisfying that $\gamma(\bot) = \emptyset$ and $\gamma(\top) = S$. A *transformer* is a monotone function on a lattice. A transformer $g : A \to A$ is a *sound abstraction* of $f : \mathcal{P}(S) \to \mathcal{P}(S)$ if for all a in A, $f(\gamma(a)) \subseteq \gamma(g(a))$.

Propagation Rules. Propagation rules in a solver describe updates to data-structure that represents potential solutions using constraints that were present in or deduced from an input formula. We present a formalization of propagation based on the formalization of satisfiability algorithms by Nieuwenhuis et al [2006]. We only consider propagation, which is the main operation of an abstract interpreter. We first recall the *unit rule* used in SAT solvers.

Example 3 Recall that a literal is a Boolean variable or its negation and a clause is a disjunction of literals. An *assignment* $\sigma : \mathit{Vars} \to \{\mathtt{tt}, \mathtt{ff}\}$ maps each variable to a Boolean value, while in a *partial assignment* $\pi : \mathit{Vars} \to \{\mathtt{tt}, \mathtt{ff}, \top\}$, a variable may also be unknown, denoted $\top$. A partial assignment π' *extends* π if for all p with $\pi(p) \neq \top$, it holds that $\pi(p) = \pi'(p)$. The definition of an assignment extending a partial assignment is similar.

A partial assignment π satisfies a variable p if $\pi(p) = \mathtt{tt}$, satisfies the literal $\neg p$ if $\pi(p) = \mathtt{ff}$, and satisfies a clause if it satisfies at least one literal in the clause. The partial assignment is *in conflict* with a clause if it makes every literal in the clause $\mathtt{ff}$.

The unit rule asserts that if π is a partial assignment and $C \vee \ell$ is a clause, and the variable p in ℓ has the unknown value in π, and π is in conflict with C, then π must be extended to π' that satisfies ℓ. The unit rule has the property that every assignment σ that extends π and satisfies $C \vee \ell$ also extends π'. During Boolean Constraint Propagation in a SAT solver, a partial assignment π is repeatedly extended by the unit rule. If some extension of π derived by unit rule applications is in conflict with a clause in the formula, no extension of π satisfies the formula. $\lhd$

Let *Form* be the set of formulae in a logic and *Struct* be the set of structures over which formulae are interpreted. The function $\mathit{mod} : \mathit{Form} \to \mathcal{P}(\mathit{Struct})$ maps a formula to its set of models. Let $(A, \sqsubseteq)$ be an abstraction of $(\mathcal{P}(\mathit{Struct}), \subseteq)$ with concretization γ. We view the lattice A as a data-structure representing potential solutions of a formula. A *propagation rule* is a set of rules of the form $(\varphi, a) \rightsquigarrow a'$ that describe how an element a is modified given a formula φ. A propagation rule is *sound* if every model of φ in a is also in a': $\mathit{mod}(\varphi) \cap \gamma(a) \subseteq \gamma(a')$.

Example 4 Consider the set *PAsg* consisting of partial assignments over variables *Vars* and an element $\bot$. Define a relation $\pi \sqsubseteq \pi'$ to hold if π is $\bot$ or if π extends π'. D'Silva et al [2013] showed that $(\mathit{PAsg}, \sqsubseteq)$ is an abstraction of $(\mathcal{P}(\mathit{Struct}), \subseteq)$ in which the concretization γ maps π to the set of assignments that extend it. Let *Form* be a set of CNF formulae. The unit rule contains elements $(\varphi, \pi) \rightsquigarrow \pi'$ where π' extends π to satisfy some clause in φ. $\lhd$

We introduce *abstract assignments* to model abstractions of WS1S(T) structures. Consider the lattice $(\mathcal{P}(\mathit{State}), \subseteq, \cap)$, where *State* is $\mathit{Vars} \to \mathit{Val}$. Let $(A, \sqsubseteq, \sqcap)$ be an abstraction of this lattice with concretization γ. Recall that *SVar* is the set of second-order variables. An *abstract assignment* is an element of $\mathit{Asg}_A \mathrel{\hat{=}} \mathit{SVar} \to A$, which forms a lattice $(\mathit{Asg}_A, \sqsubseteq, \sqcap)$, in which the order and meet are defined pointwise. When

convenient, use lambda expressions to define abstract assignments. An abstract assignment abstracts WS1S(T) structures by retaining set of states at each program location but forgetting the order in which states are visited.

Lemma 1 *Let $(A, \sqsubseteq)$ be an abstraction of the lattice $\mathcal{P}(State, \subseteq)$ with concretization γ, and Struct be the set of* WS1S(T) *structures for interpreting formulae over a set SVar of second-order variables. The lattice of abstract assignments $(Asg_A, \sqsubseteq, \sqcap)$ is an abstraction of $(\mathcal{P}(Struct), \subseteq)$.*

Proof Let *Struct* be the set of pairs (τ, σ) of WS1S(T) structures. We show that the function $conc : Asg_A \to \mathcal{P}(Struct)$ below is a concretization function.

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

There are three properties to show. The least element of Asg_A is $\lambda X.\bot$ and the greatest is $\lambda X.\top$. Since $\gamma(\bot) = \emptyset$ and $\gamma(\top) = State$, we have that $conc(\lambda X.\bot) = \emptyset$ and $conc(\lambda X.\top) = Struct$. The monotonicity of *conc* follows from that of γ. □

In Lemma 1, the lattice A abstracts states and Asg_A abstracts WS1S(T) structures. Transitions are abstracted by transformers. A relation $R \subseteq S \times S$ generates a *successor transformer* $post_R : \mathcal{P}(S) \to \mathcal{P}(S)$ that maps every $X \subseteq S$ to its image $R(X)$. We write $post_c$ for the successor transformer of the transition relation of a command c, and similarly write $post_e$ for the transformer of the command labelling a CFG edge e. We write $apost_c$ and $apost_e$ for the corresponding abstract transformers.

An abstract interpreter can be viewed as a solver for the formula $Reach_{G,L}$. The abstract interpreter begins with the abstract assignment $\lambda Y.\top$ indicating that every structure may be a model of $Reach_{G,L}$. Abstract assignments are updated using the propagation rule below. If a location in L is not reachable, the formula is unsatisfiable, as deduced by the conflict rule.

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

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

We highlight two differences between Boolean constraint propagation (BCP) and propagation in an abstract interpreter. First, abstract assignments are updated with lattice elements, not extended with values. Second, BCP extends a partial assignment from π to $\pi' \sqsubseteq \pi$. However, if an abstract interpreter generates asg' from asg, then $asg'(X_v) \sqsubseteq asg(X_v)$ for locations v outside loops, but the converse may hold for locations inside a loop. The theorem below expresses the soundness of propagation.

Lemma 2 *If Asg_A is an abstraction of $(\mathcal{P}(Struct), \subseteq)$ and the abstract transformers are sound, the propagation rule is sound.*

Proof Consider a formula $Reach_{G,L}$, an abstract assignment asg and a structure (τ, σ) in $mod(Reach_{G,L}) \cap conc(asg)$. By definition of *conc*, the set $S_w \triangleq \{\tau(i) \mid i \in \sigma(X_w)\}$ is contained in $\gamma(asg(X_w))$ for every location w. Consider also a location v such that $\sigma(X_v) \neq \emptyset$ and $asg' = asg[X_v \mapsto d]$ as in the propagation rule, By the semantics of

WS1S(T), there exists $i+1$ in $\sigma(X_v)$ and some location u such that (u,v) is an edge, i is in $\sigma(X_u)$, and $(\tau(i),\tau(i+1))$ is a model of the transition formula $Trans_{(u,v)}$. From the definition of the successor transformer it follows that $\tau(i+1)$ is in $post_{(u,v)}(S_u)$, and by monotonicity, it also follows that $\tau(i+i)$ is in $\bigcup_{(u,v)} post_{(u,v)}(S_u)$. Since the abstract transformers are sound, we have that $\tau(i+1)$ is in $\bigsqcup_{(u,v)} apost_{(u,v)}(X_u)$. It follows that (τ,σ) is also in $conc(asg')$. □

Theorem 2 captures the use of an abstract interpreter as a solver for satisfiability of $Reach_{G,L}$. Since the abstract interpreter begins with $\lambda X.\top$, and propagation is sound, and unsat is only reached if $conc(asg)$ is the empty set we can soundly conclude that $Reach_{G,L}$ has no models.

Theorem 2 *If the repeated application of the propagation and conflict rules leads to* unsat*, the formula $Reach_{G,L}$ is unsatisfiable.*

4 Fragments of First-Order Theories

The description of an abstract interpreter as a solver in the previous section was agnostic of the domain and transformers used. We now identify logical theories corresponding to lattices used in practice and in § 5 we show how these theories characterize the lattices of constants, signs and intervals up to isomorphism.

4.1 First-Order Theories

All the theories we consider in this section are fragments of integer arithmetic. We assume a set of first-order variables *Vars*, the integer constants, functions for binary addition and multiplication, denoted $x+y$ and $x\cdot y$ respectively, and the relational symbols $<,\leq,>$ and $\geq$. All these symbols have their standard interpretation over the integers, $\mathbb{Z}$. For the remaining sections, a structure σ in $Struct \triangleq Vars \to \mathbb{Z}$ is a map from variables to integers. We assume the standard model theoretic semantics for formulae and write $\sigma \models \varphi$ if the structure σ satisfies the formula φ.

4.1.1 Logical Languages

A *logical language* $(\mathscr{L},\vdash_{\mathscr{L}},\models_{\mathscr{L}})$ consists of a set of formulae, a proof system, and an interpretation $\models_{\mathscr{L}} \subseteq (Struct \times \mathscr{L})$ of those formulae over structures. The logics we consider are interpreted over the same structures, so we usually omit $\models_{\mathscr{L}}$. We use logical languages to give proof-theoretic characterizations of the lattices in an abstract domain. We present the set of formulae with a grammar and present the proof system as a sequent-style calculus.

The three logical languages we introduce are *sign logic*, *constant logic* and *interval logic*. The names for these languages derive from the names of the abstract domains they model. Each grammar below defines a set of formulae in terms of atomic

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

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

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

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

Table 2 Proof rules for the core calculus $\vdash_{\text{CORE}}$ and its extensions. The core calculus contains rules for introduction (I), weakening (WL), contraction (CL) and permutation (PL) on the left, logical rules for false (ffL), in which $\varphi(x)$ has only one free variable x, true (ttR), and conjunction ($\wedge$L, $\wedge$R), and the cut rule.

formulae, logical constants and connectives. All the languages contain unary predicates and are closed under conjunction but not under disjunction or negation. The languages also contain the symbol tt for the logical constant true.

$$\varphi_{\mathscr{S}} ::= \mathsf{ff}_x \mid x < 0 \mid x = 0 \mid x > 0 \mid \varphi_{\mathscr{S}} \wedge \varphi_{\mathscr{S}} \mid \mathsf{tt} \qquad \mathscr{S}$$

$$\varphi_{\mathscr{C}} ::= \mathsf{ff}_x \mid x = k \mid \varphi_{\mathscr{C}} \wedge \varphi_{\mathscr{C}} \mid \mathsf{tt} \qquad \mathscr{C}$$

$$\varphi_{\mathscr{I}} ::= \mathsf{ff}_x \mid x \leq k \mid x \geq k \mid \varphi_{\mathscr{I}} \wedge \varphi_{\mathscr{I}} \mid \mathsf{tt} \qquad \mathscr{I}$$

The language $\mathscr{S}$ models the domain of signs. The three atomic formulae in $\mathscr{S}$ express that a variable is negative ($x < 0$), equal to zero ($x = 0$), or positive ($x > 0$). The language $\mathscr{C}$ models the domain of constants. This language contains a countable number of atomic formulae of the form $x = k$ expressing that a variable has a constant value. The language $\mathscr{I}$ models the domain of intervals. Its atomic formulae express upper bounds ($x \leq k$) and lower bounds ($x \geq k$) on the values of a variable.

All languages we introduce contain the logical constant tt. These languages model *non-relational domains*, meaning that they cannot express constraints that explicitly relate the values of two or more variables. For example, the language $\mathscr{C}$ contains the formula $x = 5 \wedge y = 5$ which implicitly expresses that x and y have the same value. However, the language cannot explicitly codify this information with the formula $x = y$. The non-relational nature of the domains we consider leads to a non-standard treatment of false. These languages do not contain the constant ff but instead have a family of logical constants ff_x parameterized by variables. We discuss the lattice theoretic basis for this non-standard treatment shortly.

4.1.2 The Core Calculus

Reasoning within the logical languages we consider is encoded by proof rules in a sequent calculus. Our sequents are of the form $\Gamma \vdash_{\mathscr{L}} \varphi$, where the *premise* Γ is a comma-separated sequence of formulae, and the *consequent* φ is a single, first-order formula. A *proof system* is a set of sequents. We use the standard notion of derivability of a sequent in a proof system. Two formulae are *inter-derivable* if the sequents $\varphi \vdash_{\mathscr{L}} \psi$ and $\psi \vdash_{\mathscr{L}} \varphi$ are both derivable.

We write $\bigwedge \Gamma$ for the conjunction of formulae in the sequence Γ. We use this syntax for convenience in this discussion and it is external to the logical languages we consider. A proof system $\vdash_{\mathscr{L}}$ is *sound* if every derivable sequent $\Gamma \vdash_{\mathscr{L}} \psi$ satisfies the classical implication $\bigwedge \Gamma \Rightarrow \psi$ with respect to the semantics defined by $\models$. The only formulae for which the semantics $\models$ is not standard are those involving $\mathtt{ff}_x$. Every constant $\mathtt{ff}_x$ has the same semantics: there is no structure σ for which $\sigma \models \mathtt{ff}_x$ holds. A proof system is *complete* if whenever the classical implication $\bigwedge \Gamma \Rightarrow \psi$ holds, the sequent $\Gamma \vdash_{\mathscr{L}} \psi$ is derivable.

Sequent calculi usually contain structural, logical and cut rules, and in the case of theories, also contain theory rules. Table 2 shows a core calculus $\vdash_{\text{CORE}}$ which contains the rules common to all theories we introduce. Our *introduction* rule (I) is standard. The *structural* rules for weakening on the left (WL), contraction on the left (CL), and permutation on the left (PL) are also standard. Due to the asymmetry in our definition of sequents, we only use rules for structural manipulation on the left. The cut rule (CUT) is also standard.

The core calculus has four logical rules. The *false-left* rule (ffL) allows for the derivation of a formula $\varphi(x)$ with exactly one free variable x from the premise $\mathtt{ff}_x$. For example, in the interval proof system that we introduce shortly, the formula $x \geq 5 \wedge x \leq 10$ is derivable from $\mathtt{ff}_x$, but $x \geq 5 \wedge y \geq 3$, is not derivable from $\mathtt{ff}_x$ because the second formula includes the variable y. Thus, arbitrary formulae are not derivable even from a premise that contains false. Our non-standard treatment of false is influenced by the way abstract domains reason about contradictions. We believe this is one counterintuitive way in which the logic of an abstract domain deviates from classical logics and proof systems.

The treatment of $\mathtt{tt}$ is the same as that of classical sequent calculi: $\mathtt{tt}$ is derivable from every premise. The syntactic asymmetry between true and false in our languages lifts to a corresponding asymmetry in the proof systems. We have a single rule for conjunction on the left ($\wedge$L) instead of the two standard rules. The rule for conjunction on the right is standard.

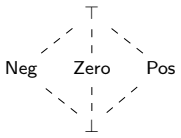
4.1.3 Theory Specific Rules

We introduce rules for reasoning within each theory. The reader should be warned that these logics have a restricted syntax and weak proof calculi, so the theorems derivable within the logic are rather uninteresting.

The sign calculus $\vdash_{\mathscr{S}}$, in Fig. 3, extends the core calculus with rules for deriving $\mathtt{ff}_x$ from conjunctions of atomic formulae. Every conjunction of atomic formulae in $\mathscr{S}$ is unsatisfiable in standard arithmetic. The theory rules allow us to derive $\mathtt{ff}_x$ from formulae such as $x < 0 \wedge x = 0$ or $x = 0 \wedge x > 0$. This logic supports no other form of theory-specific reasoning. We show later that this logic contains exactly three formulae that are not the logical constants and that are not inter-derivable.

Example 5 Fig. 4 shows a derivation of $x < 0 \vdash_{\mathscr{S}} x < 0 \wedge \mathtt{tt}$ and a derivation of $x < 0 \wedge \mathtt{tt} \vdash_{\mathscr{S}} x < 0$, thus showing that $x < 0$ and $x < 0 \wedge \mathtt{tt}$ are inter-derivable. $\lhd$

We now consider proof calculi for the constant and interval languages. Like the sign language, these languages only have atomic predicates over one variable. Un-



The sign calculus $\vdash_{\mathscr{S}}$

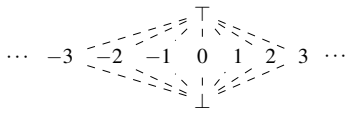
$$\vdash_{\text{CORE}}$$

$$\frac{}{\Gamma, x<0, x=0 \vdash \mathsf{ff}_x}\ \mathsf{ffR}_1 \qquad \frac{}{\Gamma, x=0, x>0 \vdash \mathsf{ff}_x}\ \mathsf{ffR}_2 \qquad \frac{}{\Gamma, x<0, x>0 \vdash \mathsf{ff}_x}\ \mathsf{ffR}_3$$

Fig. 3 The lattice of signs and the proof calculus $\vdash_{\mathscr{S}}$ for the sign logic.

$$\dfrac{\dfrac{}{x<0 \vdash_{\mathscr{S}} x<0}\ \text{I} \qquad \dfrac{}{x<0 \vdash_{\mathscr{S}} \mathsf{tt}}\ \mathsf{ttR}}{x<0 \vdash_{\mathscr{S}} x<0 \wedge \mathsf{tt}}\ \wedge\text{R} \qquad\qquad \dfrac{\dfrac{\dfrac{}{x<0 \vdash_{\mathscr{S}} x<0}\ \text{I}}{x<0, \mathsf{tt} \vdash_{\mathscr{S}} x<0}\ \text{WL}}{x<0 \wedge \mathsf{tt} \vdash_{\mathscr{S}} x<0}\ \wedge\text{L}$$

Fig. 4 A derivation in the sign calculus $\vdash_{\mathscr{S}}$.

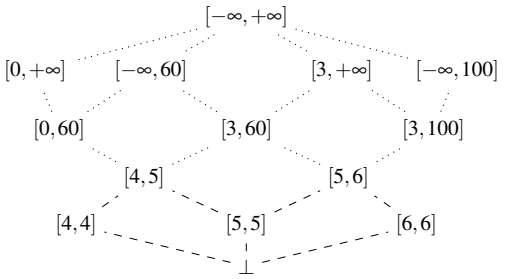


The constant calculus $\vdash_{\mathscr{C}}$

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

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

Fig. 5 The lattice of constants and the proof calculus $\vdash_{\mathscr{C}}$ for the constant logic.



The interval calculus $\vdash_{\mathscr{I}}$

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

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

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

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

Fig. 6 The lattice of intervals and the proof calculus $\vdash_{\mathscr{I}}$ for the interval logic.

like the sign languages these languages have a countably infinite number of atomic predicates. The constant calculus $\vdash_{\mathscr{C}}$ in Fig. 5 is similar to the sign calculus because both logics can only derive ff_x for each variable. The interval calculus $\vdash_{\mathscr{I}}$ in Fig. 6 contains rules for modifying upper and lower bounds on a variable. Specifically, the order on the integers dictates that one can weaken an upper bound $x \leq m$ to $x \leq n$ if m is smaller than n and a dual rule applies to lower bounds. It also contains a rule for deriving ff_x from inconsistent lower and upper bounds.

Example 6 An abstract interpreter computing intervals on variable values will manipulate formulae over multiple variables. Suppose the abstract interpreter has derived the bounds $x \in [7,5], y \in [-\infty,0]$ for some location in a program. We have deliberately written $[7,5]$, which would correspond to an empty interval, meaning that there is no feasible value for x and consequently, that the program location for which this bound was derived is unreachable. The conversion of $[7,5]$ to the empty interval is a calculation an abstract interpreter performs.

Logically, the bounds on x and y can be written as the sequence of predicates $y \leq 0, x \leq 5 \wedge x \geq 7$. The interval proof calculus allows us to derive the sequent $y \leq 0, x \leq 5 \wedge x \geq 7 \vdash_{\mathscr{J}} \mathsf{ff}_x \wedge y \leq 0$ showing that the bound is infeasible and the inconsistency arises from x. ◁

4.2 Soundness of the Proof Systems

Theorem 3 summarizes the soundness results we present, though, we prove the soundness of each calculus separately in Lemmas 3 to 6.

Theorem 3 *The proof calculi $\vdash_{\mathscr{S}}$, $\vdash_{\mathscr{C}}$, and $\vdash_{\mathscr{J}}$ are sound.*

We begin with the soundness of $\vdash_{\text{CORE}}$, which underlies all our calculi.

Lemma 3 *The core proof calculus $\vdash_{\text{CORE}}$ is sound.*

Proof We have to show that for every derivable sequent $\Gamma \vdash \varphi$, a structure σ that satisfies every formula in Γ, the structure also satisfies φ. The proof is by induction on the structure of a derivation.
(*Base Case*) The base cases are rules with no premise. The introduction (I) and truth ($\mathtt{tt}$R) rules are trivially sound. The false rule is sound because ff_x has no models.
(*Induction Step*) The induction hypothesis is that a sequent derived using the core calculus is sound. For the induction step, we have to show that a sequent obtained by applying rules in the core calculus to a soundly derived sequent is also sound. The weakening rule is sound because if $\bigwedge\Gamma$ implies ψ, then $\bigwedge\Gamma \wedge \varphi$ also implies ψ for the standard semantics of conjunction. The left and right conjunction rules are sound for the same reason. The contraction rule is sound because conjunction is idempotent and the permutation rule is sound because conjunction is commutative. □

The soundness proofs for the other proof calculi are extensions of this lemma.

Lemma 4 *The sign proof calculus $\vdash_{\mathscr{S}}$ is sound.*

Proof The proof extends the induction argument used for the soundness of the core calculus. Additional conditions for the base case are the rules $\mathsf{ff}\mathtt{R}_1$, $\mathsf{ff}\mathtt{R}_2$ and $\mathsf{ff}\mathtt{R}_3$, which are sound because their premises are unsatisfiable. The induction hypothesis and inductive step are unmodified so the sign calculus is sound. □

Lemma 5 *The constant proof calculus $\vdash_{\mathscr{C}}$ is sound.*

Proof The proof extends the induction arguments used for the core calculus. An additional condition for the base case is the rule $\mathrm{ff}_{\mathrm{R}_4}$, which is sound because its premise is unsatisfiable. The induction hypothesis and inductive step are unmodified so the constant calculus is sound. □

Lemma 6 *The interval proof calculus $\vdash_{\mathscr{I}}$ is sound.*

Proof The proof requires extensions to the base case and induction step of the induction arguments used for proving the soundness of the core calculus. An additional condition for the base case is the rule $\mathrm{ff}_{\mathrm{R}_5}$, which is sound because the premise is unsatisfiable. The rules UB-L, LB-L for strengthening bounds on the left, and the rules UB-R, LB-R for weakening bounds on the right are sound due to the order on the integers so the interval calculus is sound. □

5 Characterizing Lattices with First-Order Theories

We now apply the Lindenbaum-Tarski construction to generate a lattice from each logical language and then prove that the generated lattice is isomorphic to a lattice studied in program analysis.

5.1 The Lindenbaum-Tarski Construction and Logical Characterization

Tarski generalized a construction due to Lindenbaum to generate Boolean algebras from propositional calculus [Surma, 1982]. This construction has since be generalized to construct what is called the Lindenbaum-Tarski algebra of a logic. The essence of the construction is to quotient formulae in a logical language with respect to interderivability in that language. These equivalence classes form the carrier set of an algebra whose meet and join operations are defined by lifting conjunction and disjunction to equivalence classes. Derivability between formulae in equivalence classes defines a partial order on equivalence classes. The original construction has been extended to non-classical and first-order logics. We use a generalization of this construction to formulae with free variables due to Rasiowa and Sikorski [1963], sometimes called the *Rasiowa-Sirkoski construction of a Lindenbaum-Tarski algebra*. In the definition below, we write $[\varphi]_{\mathscr{L}}$ for the equivalence class of φ with respect to $\equiv_{\mathscr{L}}$.

Definition 1 Let $(\mathscr{L},\vdash_{\mathscr{L}})$ be a logical language and $\equiv_{\mathscr{L}}$ be an equivalence relation on formulae. A logic $(\mathscr{L},\vdash_{\mathscr{L}})$ that is closed under conjunction generates the *Lindenbaum-Tarski algebra* $Alg(\mathscr{L},\vdash_{\mathscr{L}}) = (\mathscr{L}/\equiv_{\mathscr{L}}, \preccurlyeq_{\mathscr{L}}, \curlywedge_{\mathscr{L}})$ in which the relation $\preccurlyeq_{\mathscr{L}}$ and operator $\curlywedge_{\mathscr{L}}$ are defined on equivalence classes as shown below.

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

The literature contains characterizations of Lindenbaum-Tarski algebras for various propositional and first-order logics. The classical propositional calculus provides an instructive example of the difference between what we study and what exists. The set of Boolean formulae over n propositional variables is countably infinite. The Lindenbaum-Tarski algebra over these formulae will contain 2^{2^n} elements, and is isomorphic to the *free Boolean algebra* over n generators. A Boolean algebras with 2^n elements for odd values of n will not be generated by this construction if one only uses the standard complete deductive systems for propositional calculus. The non-free Boolean algebras are Lindenbaum-Tarski algebras of *propositional theories*, meaning that they require additional axioms.

Example 7 The Lindenbaum-Tarski algebra of a propositional logic with one variable has the elements $\{\mathtt{ff}, p, \neg p, \mathtt{tt}\}$. A propositional logic with two variables generates a lattice with 16 elements. The four, least, non-bottom elements (atoms) are $\{p \wedge q, p \wedge \neg q, \neg p \wedge q, \neg p \wedge \neg q\}$ and the other elements are equivalent to disjunctions of these elements. The lattice $\mathcal{P}(\{a,b,c\})$ has eight elements and is not isomorphic to the Lindenbaum-Tarski algebra of either of these two logics. One way to generate the eight element Boolean algebra using the Lindenbaum-Tarski construction, is to add the axiom $p \wedge q$. Alternative axioms are $p \wedge \neg q$, $\neg p \wedge q$ and $\neg p \wedge \neg q$. $\lhd$

We show how lattices in abstract interpretation are Lindenbaum-Tarski algebras of first-order theories. Ex. 7 illustrates that different theories generate isomorphic algebras. A characterization of a lattice-based abstraction, defined below, consists of a *structural condition* and a *semantic condition*. The structural condition uses a proof system to generate the lattice, and the semantic condition uses the concretization function to express that the logic and the abstraction have the same semantics. There may be multiple isomorphisms h between the Lindenbaum-Tarski algebra and a lattice, intuitively corresponding to different axiomatizations. Only some isomorphisms will satisfy the second condition and capture the semantics of the abstraction.

Definition 2 Let $(A, \sqsubseteq)$ be an abstraction of $(\mathcal{P}(\mathit{Struct}), \subseteq)$ with concretization function $\gamma : A \to \mathcal{P}(\mathit{Struct})$. A logical language $(\mathscr{L}, \vdash_{\mathscr{L}}, \models_{\mathscr{L}})$ *characterizes* $(A, \sqsubseteq)$, if the following conditions hold.

1. There exists an isomorphism $h : \mathit{Alg}(\mathscr{L}, \vdash_{\mathscr{L}}, \models_{\mathscr{L}}) \to A$ between the Lindenbaum-Tarski algebra of $\mathscr{L}$ and A.
2. An element a in A concretizes to the same set of structures as the formula it represents: $\mathit{mod}(\varphi) = \gamma(h([\varphi]_{\mathscr{L}}))$.

We now study the Lindenbaum-Tarski algebras of logical languages corresponding to the sign, constants and interval languages. To avoid cumbersome distinctions between an equivalence class and its representatives, we use the following lemma that allows us to work directly with the syntactic representation of an equivalence class.

Lemma 7 *Let φ and ψ be two formulae. Then, $[\varphi]$and$[\psi]$ are the same equivalence class if and only if $\varphi \vdash \psi$ and $\psi \vdash \varphi$.*

Proof ($\Rightarrow$) Since $[\varphi]$ and $[\psi]$ are the same equivalence class there is a formula θ in $[\varphi]$ such that $\theta \vdash \varphi$, $\varphi \vdash \theta$, $\theta \vdash \psi$ and $\psi \vdash \theta$. We can now prove $\varphi \vdash \psi$ and $\psi \vdash \varphi$:

$$\frac{\varphi \vdash \theta \qquad \theta \vdash \psi}{\varphi \vdash \psi}\ \text{CUT} \qquad\qquad \frac{\psi \vdash \theta \qquad \theta \vdash \varphi}{\psi \vdash \varphi}\ \text{CUT}$$

($\Leftarrow$) Since $\varphi \vdash \psi \wedge \psi \vdash \varphi$, we have $\varphi \equiv_{\mathscr{L}} \psi$ and so $[\varphi]$ and $[\psi]$ are the same. □

5.2 Characterization of the Sign Proof Calculus

The lattice of signs $(\mathit{Sign}, \sqsubseteq)$ is depicted in Fig. 3. It consists of five elements $\mathit{Sign} = \{\bot, \mathsf{Neg}, \mathsf{Zero}, \mathsf{Pos}, \top\}$ with $\bot$ and $\top$ as the least and greatest elements in the order $\sqsubseteq$, and with the elements in $\{\mathsf{Neg}, \mathsf{Zero}, \mathsf{Pos}\}$ being pairwise incomparable. The concretization function $\gamma_{\mathit{Sign}} : \mathit{Sign} \to \mathcal{P}(\mathbb{Z})$ is defined below. For all the lattices we consider, the concretization of $\bot$ is $\emptyset$ and of $\top$ is $\mathbb{Z}$, so we skip these elements.

$$\gamma_{\mathit{Sign}}(\mathsf{Neg}) = \{n \mid n < 0\} \quad \gamma_{\mathit{Sign}}(\mathsf{Zero}) = \{n \mid n = 0\} \quad \gamma_{\mathit{Sign}}(\mathsf{Pos}) = \{n \mid n > 0\}$$

The lattice of *sign environments* $(\mathit{Vars} \to \mathit{Sign}, \sqsubseteq)$ is the pointwise lift of *Sign* to a set of functions. We use the same notation for the order and operations and their pointwise lifts. We first prove the special case of isomorphism between the sign logic $\mathscr{S}$ with a single variable and the lattice $\{x\} \to \mathit{Sign}$. Since the lattice $\{x\} \to \mathit{Sign}$ is isomorphic to *Sign*, we do not distinguish between the two. The concretization of a sign environment is defined below.

$$\gamma_{\mathit{Sign}}(\mathit{Vars} \to \mathit{Sign}) \to \mathcal{P}(\mathit{Struct}) \qquad \gamma_{\mathit{Sign}}(\varepsilon) = \big\{\sigma \mid \sigma(x) \in \gamma_{\mathit{Sign}}(\varepsilon(x))\big\}$$

An environment ε concretizes to the set of structures that respect the signs of the variables in ε.

Lemma 8 *The Lindenbaum-Tarski algebra of the sign logic $\mathscr{S}$ with a single variable has exactly five equivalence classes $\{[\mathsf{ff}_x]_{\mathscr{S}}, [x < 0]_{\mathscr{S}}, [x = 0]_{\mathscr{S}}, [x > 0]_{\mathscr{S}}, [\mathsf{tt}]_{\mathscr{S}}\}$.*

Proof We proceed by induction on formula structure and abbreviate $[\varphi]_{\mathscr{S}}$ to $[\varphi]$.
(*Base Case*) The constants ff_x and tt, and the atomic formulae $x < 0$, $x = 0$ and $x > 0$ are each in one of these equivalence classes by the introduction rule.
(*Induction Step*) The induction hypothesis is that every formula of $\mathscr{S}$ belongs to one of these equivalence classes. For the induction step, consider a formula $\varphi \wedge \psi$, where $[\varphi]$ and $[\psi]$ are among the equivalence classes above. We consider four cases for these two equivalence classes.

1. The two formulae are in the same equivalence class. By Lemma 7, we have the two sequents $\varphi \vdash_{\mathscr{S}} \psi$ and $\psi \vdash_{\mathscr{S}} \varphi$. The derivations below show that $\varphi \wedge \psi \vdash_{\mathscr{S}} \psi$ and $\psi \vdash_{\mathscr{S}} \varphi \wedge \psi$, so that $[\varphi \wedge \psi]$ is the same equivalence class as $[\varphi]$.

$$\dfrac{\dfrac{\dfrac{}{\psi \vdash_{\mathscr{S}} \psi}\ \text{I}}{\psi, \varphi \vdash_{\mathscr{S}} \psi}\ \text{WL}}{\varphi \wedge \psi \vdash_{\mathscr{S}} \psi}\ \wedge\text{L,PL} \qquad\qquad \dfrac{\psi \vdash_{\mathscr{S}} \varphi \qquad \dfrac{}{\psi \vdash_{\mathscr{S}} \psi}\ \text{I}}{\psi \vdash_{\mathscr{S}} \varphi \wedge \psi}\ \text{CL,}\wedge\text{R}$$

2. The two formulae are in distinct equivalence classes and $[\varphi]$ is $[\mathsf{ff}_x]$. In this case, $[\varphi \wedge \psi]$ is also $[\mathsf{ff}_x]$. We prove $\varphi \wedge \psi \vdash_{\mathscr{S}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{S}} \varphi \wedge \psi$ knowing that $\varphi \vdash_{\mathscr{S}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{S}} \varphi$. The case for $[\psi]$ being $[\mathsf{ff}_x]$ is identical.

$$\dfrac{\dfrac{\varphi \vdash_{\mathscr{J}} \mathsf{ff}_x}{\varphi, \psi \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\text{WL}}{\varphi \wedge \psi \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\wedge\text{L} \qquad\qquad \dfrac{}{\mathsf{ff}_x \vdash \varphi \wedge \psi}\,\mathsf{ff}\text{L}$$

3. The two equivalence classes are distinct and $[\varphi]$ is $[\mathsf{tt}]$. In this case, $[\varphi \wedge \psi]$ is $[\psi]$. It suffices to derive $\varphi \wedge \psi \vdash_{\mathscr{J}} \psi$ and $\psi \vdash_{\mathscr{J}} \varphi \wedge \psi$ given $\varphi \vdash_{\mathscr{J}} \mathsf{tt}$ and $\mathsf{tt} \vdash_{\mathscr{J}} \varphi$.

$$\dfrac{\dfrac{\dfrac{}{\psi \vdash_{\mathscr{J}} \psi}\,\text{I}}{\psi, \varphi \vdash_{\mathscr{J}} \psi}\,\text{WL}}{\varphi \wedge \psi \vdash_{\mathscr{J}} \psi}\,\wedge\text{L,PL} \qquad \dfrac{\dfrac{\dfrac{}{\vdash_{\mathscr{J}} \mathsf{tt}}\,\mathsf{tt}\text{R} \quad \mathsf{tt} \vdash_{\mathscr{J}} \varphi}{\vdash_{\mathscr{J}} \varphi}\,\text{CUT} \quad \dfrac{}{\psi \vdash_{\mathscr{J}} \psi}\,\text{I}}{\psi \vdash_{\mathscr{J}} \varphi \wedge \psi}\,\wedge\text{R}$$

The case for $[\psi]$ being $[\mathsf{tt}]$ is identical.

4. The final case is when $[\varphi]$ and $[\psi]$ are distinct and neither of them is the equivalence class of ff_x or of tt. In this case, we show that $[\varphi \wedge \psi]$ is $[\mathsf{ff}_x]$ We show that $\varphi \wedge \psi \vdash_{\mathscr{J}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{J}} \varphi \wedge \psi$ are derivable for all distinct, non-constant, atomic formulae φ and ψ. By Lemma 7, the induction hypothesis and the assumption that the equivalence classes are not those for logical constants, there are only three cases to consider.

$$\dfrac{}{x<0 \wedge x=0 \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\wedge\text{L,ff}\text{R}_1 \qquad \dfrac{}{\mathsf{ff}_x \vdash_{\mathscr{J}} x<0 \wedge x=0}\,\mathsf{ff}\text{L}$$

$$\dfrac{}{x=0 \wedge x>0 \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\wedge\text{L,ff}\text{R}_2 \qquad \dfrac{}{\mathsf{ff}_x \vdash_{\mathscr{J}} x=0 \wedge x>0}\,\mathsf{ff}\text{L}$$

$$\dfrac{}{x<0 \wedge x>0 \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\wedge\text{L,ff}\text{R}_3 \qquad \dfrac{}{\mathsf{ff}_x \vdash_{\mathscr{J}} x<0 \wedge x>0}\,\mathsf{ff}\text{L}$$

The proof so far shows that there are at most five equivalence classes. It does not show that these equivalence classes are distinct, e.g., that there is no way to derive $x>0$ from $x=0$ in $\vdash_{\mathscr{J}}$. The formulae $\{x<0, x=0, x>0\}$ do not semantically entail each other, do not entail ff_x and are not entailed by tt. By the soundness of the sign calculus, it follows that distinct formulae in *Sign* are pairwise not inter-derivable. □

Lemma 9 *The sign logic with a single variable characterizes the abstraction Sign.*

Proof Define the function $h : \mathscr{S}/\equiv_{\mathscr{J}} \to \mathit{Sign}$ as the witness for the isomorphism.

$$h([\mathsf{tt}]) \;\hat{=}\; \top$$

$$h([x<0]) \;\hat{=}\; \mathsf{Neg} \qquad h([x=0]) \;\hat{=}\; \mathsf{Zero} \qquad h([x>0]) \;\hat{=}\; \mathsf{Pos}$$

$$h([\mathsf{ff}_x]) \;\hat{=}\; \bot$$

From Lemma 8, h is a bijection. We show that h is an order isomorphism: $[\varphi] \preccurlyeq_{\mathscr{J}} [\psi]$ if and only if $h([\varphi]) \sqsubseteq h([\psi])$. We should consider different cases. In the following, for brevity, we write $\preccurlyeq$ for $\preccurlyeq_{\mathscr{J}}$.

– $[\varphi] \preccurlyeq [\mathsf{tt}] \Leftrightarrow h([\varphi]) \sqsubseteq h([\mathsf{tt}])$. The implication holds because $h([\mathsf{tt}])$ is $\top$ in the lattice. For the converse we apply the rule for true.

$$\dfrac{}{\varphi \vdash_{\mathscr{J}} \mathsf{tt}}\,\mathsf{tt}\text{R}$$

– $[\mathsf{ff}_x] \preccurlyeq [\varphi] \Leftrightarrow h([\mathsf{ff}_x]) \sqsubseteq h([\varphi])$. The implication holds because $h([\mathsf{ff}_x])$ is $\bot$. The converse holds because of the rule for false.

$$\frac{}{\mathsf{ff}_x \vdash_{\mathscr{S}} \varphi}\ \text{ff}_L$$

– For all other cases, observe that if $[\varphi]$ and $[\psi]$ are incomparable, then so are $h([\varphi])$ and $h([\psi])$. Conversely if a is not comparable to b in *Sign*, then, the inverse maps $h^{-1}(a)$ and $h^{-1}(b)$ map to elements that do not entail each other and by soundness of $\vdash_{\mathscr{S}}$, these elements cannot be in the same equivalence class.

It remains to show that h distributes over meets. In the following, for brevity, we write $\curlywedge$ for $\curlywedge_{\mathscr{S}}$. We should consider all possible cases. For example:

– $h([x<0] \curlywedge [x=0]) = h([x<0]) \sqcap h([x=0])$. We have $h([x<0] \curlywedge [x=0]) = h([x<0 \wedge x=0]) = h([\mathsf{ff}_x]) = \bot = \mathsf{Neg} \sqcap \mathsf{Zero} = h([x<0]) \sqcap h([x=0])$.

The proof is similar for all cases where the meet is bottom. Otherwise, the meet is either of the form $a \curlywedge b$ with either a and b being the same, or one of them being the top element. Verification of these cases is routine.

To complete the proof, we have to show that $\mathscr{S}$ and *Sign* have the same semantics. By Lemma 4, $\vdash_{\mathscr{S}}$ is sound, so if $[\varphi]$ and $[\psi]$ are the same equivalence class, then $mod(\varphi) = mod(\psi)$. By Lemma 8, we only have to consider one formula in each of the five equivalence classes. Consider $x > 0$.

– We have that $\gamma_{Sign}(h(x>0)) = \gamma_{Sign}(\mathsf{Pos}) = \{x\} \to \{n \mid n > 0\}$. Note that we distinguish between $\{x\} \to Sign$ and *Sign* here because we need to concretize structures. Since $mod(x>0) = \{(x{:}n) \mid n > 0\}$, the semantic condition follows.

The condition can similarly be verified for the other equivalence classes. □

The chosen isomorphism h is crucial for the semantic condition to hold. Consider the function g that maps $[x<0]$ to Pos and $[x>0]$ to Neg and otherwise agrees with h. Note that g is an isomorphism but will not satisfy the semantic condition. We show that the sign logic with multiple variables characterizes sign environments.

Lemma 10 *In the sign logic $\mathscr{S}$ over a finite set of variables Vars, every formula φ is inter-derivable with a formula of the form $\bigwedge_{x \in V} \psi(x)$, for some $V \subseteq Vars$.*

Proof The proof is by induction on the number of variables and the structure of $\mathscr{S}$-formulae. The base case is a logic with one variable, which follows from Lemma 8. The induction hypothesis is that the lemma holds for $n \geq 1$ variables. For the induction step, we need to show that the lemma holds for a formula φ with $n+1$ variables. First note that φ must be of the form $\varphi_1 \wedge \varphi_2$, where at most n variables occur in φ_1 and φ_2. This is because the only way to introduce variables is by conjunction with an atomic predicate, the only way to compose formulae is by conjunction and because formulae are finite in length. By the induction hypothesis, φ_1 and φ_2 are inter-derivable with formulae of the form $\bigwedge_{x \in V_1} \psi_1(x)$ and $\bigwedge_{x \in V_2} \psi_2(x)$, respectively. Thus, $\varphi_1 \wedge \varphi_2$ is inter-derivable with a formula of the form

$$\left(\bigwedge_{x \in V_1 \setminus V_2} \psi_1(x) \right) \wedge \left(\bigwedge_{x \in V_1 \cap V_2} \psi_1(x) \wedge \psi_2(x) \right) \wedge \left(\bigwedge_{x \in V_2 \setminus V_1} \psi_2(x) \right)$$

which is of the form in the lemma. □

Lemma 11 *The logic $\mathscr{S}$ over a finite set of variables Vars characterizes the lattice of sign environments $(Vars \to Sign, \sqsubseteq)$.*

Proof Let h be the isomorphism from $\mathscr{S}$ to *Sign* for the one variable case from Lemma 9. In the following, we write $var(\varphi)$ for the set of variables in a formula. Define the candidate isomorphism $g : \mathscr{S}/\equiv_{\mathscr{S}} \to (Vars \to Sign)$ as follows:

$$g([\varphi])(x) \mathrel{\hat{=}} \begin{cases} h([\psi(x)]) & x \in var(\varphi) \\ \top & x \notin var(\varphi) \end{cases}$$

Note that $Vars \to Sign$ is isomorphic to the product lattice $Sign^{|Vars|}$. We have to show that there are as many equivalence classes as elements in the product lattice. It follows from Lemma 10 that every equivalence class can be written as the conjunction $\bigwedge_{x \in var(\varphi)} \psi(x)$, where every formula $\psi(x)$ belongs to one equivalence class of $\mathscr{S}$ over one variable. Thus, by Lemma 8 and the rule $\wedge$R, we can combine a derivation of each individual formula to obtain a derivation of the entire conjunction. It follows that there are at least as many equivalence classes as elements in the product. To show that there are at most as many equivalence classes, we observe that every formula of the form $\bigwedge_{x \in V} \psi(x)$, for some $V \subseteq Vars$, is inter-derivable with a formula of the form $\bigwedge_{x \in V} \psi(x) \wedge \bigwedge_{x \in Vars \setminus V} \mathtt{tt}$. Thus, every formula φ derivable from a formula of the form $\bigwedge_{x \in V} \psi(x)$, for some $V \subseteq Vars$, is of the form $\bigwedge_{x \in V} \psi(x) \wedge \bigwedge_{x \in Vars \setminus V} \mathtt{tt}$. By lifting the proof of Lemma 9 component-wise, we can conclude that g is an isomorphism.

To verify the semantic condition. consider ε in $mod(\varphi)$. By Lemma 10, φ is equivalent to some $\bigwedge_{x \in var(\varphi)} \psi(x)$, so $\varepsilon(x)$ is a value satsifying $\psi(x)$. By Lemma 9, $\varepsilon(x)$ is also in $\gamma_{Sign}(h([\psi(x)]))$. It follows from the definition of g that $\varepsilon(x)$ is in $\gamma_{Sign}(g([\varphi])(x))$ for each x. From the definition of concretization for sign environments, it follows that ε is in $\gamma_{Sign}(g([\varphi]))$. □

Example 8 This example shows that the proof system of the sign calculus is incomplete and this incompleteness is fundamental to characterizing sign environments. Consider the lattice *Sign* and the sign environments $\{x, y\} \to Sign$. Three distinct sign environments and their logical representations are shown below.

$$\varepsilon_1 = \{x \mapsto \mathsf{Pos}, y \mapsto \bot\} \qquad \varepsilon_2 = \{x \mapsto \bot, y \mapsto \mathsf{Neg}\} \qquad \varepsilon_3 = \{x \mapsto \bot, y \mapsto \bot\}$$
$$h^{-1}(\varepsilon_1) = [x > 0 \wedge \mathtt{ff}_y] \qquad h^{-1}(\varepsilon_2) = [\mathtt{ff}_x \wedge y < 0] \qquad h^{-1}(\varepsilon_3) = [\mathtt{ff}_x \wedge \mathtt{ff}_y]$$

Observe that none of the formulae are satisfiable. In order for the the Lindenbaum-Tarski algebra to be isomorphic to the lattice of sign environments, we need that the formulae are not inter-derivable. That is, the proof system must be incomplete. ◁

5.3 Characterization of the Constant Proof Calculus

The lattice of integer constants $(Const, \sqsubseteq)$ is depicted in Fig. 5. It consists of the elements $Const = \mathbb{Z} \cup \{\bot, \top\}$, with $\bot$ and $\top$ as the least and greatest elements, and with all other elements being incomparable. The concretization $\gamma_{Const} : Const \to \mathcal{P}(\mathbb{Z})$ maps n to $\{n\}$. The concretization for constant environments $Vars \to Const$ is defined in a similar manner to the concretization for sign environments. We prove the characterization for the one-variable case and then generalize to more variables.

Lemma 12 *The set of equivalence classes of the Lindenbaum-Tarski algebra of the constant logic $\mathscr{C}$ with a single variable is $\{[\mathsf{ff}_x]_{\mathscr{C}}\} \cup \{[x = k]_{\mathscr{C}} \mid k \in \mathbb{Z}\} \cup \{[\mathsf{tt}]_{\mathscr{C}}\}$.*

Proof The proof is similar to the proof of Lemma 8. We reason by induction on the structure of the formulae of $\mathscr{C}$. We abbreviate $[\varphi]_{\mathscr{C}}$ to $[\varphi]$.
(*Base Case*) The constants ff_x and tt, and the atomic formulae $x = k$ are, by the introduction rule of the core calculus, in one of these equivalence classes.
(*Induction Step*) The induction hypothesis is that every formula of $\mathscr{C}$ belongs to one of these equivalence classes. For the induction step, consider a formula $\varphi \wedge \psi$. When φ and ψ are in the same equivalence class, as in Lemma 8 we have $\varphi \wedge \psi \vdash_{\mathscr{C}} \psi$ and $\psi \vdash_{\mathscr{C}} \varphi \wedge \psi$, so that $[\varphi \wedge \psi]$ is the same class as $[\varphi]$. If the two equivalence classes are distinct, we have to consider if they are comparable or incomparable. Comparisons are only possible if one of the equivalence classes is $[\mathsf{ff}_x]$ or is $[\mathsf{tt}]$. If either $[\varphi]$ or $[\psi]$ is $[\mathsf{ff}_x]$, as in Lemma 8 we have $\varphi \wedge \psi \vdash_{\mathscr{C}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{C}} \varphi \wedge \psi$, so $[\varphi \wedge \psi]$ is also $[\mathsf{ff}_x]$. If $[\varphi]$ is $[\mathsf{tt}]$, as in Lemma 8 we have $\varphi \wedge \psi \vdash_{\mathscr{C}} \psi$ and $\psi \vdash_{\mathscr{C}} \varphi \wedge \psi$, so $[\varphi \wedge \psi]$ is $[\psi]$. The case for $[\psi]$ being $[\mathsf{tt}]$ is identical. If $[\varphi]$ and $[\psi]$ are distinct and are not the equivalence classes of ff_x and tt, we show that $[\varphi \wedge \psi]$ is $[\mathsf{ff}_x]$. We show that $\varphi \wedge \psi \vdash_{\mathscr{C}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{C}} \varphi \wedge \psi$ are derivable for all distinct, non-constant, atomic formulae $x = m$ and $x = n$.

$$\dfrac{[m \neq_{\mathbb{Z}} n]}{x = m \wedge x = n \vdash_{\mathscr{C}} \mathsf{ff}_x}\ {\scriptstyle \wedge \mathrm{L}, \mathsf{ff}\mathrm{R}_4} \qquad\qquad \dfrac{}{\mathsf{ff}_x \vdash_{\mathscr{C}} x = m \wedge x = n}\ {\scriptstyle \mathsf{ff}\mathrm{L}}$$

The equivalence classes are distinct because the constant calculus is sound. □

Lemma 13 *Constant logic with one variable characterizes the lattice of constants.*

Proof Define the function $h : \mathscr{C}/{\equiv_{\mathscr{C}}} \to \mathit{Const}$ as the witness for the isomorphism.

$$h([\mathsf{tt}]) \mathrel{\hat{=}} \top \qquad h([x = k]) \mathrel{\hat{=}} k \qquad h([\mathsf{ff}_x]) \mathrel{\hat{=}} \bot$$

From Lemma 12, h is a bijection. The proof that h is an order isomorphism and distributes over meets is identical to the proof of Lemma 9 and verification of the semantic condition is straightforward. □

The following result now shows that the Lindenbaum-Tarski algebra of the constant logic $\mathscr{C}$ is isomorphic to the pointwise lift $\mathit{Vars} \to \mathscr{C}$ of $\mathscr{C}$.

Lemma 14 *Constant logic over a finite set of variables Vars characterizes the constant environments $(\mathit{Vars} \to \mathit{Const}, \sqsubseteq)$.*

Proof The proof is identical to the proof of Lemma 11 using the candidate isomorphism $g : \mathscr{C}/{\equiv_{\mathscr{C}}} \to (\mathit{Vars} \to \mathit{Const})$:

$$g([\varphi])(x) \mathrel{\hat{=}} \begin{cases} h([\psi(x)]) & x \in \mathit{var}(\varphi) \\ \top & x \notin \mathit{var}(\varphi) \end{cases}$$

where h is the isomorphism from the Lindenbaum-Tarski algebra of $\mathscr{C}$ to the lattice *Const* of constants for the one variable case from Lemma 13. The concretization for constant environments is defined in a similar manner to the concretization for sign environments, hence the verification of the semantic condition is also similar. □

5.4 Characterization of the Interval Proof Calculus

The lattice of integer intervals $(Itv, \sqsubseteq)$ is depicted in Fig. 6. It 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. The concretization is $\gamma_{Itv}([a,b]) = \{n \in \mathbb{Z} \mid a \leq n \leq b\}$ for non-$\bot$ elements. The lattice of *interval environments* is $(Vars \to Itv, \sqsubseteq)$ with the pointwise order. The concretization of an interval environment is defined similarly to concretization for sign environments: $\gamma_{Itv}(\varepsilon) = \{\sigma \mid \sigma(x) \in \gamma_{Itv}(\varepsilon(x))\}$. We characterize *Itv* by the interval logic $\mathscr{I}$ over one variable.

Lemma 15 *The set of equivalence classes of the Lindenbaum-Tarski algebra of the interval logic $\mathscr{I}$ with a single variable is $\{[x \leq k]_{\mathscr{I}} \mid k \in \mathbb{Z}\} \cup \{[x \geq k]_{\mathscr{I}} \mid k \in \mathbb{Z}\} \cup \{[x \leq n \wedge x \geq m]_{\mathscr{I}} \mid m,n \in \mathbb{Z}, m \leq n\} \cup \{[\mathsf{ff}_x]_{\mathscr{I}}, [\mathsf{tt}]_{\mathscr{I}}\}$.*

Proof The proof is by induction on formula structure. We write $[\varphi]_{\mathscr{I}}$ for $[\varphi]$.
(*Base Case*) The constants ff_x and tt, and the atomic formulae $x \geq k$ and $x \leq k$ are, by the introduction rule of the core calculus, in one of these equivalence classes.
(*Induction Step*) The induction hypothesis is that every formula of $\mathscr{I}$ belongs to one of these equivalence classes. For the induction step, consider a formula $\varphi \wedge \psi$. When φ and ψ are in the same equivalence class, as in Lemma 8 we have $\varphi \wedge \psi \vdash_{\mathscr{I}} \psi$ and $\psi \vdash_{\mathscr{I}} \varphi \wedge \psi$, so that $[\varphi \wedge \psi]$ is the same class as $[\varphi]$. If the equivalence classes are distinct, we have to consider the cases where there is either an order between the classes or they are incomparable. If either $[\varphi]$ or $[\psi]$ is $[\mathsf{ff}_x]$, as in Lemma 8 we have $\varphi \wedge \psi \vdash_{\mathscr{I}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{I}} \varphi \wedge \psi$, so $[\varphi \wedge \psi]$ is also $[\mathsf{ff}_x]$. If $[\varphi]$ is $[\mathsf{tt}]$, as in Lemma 8 we have $\varphi \wedge \psi \vdash_{\mathscr{I}} \psi$ and $\psi \vdash_{\mathscr{I}} \varphi \wedge \psi$, so $[\varphi \wedge \psi]$ is $[\psi]$. The case for $[\psi]$ being $[\mathsf{tt}]$ is identical. If $[\varphi]$ and $[\psi]$ are distinct and are not the equivalence classes of ff_x and tt, without loss of generality have the following cases:

1. If $[\varphi]$ is $[x \leq m]$ and $[\psi]$ is $[x \leq n]$, then, if $m < n$, we have that $[\varphi \wedge \psi]$ is $[\varphi]$. We show that $\varphi \wedge \psi \vdash_{\mathscr{I}} \varphi$ and $\varphi \vdash_{\mathscr{I}} \varphi \wedge \psi$ are derivable.

$$\dfrac{\dfrac{\dfrac{}{x \leq m \vdash_{\mathscr{I}} x \leq m}\,\mathrm{I}}{x \leq m, x \leq n \vdash_{\mathscr{I}} x \leq m}\,\mathrm{WL}}{x \leq m \wedge x \leq n \vdash_{\mathscr{I}} x \leq m}\,\wedge\mathrm{L} \qquad \dfrac{\dfrac{}{x \leq m \vdash_{\mathscr{I}} x \leq m}\,\mathrm{I} \qquad \dfrac{\dfrac{}{x \leq n \vdash_{\mathscr{I}} x \leq n}\,\mathrm{I}}{x \leq m \vdash_{\mathscr{I}} x \leq n}\,\text{UB-L}}{x \leq m \vdash_{\mathscr{I}} x \leq m \wedge x \leq n}\,\text{CL},\wedge\mathrm{R}$$

   Otherwise, if $n < m$, we have that $[\varphi \wedge \psi]$ is $[\psi]$. The proof is similar, requiring an application of the PL rule before the WL rule.
2. If $[\varphi]$ is $[x \geq m]$ and $[\psi]$ is $[x \geq n]$, then $[\varphi \wedge \psi]$ is $[\psi]$ if $m < n$ and is $[\varphi]$ if $n < m$. The proof is similar to the previous case but uses the LB-R rule instead of UB-L. The following derivations shows that, if $m < n$, $[\varphi \wedge \psi]$ is $[\psi]$.

$$\dfrac{\dfrac{\dfrac{}{x \geq n \vdash_{\mathscr{I}} x \geq n}\,\mathrm{I}}{x \geq n, x \geq m \vdash_{\mathscr{I}} x \geq n}\,\mathrm{WL}}{x \geq m \wedge x \geq n \vdash_{\mathscr{I}} x \geq n}\,\wedge\mathrm{L},\mathrm{PL} \qquad \dfrac{\dfrac{\dfrac{}{x \geq n \vdash_{\mathscr{I}} x \geq n}\,\mathrm{I}}{x \geq n \vdash_{\mathscr{I}} x \geq m}\,\text{LB-R} \qquad \dfrac{}{x \geq n \vdash_{\mathscr{I}} x \geq n}\,\mathrm{I}}{x \geq n \vdash_{\mathscr{I}} x \geq m \wedge x \geq n}\,\text{CL},\wedge\mathrm{R}$$

The derivations showing that, if $n < m$, $[\varphi \wedge \psi]$ is $[\varphi]$ are almost identical except that the PL rule need not be applied before WL.

3. If $[\varphi]$ is $[x \leq n]$ and $[\psi]$ is $[x \geq m]$, then, if $m \leq n$, by the introduction rule, we have that $[\varphi \wedge \psi]$ is $[x \leq n \wedge x \geq m]$. Otherwise, if $n < m$, we have that $[\varphi \wedge \psi]$ if $[\mathsf{ff}_x]$. We show that $\varphi \wedge \psi \vdash_{\mathscr{J}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{J}} \varphi \wedge \psi$ are derivable.

$$\dfrac{\dfrac{}{x \leq n, x \geq m \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\mathsf{ff}\text{R}_5}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\wedge\text{L} \qquad \dfrac{}{\mathsf{ff}_x \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\mathsf{ff}\text{L}$$

4. If $[\varphi]$ is $[x \leq k]$ and $[\psi]$ is $[x \leq n \wedge x \geq m]$, then we have three cases.
(case *A*) If $m \leq n < k$ we have that $[\varphi \wedge \psi]$ is $[\psi]$. We show that $\varphi \wedge \psi \vdash_{\mathscr{J}} \psi$ and $\psi \vdash_{\mathscr{J}} \varphi \wedge \psi$ are derivable.

$$\dfrac{\dfrac{\dfrac{}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\text{I}}{x \leq n \wedge x \geq m, x \leq k \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\text{WL}}{x \leq k \wedge (x \leq n \wedge x \geq m) \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\wedge\text{L,PL}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \leq k \vdash_{\mathscr{J}} x \leq k}\,\text{I}}{x \leq n \vdash_{\mathscr{J}} x \leq k}\,\text{UB-L} \quad \dfrac{}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\text{I}}{x \leq n, x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq k \wedge x \leq n \wedge x \geq m}\,\wedge\text{R}}{x \leq n \wedge x \geq m, x \leq n, x \geq m \vdash_{\mathscr{J}} x \leq k \wedge (x \leq n \wedge x \geq m)}\,\text{WL,PL}}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq k \wedge (x \leq n \wedge x \geq m)}\,\text{CL,}\wedge\text{L}$$

(case *B*) If $m \leq k \leq n$ we have that $[\varphi \wedge \psi]$ is $[x \leq k \wedge x \geq m]$. We show that $\varphi \wedge \psi \vdash_{\mathscr{J}} x \leq k \wedge x \geq m$ is derivable.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \leq k \vdash_{\mathscr{J}} x \leq k}\,\text{I} \quad \dfrac{}{x \geq m \vdash_{\mathscr{J}} x \geq m}\,\text{I}}{x \leq k, x \geq m \vdash_{\mathscr{J}} x \leq k \wedge x \geq m}\,\wedge\text{R}}{x \leq k, x \geq m, x \leq n \vdash_{\mathscr{J}} x \leq k \wedge x \geq m}\,\text{WL}}{x \leq k \wedge (x \leq n \wedge x \geq m) \vdash_{\mathscr{J}} x \leq k \wedge x \geq m}\,\wedge\text{L,}\wedge\text{L,PL}$$

The following derivation shows that $x \leq k \wedge x \geq m \vdash_{\mathscr{J}} \varphi \wedge \psi$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \leq k \vdash_{\mathscr{J}} x \leq k}\,\text{I} \quad \dfrac{\dfrac{\dfrac{}{x \leq n \vdash_{\mathscr{J}} x \leq n}\,\text{I}}{x \leq k \vdash_{\mathscr{J}} x \leq n}\,\text{UB-L} \quad \dfrac{}{x \geq m \vdash_{\mathscr{J}} x \geq m}\,\text{I}}{x \leq k \wedge x \geq m \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\wedge\text{L,}\wedge\text{R}}{x \leq k, x \leq k \wedge x \geq m \vdash_{\mathscr{J}} x \leq k \wedge (x \leq n \wedge x \geq m)}\,\text{WL,PL}}{x \leq k \wedge x \geq m, x \leq k, x \geq m \vdash_{\mathscr{J}} x \leq k \wedge (x \leq n \wedge x \geq m)}\,\text{CL,}\wedge\text{L}}{x \leq k \wedge x \geq m \vdash_{\mathscr{J}} x \leq k \wedge (x \leq n \wedge x \geq m)}$$

(case *C*) If $k < m \leq n$ we have that $[\varphi \wedge \psi]$ is $[\mathsf{ff}_x]$. We show that $\varphi \wedge \psi \vdash_{\mathscr{J}} \mathsf{ff}_x$ and $\mathsf{ff}_x \vdash_{\mathscr{J}} \varphi \wedge \psi$ are derivable.

$$\dfrac{\dfrac{\dfrac{}{x \leq k, x \geq m \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\mathsf{ff}\text{R}_5}{x \leq k, x \geq m, x \leq n \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\text{WL}}{x \leq k \wedge (x \leq n \wedge x \geq m) \vdash_{\mathscr{J}} \mathsf{ff}_x}\,\wedge\text{L,}\wedge\text{L,PL} \qquad \dfrac{}{\mathsf{ff}_x \vdash_{\mathscr{J}} x \leq k \wedge (x \leq n \wedge x \geq m)}\,\mathsf{ff}\text{L}$$

5. If $[\varphi]$ is $[x \geq k]$ and $[\psi]$ is $[x \leq n \wedge x \geq m]$, then, we also have three cases.
(case *A*) If $k < m \leq n$, we have that $[\varphi \wedge \psi]$ is $[\psi]$. The proof of the derivation $\varphi \wedge \psi \vdash_{\mathscr{J}} \psi$ is identical to case 4*A*. The derivation showing that $\psi \vdash_{\mathscr{J}} \varphi \wedge \psi$ requires an application of the PL rule also before the WL rule.
(case *B*) f $m \leq k \leq n$ we have that $[\varphi \wedge \psi]$ is $[x \leq n \wedge x \geq k]$. The derivation showing that $\varphi \wedge \psi \vdash_{\mathscr{J}} x \leq n \wedge x \geq k$ is almost identical to case 4*B* except for requiring an application of the PL rule after the WL rule. The derivation showing that $x \leq n \wedge x \geq k \vdash_{\mathscr{J}} \varphi \wedge \psi$ requires an an application of the PL rule also before the WL rule and an application of the LB-L instead of the UB-L rule.
(case *C*) If $m \leq n < k$ we have that $[\varphi \wedge \psi]$ is $[\mathsf{ff}_x]$. The derivations are identical to case 4*C* except for requiring an application of the PL rule after the WL rule.
6. If $[\varphi]$ is $[x \leq q \wedge x \geq p]$ and $[\psi]$ is $[x \leq n \wedge x \geq m]$, then we have five cases.
(case *A*) If $p \leq m \leq n \leq q$, we have that $[\varphi \wedge \psi]$ is $[\psi]$. The derivation $\varphi \wedge \psi \vdash_{\mathscr{J}} \psi$ is identical to case 4*A*. The proof that $\psi \vdash_{\mathscr{J}} \varphi \wedge \psi$ is derivable is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq n}\,\mathrm{I}}{x \leq n \vdash_{\mathscr{J}} x \leq q}\,\text{UB-L} \qquad \dfrac{\dfrac{}{x \geq p \vdash_{\mathscr{J}} x \geq p}\,\mathrm{I}}{x \geq m \vdash_{\mathscr{J}} x \geq p}\,\text{LB-L}}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq q \wedge x \geq p}\,\wedge\mathrm{L},\wedge\mathrm{R}}{*}$$

$$\dfrac{* \qquad \dfrac{}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\mathrm{I}}{x \leq n \wedge x \geq m \vdash_{\mathscr{J}} (x \leq q \wedge x \geq p) \wedge (x \leq n \wedge x \geq m)}\,\mathrm{CL},\wedge\mathrm{R}$$

(case *B*) If $m \leq p \leq q \leq n$, we have that $[\varphi \wedge \psi]$ is $[\varphi]$. The derivation $\varphi \wedge \psi \vdash_{\mathscr{J}} \varphi$ only requires an application of $\wedge$L, WL, and the introduction rule. The derivation $\varphi \vdash_{\mathscr{J}} \varphi \wedge \psi$ is identical to case 6*A*.
(case *C*) If $p \leq m \leq q \leq n$, we have that $[\varphi \wedge \psi]$ is $[x \leq q \wedge x \geq m]$. The derivation showing that $\varphi \wedge \psi \vdash_{\mathscr{J}} x \leq q \wedge x \geq m$ is almost identical to case 5*B* except for requiring an application of $\wedge$L, WL, and PL before $\wedge$R. The derivation showing that $x \leq q \wedge x \geq m \vdash_{\mathscr{J}} \varphi \wedge \psi$ is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \leq n \vdash_{\mathscr{J}} x \leq n}\,\mathrm{I}}{x \leq q \vdash_{\mathscr{J}} x \leq n}\,\text{UB-L} \qquad \dfrac{}{x \geq m \vdash_{\mathscr{J}} x \geq m}\,\mathrm{I}}{x \leq q \wedge x \geq m \vdash_{\mathscr{J}} x \leq n \wedge x \geq m}\,\wedge\mathrm{L},\wedge\mathrm{R}}{*}}{x \leq q \wedge x \geq m \vdash_{\mathscr{J}} (x \leq q \wedge x \geq p) \wedge (x \leq n \wedge x \geq m)}\,\mathrm{CL},\wedge\mathrm{R}$$

$$\dfrac{\dfrac{\dfrac{}{x \leq q \vdash_{\mathscr{J}} x \leq q}\,\mathrm{I} \qquad \dfrac{\dfrac{}{x \geq p \vdash_{\mathscr{J}} x \geq p}\,\mathrm{I}}{x \geq m \vdash_{\mathscr{J}} x \geq p}\,\text{LB-L}}{x \leq q \wedge x \geq m \vdash_{\mathscr{J}} x \leq q \wedge x \geq p}\,\wedge\mathrm{L},\wedge\mathrm{R}}{*}$$

(case *D*) If $m \leq p \leq n \leq q$, we have that $[\varphi \wedge \psi]$ is $[x \leq n \wedge x \geq p]$. The derivation showing that $\varphi \wedge \psi \vdash_{\mathscr{J}} x \leq q \wedge x \geq m$ is almost identical to case 6*C* except for requiring the application of PL after (and not before) the first application of the WL rule and the last application of $\wedge$L. The derivation $x \leq n \wedge x \geq p \vdash_{\mathscr{J}} \varphi \wedge \psi$ is identical to case 6*C*.
(case *E*) If $q < m$ or $n < p$ we have that $[\varphi \wedge \psi]$ is $[\mathsf{ff}_x]$. If $q < m$, the proof is similar to case 4*C* except for requiring the applications of PL, $\wedge$L, WL, and again

PL before applying $\mathsf{ff}_{\mathrm{R}_5}$. If $n < p$, the proof is similar case $5C$ except for requiring the applications of $\wedge$L, PL and WL before applying $\mathsf{ff}_{\mathrm{R}_5}$.

The equivalence classes are distinct because the interval calculus is sound. □

Lemma 16 *The interval logic over one variable characterizes Itv.*

Proof Define the function $h : \mathscr{I}/\equiv_{\mathscr{I}} \to \mathit{Itv}$ as the witness for the isomorphism.

$$h([\mathsf{tt}]) \mathrel{\hat{=}} \top$$
$$h([x \leq k]) \mathrel{\hat{=}} [-\infty, k] \qquad h([x \leq n \wedge x \geq m]) \mathrel{\hat{=}} [m, n] \qquad h([x \geq k]) \mathrel{\hat{=}} [k, +\infty]$$
$$h([\mathsf{ff}_x]) \mathrel{\hat{=}} \bot$$

From Lemma 15, h is a bijection. We show that h is an order isomorphism: $[\varphi] \preccurlyeq_{\mathscr{I}} [\psi]$ if and only if $h([\varphi]) \sqsubseteq h([\psi])$. We have different other cases to consider. In the following, for brevity, we write $\preccurlyeq$ for $\preccurlyeq_{\mathscr{I}}$.

- $[\varphi] \preccurlyeq [\mathsf{tt}] \Leftrightarrow h([\varphi]) \sqsubseteq h([\mathsf{tt}])$. The implication holds because $h([\mathsf{tt}]) \mathrel{\hat{=}} \top$. For the converse we apply the rule for true as in Lemma 9.
- $[\mathsf{ff}_x] \preccurlyeq [\varphi] \Leftrightarrow h([\mathsf{ff}_x]) \sqsubseteq h([\varphi])$. The implication holds because $h([\mathsf{ff}_x]) \mathrel{\hat{=}} \bot$. The converse holds because of the rule for false as in Lemma 9.
- $[x \leq m] \preccurlyeq [x \leq n] \Leftrightarrow h([x \leq m]) \sqsubseteq h([x \leq n])$, for any given $m, n \in \mathbb{Z}$ such that $m \leq n$. The implication holds because $h([x \leq m]) \mathrel{\hat{=}} [-\infty, m]$ and $h([x \leq n]) \mathrel{\hat{=}} [-\infty, n]$. The converse holds because of the UB-L rule and the introduction rule.

$$[m \leq n]\ \frac{\dfrac{}{x \leq n \vdash_{\mathscr{I}} x \leq n}\,\mathrm{I}}{x \leq m \vdash_{\mathscr{I}} x \leq n}\,\text{UB-L}$$

- $[x \geq n] \preccurlyeq [x \geq m] \Leftrightarrow h([x \geq n]) \sqsubseteq h([x \geq m])$, for any given $m, n \in \mathbb{Z}$ such that $m \leq n$. The implication holds because $h([x \geq n]) \mathrel{\hat{=}} [n, +\infty]$ and $h([x \geq m]) \mathrel{\hat{=}} [m, +\infty]$. The converse holds because of the LB-L rule and the introduction rule.
- $[x \leq n \wedge x \geq m] \preccurlyeq [x \leq k] \Leftrightarrow h([x \leq n \wedge x \geq m]) \sqsubseteq h([x \leq k])$, for any given $k, m, n \in \mathbb{Z}$ such that $m \leq n \leq k$. The implication holds because $h([x \leq n \wedge x \geq m]) \mathrel{\hat{=}} [m, n]$ and $h([x \leq k]) \mathrel{\hat{=}} [-\infty, k]$. The converse is show below.

$$[n \leq k]\ \frac{\dfrac{\dfrac{}{x \leq k \vdash_{\mathscr{I}} x \leq k}\,\mathrm{I}}{x \leq n \vdash_{\mathscr{I}} x \leq k}\,\text{UB-L}}{x \leq n \wedge x \geq m \vdash_{\mathscr{I}} x \leq k}\,\wedge\text{L,WL}$$

- $[x \leq n \wedge x \geq m] \preccurlyeq [x \geq k] \Leftrightarrow h([x \leq n \wedge x \geq m]) \sqsubseteq h([x \geq k])$, for any given $k, m, n \in \mathbb{Z}$ such that $k \leq m \leq n$. The implication holds because $h([x \leq n \wedge x \geq m]) \mathrel{\hat{=}} [m, n]$ and $h([x \geq k]) \mathrel{\hat{=}} [k, +\infty]$. The converse is similar to the previous case except for also requiring PL before WL and for requiring the LB-L rule instead of UB-L.
- $[x \leq n \wedge x \geq m] \preccurlyeq [x \leq q \wedge x \geq p] \Leftrightarrow h([x \leq n \wedge x \geq m]) \sqsubseteq h([x \leq q \wedge x \geq p])$, for any given $m, n, p, q \in \mathbb{Z}$ such that $p \leq m \leq n \leq q$. The implication holds because $h([x \leq n \wedge x \geq m]) \mathrel{\hat{=}} [m, n]$ and $h([x \leq q \wedge x \geq p]) \mathrel{\hat{=}} [p, q]$. The converse is below.

$$\dfrac{\dfrac{[n \leq q]\ \dfrac{\dfrac{}{x \leq q \vdash_{\mathscr{I}} x \leq q}\,\mathrm{I}}{x \leq n \vdash_{\mathscr{I}} x \leq q}\,\text{UB-L}}{x \leq n \wedge x \geq m \vdash_{\mathscr{I}} x \leq q}\,\wedge\text{L,WL} \qquad \dfrac{[p \leq m]\ \dfrac{\dfrac{}{x \geq p \vdash_{\mathscr{I}} x \geq p}\,\mathrm{I}}{x \geq m \vdash_{\mathscr{I}} x \geq p}\,\text{LB-L}}{x \leq n \wedge x \geq m \vdash_{\mathscr{I}} x \geq p}\,\wedge\text{L,PL,WL}}{x \leq n \wedge x \geq m \vdash_{\mathscr{I}} x \leq q \wedge x \geq p}\,\text{CL,}\wedge\text{R}$$

– For all other cases, if $[\varphi]$ and $[\psi]$ are incomparable, then so are $h([\varphi])$ and $h([\psi])$. Conversely if a is not comparable to b in *Sign*, then, the inverse maps $h^{-1}(a)$ and $h^{-1}(b)$ map to elements that do not entail each other and by soundness of the interval calculus, these elements cannot be in the same equivalence class.

It remains to show that h distributes over meets. In the following, for brevity, we write $\curlywedge$ for $\curlywedge_{\mathscr{I}}$. We should consider all possible cases. For example:

– $h([x \leq n \wedge x \geq m] \curlywedge [x \leq q \wedge x \geq p]) = h([x \leq n \wedge x \geq m]) \sqcap h([x \leq q \wedge x \geq p])$ where $p \leq m \leq q \leq n$. We have $h([x \leq n \wedge x \geq m] \curlywedge [x \leq q \wedge x \geq p]) = h([(x \leq n \wedge x \geq m) \wedge (x \leq q \wedge x \geq p)]) = h([x \leq q \wedge x \geq m]) = [m,q] = [m,n] \sqcap [p,q] = h([x \leq n \wedge x \geq m]) \sqcap h([x \leq q \wedge x \geq p])$.

The reasoning is analogous for all other cases.

The argument for the semantic condition is similar to that for sign logic. □

It is worth noting the difference in the proofs of the condition and the semantic condition in Lemmas 9, 13 and 16. Reasoning about Lindenbaum-Tarski algebras is specific to and proportional in complexity to the axioms of the theory. The concretization functions for all abstractions are defined similarly, so verifying the semantic condition requires the same reasoning in each case.

Lemma 17 *The logic $\mathscr{I}$ over a finite set of variables Vars characterizes the interval environments $(Vars \to Itv, \sqsubseteq)$.*

Proof The proof is identical to the proof of Lemma 11 using the candidate isomorphism $g : \mathscr{I}/\equiv_{\mathscr{I}} \to (Vars \to Itv)$:

$$g([\varphi])(x) \triangleq \begin{cases} h([\psi(x)]) & x \in var(\varphi) \\ \top & x \notin var(\varphi) \end{cases}$$

where h is the isomorphism from the Lindenbaum-Tarski algebra of $\mathscr{I}$ to the lattice *Itv* of intervals for the one variable case from Lemma 16. Verifying the semantic condition is as in Lemma 11. □

6 Related Work and Discussion

We discuss our work from a conceptual perspective and from the viewpoint of current theoretical and practical research.

6.1 Related Work

Automata, Logic and Languages. A classic family of results shows that regular expressions, finite automata over finite words, and WS1S all define regular languages. We refer to the survey by Thomas [1997] for equally classic extensions of those results to infinite words and trees. Our work has applied this perspective to programs by using Büchi's construction to define the set of executions in a control-flow graph by

a formula in WS1S(T). The standard, algorithmic approach to reasoning about regular expressions and WS1S is to compile them to automata. We have shown that abstract interpreters can be viewed as solvers for WS1S(T) formulae, which use a graph structure to represent second-order constraints, but use deductive techniques to reason about first-order constraints. Our encoding differs from the use of set constraints by Aiken [1999] and second-order Horn clauses by Grebenshchikov et al [2012] in that models of our formulae are erroneous executions, not invariants.

Algebraic Logic and Stone Duality. The framework of Stone duality relate categories of lattices with operators, posets with relations, and topological spaces [Johnstone, 1986]. Since the Lindenbaum-Tarski construction generates lattices with operators, Stone duality can be viewed as a way to move between different representations of a theory. Abramsky [1987] extended Stone duality to lambda calculi by characterizing domains in semantics. We believe that logical characterization of strictness analysis by Jensen [1991] was the first application of Stone duality to abstract interpretation. Both Abramsky and Jensen characterized the structures they studied as Lindenbaum-Tarski algebras of propositional, modal, intuitionistic logics. The lattices we studied are non-distributive and arise as algebras of first-order theories. Moreover, the lattices we studied are complete, so the topological machinery of Stone duality is not required to obtain a representation of the theories involved.

The approach we used is influenced by Schmidt [2008], who first articulated the ideas in Table 1 that the partial order in a lattice can be viewed as the proof theory of a logic, and that the concretization function defines the model-theoretic semantics of the logic. Schmidt formalized this view and studied the relationship between soundness and completeness in a logic and the corresponding notions in abstract interpretation. In our work, we have identified proof calculi for specific lattices and have identified a new connection between the Lindenbaum-Tarski construction and abstract interpretation.

Abstract Interpretation and Satisfiability. A driving force behind much current research is the discovery of novel combinations of automated deduction and abstract interpretation. The dissertations of Haller [2014] and Thakur [2014] and Dagstuhl seminar notes of Kroening et al [2014] provide a summary of this research. This paper contributes to this programme by providing a logical characterizations of an instance of abstract interpretation.

The general theme of work lifting the internals of SAT and SMT solvers to abstract domains has been to provide a property-guided, on-demand refinement of abstract interpretation-based analysis. DPLL(T) and CDCL have been lifted to implement property-guided, path-sensitive analyses [D'Silva et al, 2013; Harris et al, 2010]. Stålmarck's method has been used to refine abstract transformers [Thakur and Reps, 2012a], interpolants have been used to refine widening operators [Gulavani et al, 2008] and unification has been used to obtain complete reasoning about restricted families of programs [Tiwari and Gulwani, 2007]. The Nelson-Oppen procedure, though less general than reduced product [Cousot and Cousot, 1979; Cousot et al, 2013], works as an algorithmic domain combinator [Gulwani and Tiwari, 2006].

Conversely, abstract interpretation has been incorporated in SMT and constraint solvers to improve theory propagation [Truchet et al, 2010; Pelleau et al, 2011], to use joins for space-efficient representation [Bjørner et al, 2008], and to use widening for generalization [Leino and Logozzo, 2007]. Algorithms based on abstract interpretation have been used to implement alternatives to DPLL(T) [Brain et al, 2014; Thakur and Reps, 2012b] and to improve SAT encodings [Brain et al, 2016].

Theoretical work combining automated deduction and abstract interpretation has attempted to give abstract interpretation formulations of SMT solvers [Cousot et al, 2013; D'Silva et al, 2014; Thakur and Reps, 2012a]. While it is natural to model the algebraic content of a solver in an order-theoretic way, it is cumbersome to model combinatorial aspects such as decision heuristics and precise details of conflict analysis. We also believe that the work discussed above uses a mathematical framework that is atypical for formalizing solver algorithms. For these reasons, we have attempted a logical description of abstract interpretation.

6.2 Applications and Extensions

We briefly discuss potential applications of this work. The practical motivation for this work was to lift the clause learning capabilities of SAT and SMT solvers to abstract interpreters. We have used the theoretical framework of § 2 and § 3 to develop an abstract interpretation for termination analysis that combines propagation and learning [D'Silva and Urban, 2015b].

Another application is to determine the correctness of an analyzer, which is crucial since abstract interpreters are often used to reason about safety-critical software. We refer to [Cachera and Pichardie, 2009] for a survey of work on constructing analyzers with proof assistants and [Jourdan et al, 2015] for a recent highlight of this research programme. Developing an analyzer in a proof assistant incurs performance penalties, requires significant development time and does not provide confidence in the correctness of existing, deployed analyzers. An alternative, inspired by proof-producing SAT and SMT solvers, is to have the analyzer generate a proof certificate. Since most abstract interpreters have a modular architecture and rely on an abstract domain library, it would be sufficient to equip such a library with the ability to generate proof certificates. Our work is a step towards this goal; we have shown how to obtain proofs for reasoning performed within a lattice. In order to apply to analyzers used in practice, our proof-theoretic characterization has to extend to transformers and we need such characterizations for analyzers used in practice.

A third motivation is to bring tools and techniques from logic and automated deduction to abstract interpreters. Given the logical characterizations in this paper, one can study cut elimination, proof size, interpolation, and other properties of an abstract interpreter. Without this work, it would not even be clear that these notions apply to an abstract domain.

7 Conclusion

This work advances the logical understanding of the internals of abstract interpreters. Our results make precise widespread folk intuition that abstract domains correspond to monadic logics that are closed under conjunction. In undertaking an explicit study, we have also highlighted the non-standard treatment of false, the limited structure of the sequents involved and the absence of terms. Though our results are unsurprising, we believe such a study contributes in novel ways to the broader research programme of studying logic and abstract interpretation.

The next steps are a characterization of transformers as modalities. We believe this step will bring new challenges and insights as first-order modal theories have received little attention in the literature. We are also not aware of a logical account of widening and narrowing operators and believe that an advance there would need to connect with approaches for inductive generalization.

In summary, we believe that our work leads to several theoretical and practical questions that can be studied from the viewpoint of automated deduction or abstract interpretation. We have begun these investigations and hope that this exposition enables the automated deduction community to participate in the same.

References

Abramsky S (1987) Domain theory and the logic of observable properties. PhD thesis, University of London

Aiken A (1999) Introduction to set constraint-based program analysis. Science of Computer Programming 35:79–111

Bjørner N, de Moura L (2014) Applications of SMT solvers to program verification. In: Notes for the Summer School on Formal Techniques

Bjørner N, Duterte B, de Moura L (2008) Accelerating lemma learning using joins – DPLL(⊔). In: Proc. of Logic for Programming, Artificial Intelligence and Reasoning

Brain M, D'silva V, Griggio A, Haller L, Kroening D (2014) Deciding floating-point logic with abstract conflict driven clause learning. Formal Methods in Systems Design 45(2):213–245

Brain M, Hadarean L, Kroening D, Martins R (2016) Automatic generation of propagation complete SAT encodings. In: Proc. of Verification, Model Checking and Abstract Interpretation, Springer, pp 536–556

Büchi JR (1960) On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science, Stanford Univ. Press, pp 1–11

Cachera D, Pichardie D (2009) Comparing techniques for certified static analysis. In: The NASA Formal Methods Symposium (NFM), NASA Ames Research Center, pp 111–115

Cousot P, Cousot R (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of Principles of Programming Languages, ACM Press, pp 238–252

Cousot P, Cousot R (1979) Systematic design of program analysis frameworks. In: Proc. of Principles of Programming Languages, ACM Press, pp 269–282

Cousot P, Cousot R, Mauborgne L (2013) Theories, solvers and static analysis by abstract interpretation. J ACM 59(6):31:1–31:56

Dalla Preda M, Giacobazzi R, Lakhotia A, Mastroeni I (2015) Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables. In: Proc. of Principles of Programming Languages, ACM Press, pp 329–341

D'antoni L, Veanes M (2015) Extended symbolic finite automata and transducers. Formal Methods in Systems Design 47(1)

D'Silva V, Urban C (2015a) Abstract interpretation as automated deduction. In: Proc. of Automated Deduction, pp 450–464

D'Silva V, Urban C (2015b) Conflict-driven conditional termination. In: Proc. of Computer Aided Verification, pp 471–286

D'Silva V, Haller L, Kroening D (2013) Abstract conflict driven learning. In: Proc. of Principles of Programming Languages, ACM Press, pp 143–154

D'Silva V, Haller L, Kroening D (2014) Abstract satisfaction. In: Proc. of Principles of Programming Languages, ACM Press, pp 139–150

van den Elsen S (2012) Weak monadic second-order theory of one successor. Seminar: Decision Procedures, http://www.mpi-sws.org/ piskac/teaching/decpro-ws12/slides/WS1S.pdf

Grebenshchikov S, Lopes NP, Popeea C, Rybalchenko A (2012) Synthesizing software verifiers from proof rules. In: Proc. of Programming Language Design and Implementation, ACM Press, pp 405–416

Gulavani BS, Chakraborty S, Nori AV, Rajamani SK (2008) Automatically refining abstract interpretations. In: Proc. of Tools and Algorithms for the Construction and Analysis of Systems, Springer, LNCS, vol 4963, pp 443–458

Gulwani S, Tiwari A (2006) Combining abstract interpreters. In: Proc. of Programming Language Design and Implementation, ACM Press, pp 376–386

Haller LCR (2014) Abstract satisfaction. PhD thesis, University of Oxford

Harris WR, Sankaranarayanan S, Ivančić F, Gupta A (2010) Program analysis via satisfiability modulo path programs. In: Proc. of Principles of Programming Languages, pp 71–82

Heizmann M, Hoenicke J, Podelski A (2013) Software model checking for people who love automata. In: Proc. of Computer Aided Verification, Springer, pp 36–52

Jensen TP (1991) Strictness analysis in logical form. In: FPCA, Springer, pp 352–366

Johnstone P (1986) Stone Spaces. Cambridge Studies in Advanced Mathematics, Cambridge University Press

Jourdan JH, Laporte V, Blazy S, Leroy X, Pichardie D (2015) A formally-verified c static analyzer. In: Proc. of Principles of Programming Languages, ACM Press, pp 247–259

Kroening D, Reps TW, Seshia SA, Thakur AV (2014) Decision procedures and abstract interpretation (Dagstuhl seminar 14351). Dagstuhl Reports 4(8):89–106

Leino KRM, Logozzo F (2007) Using widenings to infer loop invariants inside an SMT solver, or: A theorem prover as abstract domain. In: Workshop on Invariant Generation, RISC Report 07-07, pp 70–84

Nieuwenhuis R, Oliveras A, Tinelli C (2006) Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J ACM 53:937–977

Pelleau M, Truchet C, Benhamou F (2011) Octagonal domains for continuous constraints. In: CP, pp 706–720

Rasiowa H, Sikorski R (1963) The mathematics of metamathematics. Polish Academy of Science, Warsaw

Schmidt DA (2008) Internal and external logics of abstract interpretations. In: Proc. of Verification, Model Checking and Abstract Interpretation, Springer-Verlag, Berlin, Heidelberg, pp 263–278

Surma SJ (1982) On the origin and subsequent applications of the concept of the lindenbaum algebra. In: L Jonathan Cohen HP Jerzy Loś, Podewski KP (eds) Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Studies in Logic and the Foundations of Mathematics, vol 104, Elsevier, pp 719–734

Thakur AV (2014) Symbolic abstraction: Algorithms and applications. PhD thesis, The University of Wisconsin - Madison

Thakur AV, Reps T (2012a) A generalization of Stålmarck's method. In: Proc. of Static Analysis Symposium, Springer

Thakur AV, Reps TW (2012b) A method for symbolic computation of abstract operations. In: Proc. of Computer Aided Verification

Thomas W (1997) Languages, automata, and logic. In: Rozenberg G, Salomaa A (eds) Handbook of Formal Languages, Vol. 3, Springer, pp 389–455

Tiwari A, Gulwani S (2007) Logical interpretation: Static program analysis using theorem proving. In: Proc. of Automated Deduction, pp 147–166

Truchet C, Pelleau M, Benhamou F (2010) Abstract domains for constraint programming, with the example of octagons. In: Symbolic and Numeric Algorithms for Scientific Computing, pp 72–79

Vardi MY, Wilke T (2008) Automata: from logics to algorithms. In: Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]., pp 629–736

Vardi MY, Wolper P (1994) Reasoning about infinite computations. Information and Computation 115(1):1–37