Internal and External Logics of Abstract Interpretations

David A. Schmidt[⋆]

Kansas State University, Manhattan, Kansas, USA

Abstract. We show that every abstract interpretation possesses an *internal logic*, whose proof theory is defined by the partial ordering on the abstract domain's elements and whose model theory is defined by the domain's concretization function. We explain how program validation and transformation depend on this logic.
Next, when a logic *external* to the abstract interpretation is imposed, we show how to synthesize a sound, underapproximating, set-based variant of the external logic and give conditions when the underapproximating logic can be embedded within the original abstract domain, *inverted*. We show how model-checking logics depend on this construction.
The intent of this paper is tutorial, to integrate little-publicized results into a standard framework that can be used by practitioners of static analysis.

Perhaps the central issue in program validation and transformation is how to apply the results of a static program analysis to prove that a desired validation/transformation property holds true: How does the domain of logical properties "connect" to the domain of values used in the static analysis?

Here are three examples: *(i)* we use data-flow analysis to compute sets of available expressions and use the sets to decide properties of register allocation [20]; *(ii)* we complete a state-space exploration and use it to model check a temporal-logic formula that defines a safety property [4] or program-transformation criterion [15]; *(iii)* we apply predicate abstraction with counter-example-guided refinement (CEGAR) to generate an assertion set that proves a safety property [1, 2, 19, 28].

This paper asserts that the value domain used by a static analysis and the logic used for validation and transformation should be *one and the same* — the logic should be *internal* to the value domain. If the values and logical properties differ, then the logic must be defined *externally*, and this paper shows how.

Let Σ be the states/stores generated by a program; let A be an abstract domain for static analysis (e.g., sign values or sets of available expressions or names of state partitions); and let $\gamma : A \to \mathcal{P}(\Sigma)$ be the *concretization function* that maps each $a \in A$ to the values it models in Σ.

In this paper, we demonstate that

[⋆] `schmidt@cis.ksu.edu`. Supported by NSF ITR-0326577.

- γ defines a logic *internal* to A for Σ. The elements of A act *both* as computational values and as logical assertions. (Data-flow analysis, predicate abstraction, and CEGAR-based model checking exploit this coincidence.)
- The internal logic's model theory, $\models$, is defined by γ; its proof theory, $\vdash$, is defined by A's partial ordering, $\sqsubseteq$. (This is the beauty of partial-order-based static analysis — a computable $\sqsubseteq$ defines the deduction system.)
- The notion of (forwards) completeness from abstract interpretation theory [17, 25, 29] *characterizes* A's internal logic — it tells us from the start what we can express and prove.
- When a logic for Σ is proposed independently from A and γ, then an *external logic* must be fashioned, using a powerset completion. *But,* when γ preserves *both* meets and joins, the external logic can be embedded *within* A by *inverting* A's partial ordering!

We conclude, in the last case, that every abstract domain A with such a γ has *two* interpretations: an overapproximating, *computational* interpretation, used to compute the results of a static analysis, and an underapproximating, *logical* interpretation, used to prove logical properties of the results. In this formal sense, we "overapproximate the model and underapproximate the logic."

These developments are implicit in virtually all applications of static analysis to program validation and transformation, and this paper aims to present the principles as directly as possible.

1 Abstract interpretation

A program is a discrete dynamic system [6]: there is a domain of possible program states, Σ, and one or more *transition functions*, $f : \Sigma \to \Sigma$, that are repeatedly applied to an initial state selected from Σ.

For a program written in declarative notation the state might be the program's source text, and the transition function is a rewriting engine. For "flowchart programs," there is a control-flow graph, whose nodes are program points, and a transition function is attached to each arc in the graph; the transition function updates the state that traverses the arc. (See Figure 1 for such an example.) Or, there is a single, global transition function, written as a "case" command, that updates the state that is depicted as a program-point, storage pair.

A static analysis is a finitely computed estimate of the states generated by the transition functions. The estimate is typically phrased as a superset of the actual, concrete states that are reached. Based on the estimate, a program validation or transformation might be undertaken.

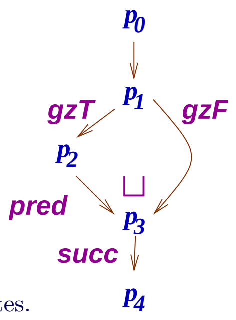
To compute finitely these state-set estimates, we define a set, A, of abstract representations of those subsets of Σ of interest.

To relate A to Σ, we define a *concretization function*, $\gamma : A \to \mathcal{P}(\Sigma)$, such that for $S \subseteq \Sigma$ and $a \in A$, S *is approximated by* a if $S \subseteq \gamma(a)$.

For a variety of implementational reasons [7, 20, 22], we partially order the abstract values as $(A, \sqsubseteq)$ such that *(i)* $\sqsubseteq$ is finitely computable and *(ii)* γ is monotone. (For some applications, a discrete ordering on A works fine.)

Example program and its flowgraph annotated by transition functions:

```
$p_0$ : readInt(x)
$p_1$ : if x>0 :
$p_2$ :   x:= pred(x)
$p_3$ : x:= succ(x)
$p_4$ : writeInt(x)
```



Let $\Sigma = Int$ be the domain of program states. (Here, the state remembers x's value.)

The transition functions (*standard semantics*) for `pred`, `succ`, and `>0` have arity, $\Sigma \to \Sigma_\perp$:

$$
\begin{array}{lll}
pred(n) = n - 1 & gzT(n) = n, \text{ if } n > 0 & gzF(n) = n, \text{ if } n \leq 0 \\
succ(n) = n + 1 & gzT(n) = \perp, \text{ if } n \leq 0 & gzF(n) = \perp, \text{ if } n > 0
\end{array}
$$

(Note: gzT and gzF act as "filter functions"; all transition functions are $\perp$-strict — $\perp$ is *not* propagated in a concrete execution trace.)

The *collecting semantics* transition functions have arity, $\mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$; let $S \subseteq \Sigma$:

$$
\begin{array}{ll}
pred(S) = \{n - 1 \mid n \in S\} & gzT(S) = \{n \in S \mid n > 0\} \\
succ(S) = \{n + 1 \mid n \in S\} & gzF(S) = \{n \in S \mid n \leq 0\}
\end{array}
$$

Fig. 1. Sample program and its transition functions

Figure 1 introduces an example program that uses transition functions *succ* and *pred* to manipulate integer input. Perhaps we wish to estimate the output sets from this program for the cases when the input sets are all the negatives, or all the positives, or just the integer, 0 — the information might enable useful validations or transformations.

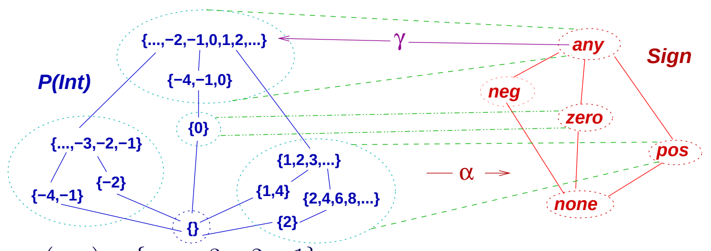
To do this, we define an abstract domain, *Sign*, with representatives for the previously mentioned data sets, along with representatives for the empty set and *Int*, partially ordered (so that $\sqcup$ is defined). γ maps the representations to the sets they represent. See Figure 2.

We might ask if γ has an inverse, which maps a state set, $S \subseteq \Sigma$, to the A-value that most precisely approximates S. If so, a *Galois connection* results:

Definition 1. *For partially ordered sets,* $(\mathcal{P}(\Sigma), \subseteq)$ *and* $(A, \sqsubseteq)$*, a pair of monotone functions,* $\alpha : \mathcal{P}(\Sigma) \to A$ *and* $\gamma : A \to \mathcal{P}(\Sigma)$*, form a* Galois connection *iff* (i) *for all* $S \in \mathcal{P}(\Sigma)$, $S \subseteq \gamma(\alpha(S))$*, and* (ii) *for all* $a \in A$, $\alpha(\gamma(a)) \sqsubseteq a$.

Equivalently, there is a Galois connection when $S \subseteq \gamma(a)$ iff $\alpha(S) \sqsubseteq a$*, for all* $S \in \mathcal{P}(\Sigma)$ *and* $a \in A$.

γ is called the *upper adjoint* and α is the *lower adjoint* of the Galois connection.



$\gamma(neg) = \{\cdots, -3, -2, -1\}$
$\gamma(zero) = \{0\}$
$\gamma(pos) = \{1, 2, 3, \cdots\}$
$\gamma(any) = Int$
$\gamma(none) = \emptyset$

$\alpha\{2, 4, 6, 8, ...\} = pos$
$\alpha\{-4, -1, 0\} = any$
$\alpha\{0\} = zero$
$\alpha\{\} = none$, etc.

α and γ form a *Galois connection*: γ interprets the properties, and $\alpha(S) = \sqcap\{a \mid S \subseteq \gamma(a)\}$ maps state set, S, to the property that best describes it [7].

Fig. 2. Abstract domain of signed values, placed within a Galois connection

Galois connections enjoy many properties [12, 16]; here are a key few. First, α and γ are inverses on each other's ranges: for all $S \in \gamma[A]$, $\gamma(\alpha(S)) = S$, and for all $a \in \alpha[\Sigma]$, $\alpha(\gamma(a)) = a$. Next, every upper (lower) adjoint has a unique lower (upper) adjoint mate, and when we say that γ "is an upper adjoint," we imply the existence of the uniquely defined α. Third, γ preserves meets: for all $T \subseteq A$, $\gamma(\sqcap T) = \cap_{a \in T}\gamma(a)$ (similar for α and joins). Conversely, when A is a complete lattice and function γ preserves meets, then γ is an upper adjoint. In Figure 2, γ is an upper adjoint.

1.1 Abstracting state transition functions

Now that concrete state sets are approximated by elements of A, we can approximate the transition functions. For generality, say that each transition function has arity, $f : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$, so that f can express nondeterminism as well as pre- and post-images of state sets ("collecting semantics" [21] — cf. Figure 1).

For each f, the corresponding *abstract transition function*, $f^\sharp : A \to A$, must soundly overestimate f's post-images:

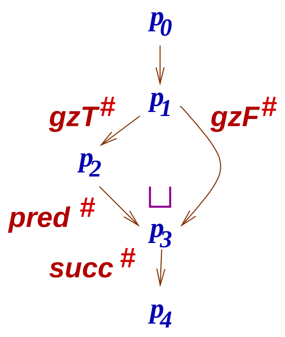
Definition 2. $f^\sharp : A \to A$ *is* sound for $f : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$*, if for all* $a \in A$*,* $f(\gamma(a)) \subseteq \gamma(f^\sharp(a))$*, that is,* $f \circ \gamma \sqsubseteq_{A \to \mathcal{P}(\Sigma)} \gamma \circ f^\sharp$*.*

When γ is an upper adjoint, the above is equivalent to $\alpha(f(S)) \sqsubseteq f^\sharp(\alpha(S))$, for all $S \in \mathcal{P}(\Sigma)$. The *most precise* ("best") sound $f^\sharp$ for f is $f^\sharp_{best} = \alpha \circ f \circ \gamma$. See Figure 3 for $succ^\sharp$ and $pred^\sharp$ and an example static analysis.

When the inequalities that define soundness are strengthened to equalities, we have:

Abstractly interpret $\mathcal{P}(Int)$ by $Sign = \{neg, zero, pos, any, none\}$; the abstracted program and flowgraph are

p_0 : $readSign(\mathtt{x})$
p_1 : `if` $gzT^\sharp(\mathtt{x})$:
p_2 : `x:=` $pred^\sharp(\mathtt{x})$
p_3 : `x:=` $succ^\sharp(\mathtt{x})$
p_4 : $writeSign(\mathtt{x})$



The abstract transition functions are

$succ^\sharp(pos) = pos$	$pred^\sharp(neg) = neg$
$succ^\sharp(zero) = pos$	$pred^\sharp(zero) = neg$
$succ^\sharp(neg) = any$	$pred^\sharp(pos) = any$
$succ^\sharp(any) = any$	$pred^\sharp(any) = any$

$gzT^\sharp(neg) = none$	$gzF^\sharp(neg) = neg$
$gzT^\sharp(zero) = none$	$gzF^\sharp(zero) = zero$
$gzT^\sharp(pos) = pos$	$gzF^\sharp(pos) = none$
$gzT^\sharp(any) = pos$	$gzF^\sharp(any) = any$ (!)

(All functions, $f^\sharp$, are *strict*: $f^\sharp(none) = none$.)

We now calculate a static analysis*, which applies the abstracted transition functions to the abstracted inputs (at* p_0*), computing the abstracted outputs (at* p_4*):*

$$\{zero \mapsto pos,\ neg \mapsto any,\ pos \mapsto any, \}$$

Fig. 3. Static analysis that calculates sign properties of example program

Definition 3. $f^\sharp$ *is* γ-complete (forwards complete) for f *iff* $f \circ \gamma = \gamma \circ f^\sharp$ *[17, 29].* $f^\sharp$ *is* α-complete (backwards complete) for f *iff* $\alpha \circ f = f^\sharp \circ \alpha$ *[11, 18]*

These notions will be explained, developed, and shown useful in Section 3, where we see the relation between logic and the transition functions.

2 Elements of *A* are logical properties

In Figure 2, note how neg and pos are both names of state sets as well as logical assertions (*"isNeg," "isPos"*). In data-flow analysis, this "pun" — state-set approximations are logical properties — is critical to program transformation, because an estimated state set, $S \subseteq \Sigma$, has *property* $S' \subseteq \Sigma$ if $S \subseteq S'$ holds.

In an analysis like the one in Figure 3, we say that an output, $a \in Sign$, has *property* $a' \in Sign$ if $a \sqsubseteq a'$. (For example, $zero$ has properties $zero$ and any.) Thus, all the concrete states modelled by a have property a', too.

In predicate-abstraction-based static analysis, the abstract data values are sets of primitive propositions (e.g., $\{\mathtt{x} > 0, \mathtt{x} \leq \mathtt{y}\}$), which can also be read as propositions ($\mathtt{x} > 0 \wedge \mathtt{x} \leq \mathtt{y}$) that denote concrete-state sets ($\{(x, y) \in \Sigma \mid x > 0 \text{ and } x \leq y\}$).

This idea is exploited in counter-example-guided refinement [1, 2, 19, 28], which enriches the abstract domain with additional primitive propositions as needed to validate a logical property. This idea also underlies the definition of Kripke structure, used in abstract model checking [4], where each state partition is married to the set of primitive propositions that hold true for the partition.

These observations are hardly novel — they harken back to Cousot's Ph.D. thesis [5] and his tutorial paper [6]. Now we develop the consequences.

2.1 Model theory of A

Treat A as a set of logical propositions. Define the entailment relation, $\models \subseteq \mathcal{P}(\Sigma) \times A$, as $S \models a$ iff $S \subseteq \gamma(a)$. (When γ is an upper adjoint, $S \models a$ iff $\alpha(S) \sqsubseteq a$ as well.) One key consequence is that $S' \subseteq S$ and $S \models a$ imply $S' \models a$.

An abstract transition function is exposed as a *postcondition transformer*: $S \models a$ implies $f(S) \models f^\sharp(a)$, and this makes $f^\sharp_{best}$ the *strongest* postcondition transformer for f in the language of propositions expressible in A.

2.2 Proof theory of A

For $a, a' \in A$, define $a \vdash a'$ iff $a \sqsubseteq a'$. (Recall that we require that $\sqsubseteq$ be finitely computable, and the "proof" that $a \vdash a'$ is the computation that affirms $a \sqsubseteq a'$.)

As is standard [14], assert $a \models a'$ iff for all $S \subseteq \Sigma$, $S \models a$ implies $S \models a'$. Evidently, $a \models a'$ iff $\gamma(a) \subseteq \gamma(a')$.

Proposition 4. (soundness) *For all $a, a' \in A$, $a \vdash a'$ implies $a \models a'$ (which implies $S \models a'$, for all $S \subseteq \gamma(a)$).*

Proof. Immediate from γ's monotonicity. □

Soundness justifies validations and transformations based on a static analyis.

Proposition 5. (completeness) *When γ is an upper adjoint and an injective (1-1) function, then $a \models a'$ implies $a \vdash a'$ for all $a, a' \in A$.*

Proof. Assume $\gamma(a) \subseteq \gamma(a')$. By the definition of Galois connection, this gives $\alpha(\gamma(a))) \sqsubseteq a'$. Since γ is injective, $\alpha(\gamma(a))) = a$. □

The best abstract transition function computes post-images that are complete:

Theorem 6. (image completeness) *When γ is an upper adjoint, then $f^\sharp_{best} = \alpha \circ f \circ \gamma$ is* image complete *in the sense that, for all $a, a' \in A$,*

1. $f^\sharp_{best}(a) \models a'$ *iff* $f^\sharp_{best}(a) \vdash a'$
2. $f(\gamma(a)) \models a'$ *iff* $f^\sharp_{best}(a) \vdash a'$

Proof. For 1., we need to prove the only-if part: Assume $\gamma(f^\sharp_{best}(a)) \subseteq \gamma(a')$. By the definition of $f^\sharp_{best}$ and the Galois connection, we have $f \circ \gamma(a) \subseteq \gamma \circ \alpha \circ f \circ \gamma(a) \subseteq \gamma(a')$. By applying monotone α to the previous inclusions and appealing to the definition of Galois connection, we get $f^\sharp_{best}(a) = \alpha \circ f \circ \gamma(a) \sqsubseteq \alpha \circ \gamma(a') \sqsubseteq a'$. The proof of 2. is similar. $\square$

Image completeness does *not* ensure completeness upon *multiple* applications of transition functions, which static analysis must do in practice. In Figure 2, note that $pred(succ\{0\}) \models zero$, yet $pred^\sharp(succ^\sharp(zero)) \not\models zero$ — the problem is that, although $\{0\} = \gamma(zero)$ falls in the range of γ, $succ(\gamma(zero)) = \{1\}$ does *not* and cannot be expressed precisely within $Sign$.

For the remainder of this subsection, assume that γ is an upper adjoint. First, say that $f : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ *is γ-complete* if there exists some $f^\sharp : A \to A$ that is γ-complete for f (cf. Defn. 3): evidently, $f \circ \gamma = \gamma \circ f^\sharp_{best}$. It is well known that f is γ-complete iff for all $a \in A$, $f(\gamma(a)) \in \gamma[A]$, that is, f stays in γ's range [17].

When f is γ-complete, then its repeated application to an initial argument in γ's range can be precisely approximated by $f^\sharp_{best}$.[1]

An inappropriate choice of A can prevent a transition function from γ-completeness, e.g., $succ : \mathcal{P}(Int) \to \mathcal{P}(Int)$ is not γ-complete for $Sign$. The repeated applications of $succ$ to $\{0\} = \gamma(zero)$ require that we add to $Sign$ new elements, I, for each $\{i\}$, $i > 0$, so that $\gamma(I) = \{i\}$ and $succ$ is γ-complete. This is too expensive of a price to pay, but we see in the next section that γ-completeness plays a critical role in determining a domain's *internal logic*.

3 Internal logic of A

We understand a *logic* as an inductively defined assertion set, an inductively defined interpretation, and (an inductively defined) proof theory, typically presented as a set of deduction rules. We now explore the logic that is internal to domain A and concretization map γ.

First, we treat the elements of A as primitive propositions and we use ϕ and ψ to represent elements from A. $\gamma : A \to \mathcal{P}(\Sigma)$ interprets the primitive propositions.

Definition 7. *Abstract domain A's internal logic* has conjunction *when*

$$S \models \phi_1 \sqcap \phi_2 \textit{ iff } S \models \phi_1 \textit{ and } S \models \phi_2, \textit{ for all } S \subseteq \Sigma.$$

Proposition 8. *When γ preserves binary meet as set intersection — $\gamma(\phi \sqcap \psi) = \gamma(\phi) \cap \gamma(\psi)$, for all $\phi, \psi \in A$ — then A's internal logic has conjunction.*

[1] It is also true that precise approximation of multiple applications of f will be maintained if some $f^\sharp$ is *α-complete for f*. (f is α-complete iff for all $S, S' \in \mathcal{P}(S)$, $\alpha(S) = \alpha(S')$ implies $\alpha(f(S)) = \alpha(f(S'))$ — f maps α-related arguments to α-related answers. Alas, when a function is not α-complete, the cost of adding extra elements to A can be just as expensive as when γ-completeness is desired.

Recall that when γ is an upper adjoint, it preserves meets; this is a major benefit of structuring A so that it admits a Galois connection.

Now, we have this inductively defined assertion set, the *internal logic* of A:

$$\phi \ ::= \ a \mid \phi_1 \sqcap \phi_2$$

γ interprets the logic, and it satisfies these inductively defined laws:

$$\begin{aligned}&\gamma(a) = \text{given}\\ &\gamma(\phi \sqcap \psi) = \gamma(\phi) \cap \gamma(\psi)\end{aligned}$$

This logic of primitives and conjunction is already strong enough to express most predicate abstractions and CEGAR structures. The internal logic for *Sign* in Figure 2 possesses conjunction.

Sign's proof theory is defined by the finitely computable $\sqsubseteq$, which obviates the usual set of deduction rules. A static analysis proves its facts, $\phi \vdash \psi$, using $\sqsubseteq$, and this is one of the beauties of the subject.

Now that conjunction exists, we can read the earlier definition of $\models \subseteq A \times A$ in the classical way: For $\Delta \subseteq A$, define $\Delta \models \psi$ iff for all $S \subseteq \Sigma$, if (for all $\psi \in \Delta$, $S \models \psi$) then ($S \models \phi$) as well. Evidently, $\Delta \models \psi$ iff $\gamma(\sqcap \Delta) \subseteq \gamma(\psi)$.

We can explore for other propositional connectives:

Proposition 9. *If γ preserves binary join as set union, then A's internal logic has* disjunction*: $S \models \phi \sqcup \psi$ iff $S \models \phi$ or $S \models \psi$, where $\gamma(\phi \sqcup \psi) = \gamma(\phi) \cup \gamma(\psi)$.*

The *Sign* domain in Figure 2 lacks disjunction: $zero \models neg \sqcup pos$ (because $neg \sqcup pos = any$ but $zero \not\models neg$ and $zero \not\models pos$). If we add new elements to *Sign*, namely, $\leq 0, \neq 0$, and ≥ 0, we have disjunction for the expanded domain.[2]

We can search for intuitionistic (Heyting) implication: Complete lattice A is *distributive* if $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$, for all $a, b, c \in A$; this makes the set, $\{a \in A \mid a \sqcap \phi \sqsubseteq \psi\}$, directed, and when $\sqcap$ is Scott-continuous, then

$$\phi \Rightarrow \psi \ \equiv \ \sqcup \{a \in A \mid a \sqcap \phi \sqsubseteq \psi\}$$

defines *implication* in A, where $a \vdash \phi \Rightarrow \psi$ iff $a \sqcap \phi \vdash \psi$ [13].

Proposition 10. *If A is a complete distributive lattice where $\sqcap$ is Scott-continuous and upper adjoint γ is injective, then A's internal logic has* Heyting implication*: $S \models \phi \Rightarrow \psi$ iff $\gamma(\alpha(S)) \cap \gamma(\phi) \subseteq \gamma(\psi)$, where*

$$\gamma(\phi \Rightarrow \psi) = \bigcup \{S \in \gamma[A] \mid S \cap \gamma(\phi) \subseteq \gamma(\psi)\}.$$

Proof. Let $T = \{a \mid a \sqcap \phi \sqsubseteq \psi\}$. First, by [13], $\sqcup T \in T$; This implies $\gamma(\sqcup T) = \cup_{a \in T} \gamma(a) = \cup\{\gamma(a) \mid a \sqcap \phi \sqsubseteq \psi\}$. Consider the predicate, $a \sqcap \phi \sqsubseteq \psi$; since γ is injective, the predicate is equivalent to $\gamma(a \sqcap \phi) \subseteq \gamma(\psi)$ (cf. Proposition 5), which is equivalent to $\gamma(a) \cap \gamma(\phi) \subseteq \gamma(\psi)$, because γ preserves meets. So we have $\gamma(\phi \Rightarrow \psi) = \cup\{\gamma(a) \mid \gamma(a) \cap \gamma(\phi) \subseteq \gamma(\psi)\} = \cup\{S \in \gamma[A] \mid S \cap \gamma(\phi) \subseteq \gamma(\psi)\} \in \gamma[A]$.

Next, $S \models \phi \Rightarrow \psi$ iff $S \subseteq \gamma(\sqcup T)$, implying $S \subseteq \gamma(\alpha(S)) \subseteq \gamma(\alpha(\gamma(\sqcup T))) = \gamma(\sqcup T)$. So, $S \models \phi \Rightarrow \psi$ iff $\gamma(\alpha(S)) \models \phi \Rightarrow \psi$. Finally, because $\gamma(\alpha(S)) \in \gamma[A]$ and pointwise reasoning on set union, $\gamma(\alpha(S)) \models \phi \Rightarrow \psi$ iff $\gamma(\alpha(S)) \cap \gamma(\phi) \subseteq \gamma(\psi)$. $\square$

[2] By adding these elements, we computed the *disjunctive completion* of *Sign* [10].

Heyting implication is weaker than classical implication (where $S \models \phi \Rightarrow \psi$ iff $S \cap \gamma(\phi) \subseteq \gamma(\psi)$ iff for all $c \in S$, if $\{c\} \models \phi$, then $\{c\} \models \psi$).

As an immediate consequence, we have a Deduction Theorem: $\Delta, \psi \vdash \phi$ iff $\Delta \vdash \psi \Rightarrow \phi$. And if $\gamma(\bot_A) = \emptyset \in \mathcal{P}(\Sigma)$, we have falsity ($\bot$); this gives us

$$\phi ::= a \mid \phi_1 \sqcap \phi_2 \mid \phi_1 \sqcup \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \bot$$

In particular, $\neg\phi$ abbreviates $\phi \Rightarrow \bot$ and defines the *refutation* of ϕ within A, such as is done with a three-valued static analyzer such as TVLA [27]. In practice, most static analyses do *not* require all this structure — conjunction alone (cf. Figure 2) often suffices.

3.1 The general principle is γ-completeness

There is a general principle for determining when an operation is "logical" and is present in A's internal logic as a connective. To expose this principle, consider again the interpretation of conjunction, where both the connective ($\sqcap$) and its concrete interpretation ($\cap$) are stated as binary functions:

$$\gamma(\sqcap(\phi, \psi)) = \cap(\gamma(\phi), \gamma(\psi))$$

γ-completeness is *exactly* the criterion for determining which connectives are embedded in A:

Corollary 11. *For $f : \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \times \cdots \to \mathcal{P}(\Sigma)$, A's internal logic has connective $f^\sharp$ iff f is γ-complete: For all $\phi \in A$, $\gamma(f^\sharp(\phi_1, \phi_2, \cdots)) = f(\gamma(\phi_1), \gamma(\phi_2), \cdots)$.*

Example: reconsider *Sign* in Figure 2, and note that $negate : \mathcal{P}(Int) \to \mathcal{P}(Int)$, where $negate(S) = \{-n \mid n \in S\}$, is γ-complete. We have that $negate^\sharp : A \to A$ (where $negate^\sharp(pos) = neg$, $negate^\sharp(neg) = pos$, etc.) exists in *Sign*'s logic:

$$\phi ::= a \mid \phi_1 \sqcap \phi_2 \mid negate^\sharp(\phi)$$

We can state "negate" assertions, e.g., $pos \vdash negate^\sharp(neg \sqcap any)$. $negate^\sharp$ is a connective, a modality, a *predicate transformer*.

3.2 Predicate transformers in the internal logic

We saw from the example for *Sign* that the absence of γ-completeness for transition functions *succ* (and *pred*) made it impossible to prove $pred^\sharp(succ^\sharp(zero)) \vdash zero$ — the two transition functions *cannot* be connectives in *Sign*'s logic.

But even when a transition function, $f : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$, is γ-complete, it is *not* used as a connective — for program analysis and transformation, it is not useful to know $a \vdash f^\sharp(\phi)$, that is, state set $\gamma(a)$ falls in f's postimage of $\gamma(\phi)$. It is more useful to know $f^\sharp(a) \vdash \phi$, that is, f's postimage of $\gamma(a)$ lies within $\gamma(\phi)$.

In programming logic, $f^\sharp(a) \vdash \phi$ is written $a \vdash [f^\sharp]\phi$, using a precondition *predicate transformer*. To formalize, for $S \subseteq \mathcal{P}(\Sigma)$, define

$$[f](S) = \widetilde{pre}_f(S) = \bigcup\{S' \in \Sigma \mid f(S') \subseteq S\}$$

We wish to add $[f]$ to the internal logic, so we must show it is γ-complete:

Theorem 12. *Assume that γ is an upper adjoint and preserves joins. Then, $\widetilde{pre}_f$ is γ-complete iff f is α-complete.*

Proof. To prove the if part, we show that $\widetilde{pre}_f(\gamma(a)) = \cup\{S \mid f(S) \subseteq \gamma(a)\}$ is in $\gamma[A]$. By definition of Galois connection, $f(S) \subseteq \gamma(a)$ iff $\alpha \circ f(S) \sqsubseteq a$, and because f is α-complete, this is equivalent to $\alpha \circ f \circ \gamma \circ \alpha(S) \sqsubseteq a$. Again, by the Galois connection, we have the equivalent $f \circ \gamma \circ \alpha(S) \sqsubseteq \gamma(a)$. Thus, $\widetilde{pre}_f(\gamma(a)) = \cup\{S \mid f \circ \gamma \circ \alpha(S) \sqsubseteq \gamma(a)\}$. Now, when some $S \in \mathcal{P}(\Sigma)$ belongs to the set, then so must $\gamma \circ \alpha(S)$ (which is a superset of S): we have $\widetilde{pre}_f(\gamma(a)) = \cup\{S \in \gamma[A] \mid f(S) \subseteq \gamma(a)\}$. Finally, because γ preserves joins, the union of the set must be itself be in $\gamma[A]$.

For the only-if part, assume that $\widetilde{pre}_f$ is γ-complete, that is, for all $a \in A$, $\cup\{S \mid \alpha(f(S)) \sqsubseteq a\} \in \gamma[A]$. To show f is α-complete, we must prove that if $\alpha(S_0) = \alpha(S_1)$, then $\alpha(f(S_0)) = \alpha(f(S_1))$. Assume $\alpha(S_0) = \alpha(S_1)$; we first show $\alpha(f(S_0)) \sqsubseteq \alpha(f(S_1))$. Consider the set, $T_1 = \{S \mid \alpha(f(S)) \sqsubseteq \alpha(f(S_1))\}$. First, note that $S_1 \subseteq \cup T_1$, implying that $\alpha(S_1) \sqsubseteq \alpha(\cup T_1)$. Since $\alpha(S_0) = \alpha(S_1)$, we have $\alpha(S_0) \sqsubseteq \alpha(\cup T_1)$ as well, and this implies $S_0 \subseteq \gamma(\alpha(S_0)) \subseteq \gamma(\alpha(\cup T_1)) = \cup T_1$, since $\cup T_1 \in \gamma[A]$. Within $\mathcal{P}(\Sigma)$, it must be that $S_0 \in T_1$, implying $\alpha(f(S_0)) \sqsubseteq \alpha(f(S_1))$. Identical reasoning with $T_0 = \{S \mid \alpha(f(S)) \sqsubseteq \alpha(f(S_0))\}$ yields the other inclusion, meaning that f is α-complete. $\square$

Similar developments are possible with the other common variants of predicate transformers, but the technicalities increase [30].

Because of their dependence on α-γ-completeness, predicate transformers might not appear in an internal logic. In this case, we must *underapproximate* the transformers with a logic that is *external* to the abstract domain.

4 External logic

This paper argues that the logical reasoning one uses with a static analysis should be based on the abstract domain's internal logic. Yet, transition functions that lack α- and γ-completeness can make such reasoning hugely imprecise, and adding additional elements to make the abstract domain complete can be too expensive or destroy finite computability — One might be forced to work with a less-precise logic that lives "outside" the internal logic.

Also, it is not uncommon to be presented with a set of assertions, $\mathcal{L}$, and an interpretation, $[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}(\Sigma)$, already fixed for the concrete domain, $\mathcal{P}(\Sigma)$, prior to the selection of A. For Figure 3, *Sign* lacks disjunction and both $succ^\sharp$ and $pred^\sharp$ and neither α- nor γ-complete, but perhaps the logic in Figure 4 is demanded, nonetheless. *How do we deal with this?*

Common sense suggests, for each assertion form, ϕ, that we collect all the $a \in A$ that satisfy ϕ and define an abstract interpretation of $[\![\phi]\!]$ as follows:

$$[\![\phi]\!]^\sharp = \{a \mid \gamma(a) \subseteq [\![\phi]\!]\}$$

$$\phi \ ::= \ a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [f]\phi \qquad \text{for } a \in \mathit{Sign} \text{ and } f \in \{\mathit{succ}, \mathit{pred}\}$$

$$[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}(\Sigma)$$

$$\begin{aligned}
&[\![a]\!] = \gamma(a)\\
&[\![\phi_1 \wedge \phi_2]\!] = [\![\phi_1]\!] \cap [\![\phi_2]\!]\\
&[\![\phi_1 \vee \phi_2]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!]\\
&[\![[f]\phi]\!] = \widetilde{pre}_f[\![\phi]\!] = \cup\{S \mid f(S) \subseteq [\![\phi]\!]\}
\end{aligned}$$

Fig. 4. Sample logic for sign properties

Then, we can use the results of a static analysis based on A to prove properties: assert $a \vdash \phi$ iff $a \in [\![\phi]\!]^\sharp$. This defines a logic that is *external* to A.

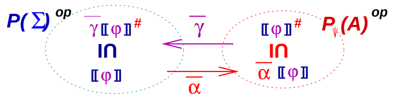
Underlying this informal development is a Galois connection whose abstract domain is $\mathcal{P}_\downarrow(A)^{op}$ — downclosed subsets of A, ordered by superset:

$$\begin{aligned}
&\mathcal{P}_\downarrow(A)^{op} = (\{T \subseteq A \mid T \text{ is downclosed}\}, \supseteq)\\
&\quad \text{where } T \text{ is } \textit{downclosed} \text{ iff } T = \{a \in A \mid \exists b \in T, a \sqsubseteq b\}
\end{aligned}$$

Elements of $\mathcal{P}_\downarrow(A)^{op}$ serve as denotations of $[\![\phi]\!]^\sharp \in \mathcal{P}_\downarrow(A)^{op}$.[3]

Here is the Galois connection: Let A be a partially ordered set and $\mathcal{P}(\Sigma)^{op} = (\mathcal{P}(\Sigma), \supseteq)$. Then, for any monotone $\gamma : A \to \mathcal{P}(\Sigma)$, the functions $\overline{\alpha} : \mathcal{P}(\Sigma)^{op} \to \mathcal{P}_\downarrow(A)^{op}$ and $\overline{\gamma} : \mathcal{P}_\downarrow(A)^{op} \to \mathcal{P}(\Sigma)^{op}$ form a Galois connection, where

$$\begin{aligned}
&\overline{\gamma}(T) = \bigcup\{\gamma(a) \mid a \in T\}\\
&\overline{\alpha}(S) = \bigcup\{T \mid S \supseteq \overline{\gamma}(T)\} = \{a \mid \gamma(a) \subseteq S\}
\end{aligned}$$



Upper adjoint $\overline{\gamma}$ "lifts" from γ and preserves unions; when γ is an upper adjoint, then $\overline{\gamma}$ preserves intersections, too [30]. $\overline{\alpha}$ is exactly the approximation we guessed earlier: $[\![\phi]\!]^\sharp = \overline{\alpha}[\![\phi]\!]$. The inverted ordering gives *underapproximation*: $[\![\phi]\!] \supseteq \overline{\gamma}[\![\phi]\!]^\sharp$. We have soundness — $a \in [\![\phi]\!]^\sharp$ implies $\gamma(a) \subseteq [\![\phi]\!]$ — because we used the adjoint to define the abstract interpretation of the logic.[4]

$\mathcal{P}_\downarrow(A)^{op}$ is itself an abstract domain, a complete lattice, and *its* internal logic contains disjunction (set union) and conjunction (set intersection) when γ is an upper adjoint. The domain is distributive, but $\overline{\gamma}$ might not be injective, so Heyting implication is not ensured (cf. Proposition 10). But it is the existence of disjunction that is the key to defining a sound $[\![\cdot]\!]^\sharp$. To summarize,

Proposition 13. *For* all *choices of partially ordered set, A, monotone $\gamma : A \to \mathcal{P}(\Sigma)$, and $[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}(\Sigma)$, there is a sound external logic, $[\![\cdot]\!]^\sharp : \mathcal{L} \to \mathcal{P}_\downarrow(A)$, in the sense that, for all $a \in A$, $a \in [\![\phi]\!]^\sharp$ implies $\gamma(a) \subseteq [\![\phi]\!]$, for all $\phi \in \mathcal{L}$. (Indeed, the most precise such logic is $\overline{\alpha} \circ [\![\cdot]\!]$.)*

This proposition underlies "abstract model checking" [3], where A holds the names of partitions of Σ and γ maps each state-partition name to its members.

[3] Clearly, all $[\![\phi]\!]^\sharp$ are downclosed sets.

[4] Precisely stated, the best approximation of $[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}_\downarrow(A)$ is $\overline{\alpha} \circ [\![\cdot]\!] \circ id_{\mathcal{L}}$.

$$
\begin{aligned}
&[\![a]\!]^\sharp_{best} = \overline{\alpha}(\gamma(a))\\
&[\![\phi_1 \wedge \phi_2]\!]^\sharp_{best} = \overline{\alpha}\,(\overline{\gamma}[\![\phi_1]\!]^\sharp_{best} \cap \overline{\gamma}[\![\phi_2]\!]^\sharp_{best})\\
&[\![\phi_1 \vee \phi_2]\!]^\sharp_{best} = \overline{\alpha}\,(\overline{\gamma}[\![\phi_1]\!]^\sharp_{best} \cup \overline{\gamma}[\![\phi_2]\!]^\sharp_{best})\\
&[\![[f]\phi]\!]^\sharp_{best} = \overline{\alpha}\,(\widetilde{pre}_f(\overline{\gamma}[\![\phi]\!]^\sharp_{best})) \;=\; \{a \mid f(\gamma(a)) \subseteq \overline{\gamma}[\![\phi]\!]^\sharp_{best}\}\\
\\
&[\![a]\!]^\sharp_{fin} = \overline{\alpha}(\gamma(a))\\
&[\![\phi_1 \wedge \phi_2]\!]^\sharp_{fin} = [\![\phi_1]\!]^\sharp_{fin} \cap [\![\phi_2]\!]^\sharp_{fin}\\
&[\![\phi_1 \vee \phi_2]\!]^\sharp_{fin} = [\![\phi_1]\!]^\sharp_{fin} \cup [\![\phi_2]\!]^\sharp_{fin}\\
&[\![[f]\phi]\!]^\sharp_{fin} = \widetilde{pre}_{f^\sharp}[\![\phi]\!]^\sharp_{fin} = \{a \in A \mid f^\sharp(a) \in [\![\phi]\!]^\sharp_{fin}\},\ \text{where } f^\sharp \text{ is sound for } f
\end{aligned}
$$

Fig. 5. Inductively defined logics for *Sign*: best and finitely computable

But we are not finished — we want an *inductively defined* abstract interpretation. This is readily obtained from the inductively defined concrete interpretation, whose equations take the form,

$$[\![\mathtt{f}(\phi_1, \phi_2, \cdots)]\!] = f([\![\phi_1]\!], [\![\phi_2]\!], \cdots)$$

We use the adjoints to abstract each logical operation, $f : \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \times \cdots \to \mathcal{P}(\Sigma)$, by $f^\sharp_{best} = \overline{\alpha} \circ f \circ (\overline{\gamma} \times \overline{\gamma} \times \cdots)$. The most precise, inductively defined, abstract logic is therefore

$$[\![\mathtt{f}(\phi_1, \phi_2, \cdots)]\!]^\sharp_{best} = f^\sharp_{best}([\![\phi_1]\!]^\sharp_{best}, [\![\phi_2]\!]^\sharp_{best}, \cdots)$$

Because the fixed-point operators are well behaved, we can also define abstract recursion operators [11, 26].

An issue that arises with sets of abstract values is that the synthesized $f^\sharp_{best} : \mathcal{P}_\downarrow(A) \times \mathcal{P}_\downarrow(A) \times \cdots \to \mathcal{P}_\downarrow(A)$ might not be finitely computable — we must locate a computable approximation of it. Consider again $[\![\cdot]\!]$ in Figure 4; its most precise inductively defined logic, $[\![\cdot]\!]^\sharp_{best}$, and a finitely computable approximation, $[\![\cdot]\!]^\sharp_{fin}$, are stated in Figure 5. We see that $\cap = (\overline{\alpha} \circ \cap \circ (\overline{\gamma} \times \overline{\gamma}))$ — precision is preserved — but this is not true for $\cup$: For example, $any \in [\![neg \vee zero \vee pos]\!]^\sharp_{best} = Sign$ but $any \notin [\![neg]\!]^\sharp_{fin} \cup [\![zero]\!]^\sharp_{fin} \cup [\![pos]\!]^\sharp_{fin} = \{neg, zero, pos, none\}$.[5]

For predicate transformers, it is well known that $\widetilde{pre}_{f^\sharp}$ is sound for $\widetilde{pre}_f$, for any $f^\sharp$ sound for f. But it is grossly imprecise. We can improve it by replacing $f^\sharp : A \to A$ by $f^\sharp_\vee : A \to \mathcal{P}_\downarrow(A)$, where $f^\sharp_\vee(a) = \downarrow\{\alpha\{c\} \mid c \in f(\gamma(a))\}$.[6] For example, from Figure 3, $succ^\sharp(neg) = any$, but $succ^\sharp_\vee(neg) = \{neg, zero, none\}$, which gives the more precise $\widetilde{pre}_{succ^\sharp_\vee}$.[7]

These technicalities let us prove that $\widetilde{pre}_{f^\sharp_\vee} = (\overline{\alpha} \circ \widetilde{pre}_f \circ \overline{\gamma})$ [30].

[5] The problem is that $\{none, neg, zero, pos\}$ and *Sign* both concretize to $neg \vee zero \vee pos$ and are candidates to represent it in $\mathcal{P}_\downarrow(Sign)$. We must eliminate one of the sets.

[6] where $\downarrow S = \{s \mid \exists s' \in S, s \sqsubseteq s'\}$

[7] Underlying $f^\sharp_\vee$'s definition is yet another Galois connection, between complete lattices $(\mathcal{P}(\Sigma), \subseteq)$ and $(\mathcal{P}_\downarrow(A), \subseteq)$, where the upper adjoint is $\overline{\gamma}$ and the lower adjoint is $\overline{\alpha_o}(S) = \cap\{T \mid S \subseteq \overline{\gamma}(T)\} = \downarrow\{\alpha\{c\} \mid c \in S\}$. Then, $f^\sharp_\vee = \overline{\alpha_o} \circ f \circ \overline{\gamma}$. This Galois connection is possible when γ is an upper adjoint.

4.1 Provability, soundness, completeness

Entailment and provability for an inductively defined external logic is defined as expected: $a \models \phi$ iff $\gamma(a) \subseteq [\![\phi]\!]$, and $a \vdash \phi$ iff $a \in [\![\phi]\!]^{\sharp}_{fin}$.[8]

Soundness (that is, $\vdash$ implies $\models$) is immediate, and completeness ($\models$ implies $\vdash$) follows when $\overline{\alpha} \circ [\![\cdot]\!] = [\![\cdot]\!]^{\sharp}_{fin}$. This is called *logical best preservation* or logical $\overline{\alpha}$-completeness [11, 29].

There is another, independent, form of completeness, *logical strong preservation* or logical γ-completeness: $\overline{\gamma} \circ [\![\cdot]\!]^{\sharp}_{fin} = [\![\cdot]\!]$ [17, 26, 29].[9] For inductively defined $[\![\cdot]\!]^{\sharp}_{fin}$, if all the mentioned abstract logical operations are α-complete, then $[\![\cdot]\!]^{\sharp}_{fin}$ has best preservation; when all abstract logical operators are γ-complete, then $[\![\cdot]\!]^{\sharp}_{fin}$ has strong preservation. (The converses might not hold [26].)

5 When the upper adjoint preserves joins, the external logic lies within the inverted abstract domain

The "lift" of abstract domain A to $\mathcal{P}_{\downarrow}(A)$ is a *disjunctive completion* [10], where an element, $\{a_0, a_1, a_2, \cdots\} \in \mathcal{P}_{\downarrow}(A)$ is meant to be read as the disjunction, $a_0 \vee a_1 \vee a_2 \vee \cdots$, and this is confirmed by the definition, $\overline{\gamma}\{a_0, a_1, a_2, \cdots\} = \gamma(a_0) \cup \gamma(a_1) \cup \gamma(a_2) \cup \cdots$.

But say that γ is an upper adjoint and that it *preserves joins*, that is,

$$\overline{\gamma}T = \bigcup_{a \in T} \gamma(a) = \gamma(\bigsqcup_{a \in T} a) = \gamma(\sqcup T)$$

So, $\overline{\gamma}[\mathcal{P}_{\downarrow}(A)] = \gamma[A]$ — their ranges are equal — and *there is no new expressivity gained by using sets of A-elements to model subsets of Σ*. So, when upper adjoint γ preserves joins, an external logic can be modelled *internally* within A. The key is to *invert* A and define an underapproximating Galois connection:

Proposition 14. *If A is a complete lattice and $\gamma : A \to \mathcal{P}(\Sigma)$ preserves joins (as unions) and meets (as intersections), then*

- *γ is the upper adjoint of a Galois connection between $(\mathcal{P}(\Sigma), \subseteq)$ and $(A, \sqsubseteq)$, where the lower adjoint, α_o, is defined $\alpha_o(S) = \sqcap\{a \mid S \subseteq \gamma(a)\}$.*
- *γ is the upper adjoint of a Galois connection between $(\mathcal{P}(\Sigma), \supseteq)$ and $(A, \sqsupseteq)$, where the lower adjoint, α_u, is defined $\alpha_u(S) = \sqcup\{a \mid S \supseteq \gamma(a)\}$.*

The first Galois connection defines an overapproximation relationship; the second defines an underapproximation relationship.

When we approximate a state-transition function, $f : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$, we apply the first Galois connection to define $f^{\sharp}_{best} = \alpha_o \circ f \circ \gamma$. We call this the *computational interpretation* of f.

[8] These notions are equivalently stated with sets: for $T \in \mathcal{P}_{\downarrow}(A)$, $T \models \phi$ iff $\overline{\gamma}(T) \subseteq [\![\phi]\!]$, and $T \vdash \phi$ iff $T \subseteq [\![\phi]\!]^{\sharp}_{fin}$.

[9] Strong preservation asserts, for all $c \in \Sigma$, that $c \in [\![\phi]\!]$ iff $\alpha\{c\} \vdash \phi$. In contrast, best preservation states, for all $a \in A$, that $\gamma(a) \subseteq [\![\phi]\!]$ iff $a \vdash \phi$. See [29] for criteria when one implies the other.

When we are given a logical interpretation function, $[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}(\Sigma)$, we apply the second Galois connection to define $[\![\cdot]\!]^\sharp_u = \alpha_u \circ [\![\cdot]\!]$. If $[\![\cdot]\!]$ is inductively defined, that is, has form $[\![\mathtt{f}(\phi_1, \phi_2, \cdots)]\!] = f([\![\phi_1]\!], [\![\phi_2]\!], \cdots)$, we apply the second Galois connection to define $f^\flat_{best} = \alpha_u \circ f \circ (\gamma \times \gamma \times ...)$, giving $[\![\mathtt{f}(\phi_1, \phi_2, \cdots)]\!]^\sharp_{best} = f^\flat_{best}([\![\phi_1]\!]^\sharp_{best}, [\![\phi_2]\!]^\sharp_{best}, \cdots))$ We call this the *logical interpretation* of f.

When all the fs are α_u-complete, then $[\![\cdot]\!]^\sharp_u = [\![\cdot]\!]^\sharp_{best}$. We can also show that the logical interpretation proves the *same assertions* as the external logic:

First, for $[\![\phi]\!]^\sharp = \overline{\alpha}[\![\phi]\!] \in \mathcal{P}_\downarrow(A)$, recall that $a \vdash \phi$ iff $a \in [\![\phi]\!]^\sharp$.

Next, for $[\![\phi]\!]^\sharp_u = \alpha_u[\![\phi]\!] \in A$, define $a \vdash \phi$ iff $a \sqsubseteq [\![\phi]\!]^\sharp_u$.

Theorem 15. *For all $a \in A$, $a \sqsubseteq [\![\phi]\!]^\sharp_u$ iff $a \in [\![\phi]\!]^\sharp$.*

Proof. First, note that $a \in [\![\phi]\!]^\sharp$ iff $\gamma(a) \subseteq [\![\phi]\!]$. Next, $a \sqsubseteq [\![\phi]\!]^\sharp_u$ iff $a \sqsubseteq \sqcup P$, where $P = \{a' \mid \gamma(a') \subseteq [\![\phi]\!]\}$.

To prove the if-part, assume $\gamma(a) \subseteq [\![\phi]\!]$. This places $a \in P$, hence $a \sqsubseteq \sqcup P$.

To prove the only-if part, assume $a \sqsubseteq \sqcup P$. Now, for all $a' \in P$, $\gamma(a') \subseteq [\![\phi]\!]$, implying $\cup_{a' \in P}\gamma(a') \subseteq [\![\phi]\!]$. But γ preserves joins, meaning $\gamma(\sqcup P) \subseteq [\![\phi]\!]$, implying that $\sqcup P \in P$ as well. Since $a \sqsubseteq \sqcup P$, we have $\gamma(a) \subseteq \gamma(\sqcup P) \subseteq [\![\phi]\!]$. □

So, when γ preserves meets and also joins, we embed the external logic as an underapproximation in A^{op}, retaining the logic's proof theory and model theory.

6 Conclusion

Abstract interpretations are fundamentally "logical" — as Cousot and Cousot have stated in key papers [5, 6, 8, 9, 11] — an abstract interpretation estimates function pre- and post-images, which are represented as finitely-sized assertions. The same idea underlies Kripke structures and abstract model checking [3, 4].

In this paper, we showed that the connection between abstract interpretation and symbolic logic is fundamental: A static analysis *computes proofs* (via $\sqsubseteq$) that are *sound* (via $\models$) within the internal/external logic.

Acknowledgements: This paper was inspired by earlier work of Alan Mycroft and Neil Jones [23, 24]. I would like to thank Alan for his interest in internal/external logics and his suggestion that I write this paper.

References

1. T. Ball, A. Podelski, and S.K. Rajamani. Boolean and cartesian abstractions for model checking C programs. In *TACAS'01*, pages 268–283. LNCS 2031, 2001.
2. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, LNCS 1855, pages 154–169. Springer, 2000.
3. E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.
4. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 2000.

5. P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* PhD thesis, University of Grenoble, 1978.
6. P. Cousot. Semantic foundations of program analysis. In S. Muchnick and N. Jones, editors, *Program Flow Analysis*, pages 303–342. Prentice Hall, 1981.
7. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs. In *Proc. 4th ACM Symp. POPL*, pages 238–252, 1977.
8. P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: mathematical foundations. In *Symp. Artificial Intelligence and Programming Languages*. ACM SIGART Newsletter 64, 1977.
9. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. POPL*, pages 269–282, 1979.
10. P. Cousot and R. Cousot. Higher-order abstract interpretation. In *Proceedings IEEE Int. Conf. Computer Lang.*, pages 95–112, 1994.
11. P. Cousot and R. Cousot. Temporal abstract interpretation. In *Proc. 27th ACM Symp. on Principles of Programming Languages*, pages 12–25. ACM Press, 2000.
12. B.A. Davey and H.A Priestley. *Introduction to Lattices and Order, 2d ed.* Cambridge Univ. Press, 2002.
13. M. Dummett. *Intuitionism.* Oxford University Press, 1977.
14. H. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.
15. D. Lacey et al. Proving correctness of compiler optimizations by temporal logic. In *Proc. 29th ACM POPL*, 2002.
16. G. Gierz et al. *Continuous Lattices and Domains.* Cambridge University Press, Cambridge, England, 2003.
17. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples, and refinements in abstract model checking. In *SAS'01*, pages 356–373. LNCS 2126, 2001.
18. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47:361–416, 2000.
19. S. Graf and H. Saidi. Verifying invariants using theorem proving. In *Proc. CAV'96*, Springer LNCS 1102, 1996.
20. M. Hecht. *Flow Analysis of Computer Programs.* Elsevier, 1977.
21. N. Jones and F. Nielson. Abstract interpretation. In S. Abramsky et al., editor, *Handbook of Logic in Computer Science, Vol. 4.* Oxford Univ. Press, 1995.
22. S. Muchnick and N.D. Jones, editors. *Program Flow Analysis: Theory and Applications.* Prentice-Hall, 1981.
23. A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proc. ACM Symp. Partial Evaluation (PEPM'93)*, pages 179–185, 1993.
24. A. Mycroft and N.D. Jones. A relational framework for abstract interpretation. In *Programs as Data Objects*, LNCS 217, pages 156–171. Springer Verlag, 1985.
25. F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In *Proc. ESOP*, LNCS 2986, pages 18–32. Springer, 2004.
26. F. Ranzato and F. Tapparo. Strong preservation of temporal fixpoint-based operators. In *Proc. VMCAI'06*, LNCS 3855, pages 332–347. Springer, 2006.
27. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24:217–298, 2002.
28. H. Saidi. Model checking guided abstraction and analysis. In *Proc. SAS'00*, pages 377–396. Springer LNCS 1824, 2000.
29. D.A. Schmidt. Comparing completeness properties of static analyses and their logics. In *Proc. APLAS'06*, LNCS 4279, pages 183–199. Springer, 2006.
30. D.A. Schmidt. Underapproximating predicate transformers. In *Proc. SAS'06*, LNCS 4134, pages 127–143. Springer, 2006.