Constructive Galois Connections

Taming the Galois Connection Framework for Mechanized Metatheory

David Darais
University of Maryland, USA
darais@cs.umd.edu

David Van Horn
University of Maryland, USA
dvanhorn@cs.umd.edu

Abstract

Galois connections are a foundational tool for structuring abstraction in semantics and their use lies at the heart of the theory of abstract interpretation. Yet, mechanization of Galois connections has remained limited to certain restricted modes of use, preventing their fully general application in mechanized metatheory and certified programming.

This paper presents *constructive Galois connections*, a framework for Galois connections that is effective both on paper and in proof assistants; is complete with respect to the set of Galois connections with computational content; and enables more general reasoning principles, including the "calculational" style advocated by Cousot.

Crucial to our technical approach is the addition of monadic structure to Galois connections to control a "specification effect." Effectful calculations may reason classically, while pure calculations have extractable computational content. Explicitly moving between the worlds of specification and implementation is enabled by our metatheory.

To validate our approach, we provide two case studies in mechanizing existing proofs from the literature: one uses calculational abstract interpretation to design a static analyzer, the other forms a semantic basis for gradual typing. Both mechanized proofs closely follow their original paper-and-pencil counterparts, employ reasoning principles not captured by previous mechanization approaches, support the extraction of verified algorithms, and are novel.

Keywords Abstract interpretation; Galois connections; mechanized metatheory; certified static analysis

1. Introduction

Abstract interpretation is a general theory of sound approximation widely applied in programming language semantics, formal verification, and static analysis [9–13]. In abstract interpretation, properties of programs are related between a pair of partially ordered sets: a concrete domain, $\langle \mathcal{C}, \sqsubseteq \rangle$, and an abstract domain, $\langle \mathcal{A}, \leq \rangle$. When concrete properties have a $\leq$-most precise abstraction, the correspondence is a *Galois connection*, formed by a pair of mappings between the domains known as *abstraction* $\alpha \in \mathcal{C} \mapsto \mathcal{A}$ and *concretization* $\gamma \in \mathcal{A} \mapsto \mathcal{C}$ such that $\forall c \in \mathcal{C}. \forall a \in \mathcal{A}. \alpha(c) \leq a \iff c \sqsubseteq \gamma(a)$.

Since its introduction by Cousot and Cousot in the late 1970s, this theory has formed the basis of many forms of static analysis, type systems, model-checkers, obfuscators, program transformations, and many other applications.[1]

Given the remarkable set of intellectual tools contributed by this theory, an obvious desire is to incorporate its use into proof assistants in order to mechanically verify proofs by abstract interpretation and to extract certified artifacts, such as static analyzers, from these proofs.

Monniaux first achieved the goal of mechanization for the theory of abstract interpretation with Galois connections in Coq [22]. However, he notes that the abstraction side (α) of Galois connections poses a serious problem since it requires the admission of non-constructive axioms. Use of these axioms prevents the extraction of certified programs. So while Monniaux was able to mechanically verify proofs by abstract interpretation in its full generality, certified artifacts could not generally be extracted.

Pichardie subsequently tackled the extraction problem by mechanizing a restricted formulation of abstract interpretation that relied only on the concretization (γ) side of Galois connections [25]. Doing so avoids the use of axioms and enables the extraction of certified artifacts. This proof technique has proved effective and has been used to construct several certified static analyzers [1, 5, 6, 25], most notably the Verasco static analyzer, part of the CompCert C compiler [16, 17]. However, the full generality of the theory is sacrificed. While in principle, the technique could achieve mechanization of existing soundness *theorems*, it cannot do so faithful to existing *proofs* and would require a wholesale redevelopment. In particular, Pichardie writes [25, p. 55]:[2]

> The framework we have retained nevertheless loses an important property of the standard framework: being able to derive a correct approximation $f^\sharp$ from the specification $\alpha \circ f \circ \gamma$. Several examples of such derivations are given by Cousot [7]. It seems interesting to find a framework for this kind of symbolic manipulation, while remaining easily formalizable in Coq.

[1] See `di.ens.fr/~cousot/AI/` for an extensive list.

[2] Translated from French by the present authors.

This important property is the so-called "calculational" style, whereby an abstract interpreter ($f^\sharp$) is derived in a correct-by-construction manner from a concrete interpreter (f) composed with abstraction and concretization ($\alpha \circ f \circ \gamma$). This style of abstract interpretation is detailed in Cousot's monograph "The calculational design of a generic abstract interpreter" [7], which concludes:

> The emphasis in these notes has been on the correctness of the design by calculus. The mechanized verification of this formal development using a proof assistant can be foreseen with automatic extraction of a correct program from its correctness proof.

However in the subsequent 17 years, this vision has remained unrealized and clearly the paramount technical challenge in achieving it is obtaining both *generality* and *constructivity* in a single mechanization framework.

This paper contributes *constructive Galois connections*, a framework for mechanized abstract interpretation with Galois connections that achieves generality and constructivity, thereby enabling calculational style proofs making use of both abstraction (α) and concretization (γ), while maintaining the ability to extract certified artifacts. Key to this achievement is incorporating monadic structure in Galois connections to isolate non-constructive specifications from constructive algorithms. Within the effectful fragment, all of classical Galois connection reasoning can be employed, while within the pure fragment, only so-called constructive Galois connections may be used. Remarkably, calculations can move between these modalities and verified programs may be extracted from the end result of calculation.

To support the utility of our theory, we build a library for constructive Galois connections in Agda [24] and mechanize two existing abstract interpretation proofs from the literature. The first is drawn from Cousot's monograph [7], which builds a correct-by-construction analyzer out of a specification induced by a concrete interpreter and Galois connection. The second is drawn from Garcia, Clark, and Tanter's "Abstracting gradual typing" [15], which uses abstract interpretation to derive static and dynamic semantics for gradually typed languages from traditional static types. Both proofs use exactly the "important property of the standard framework" identified by Pichardie and not handled by prior approaches. The mechanized proofs closely follow the original pencil-and-paper proofs, which use both abstraction and concretization, while still enabling the extraction of certified algorithms. Neither of these papers has been previously mechanized. Moreover, we know of no existing mechanized proof involving calculational abstract interpretation.

Finally, we develop the metatheory of constructive Galois connections, proving them sound and making precise their relation to classical Galois connections. The metatheory is itself mechanized; claims are marked with a "✓" whenever they are proved in Agda. (All claims are checked.)

Contributions This paper contributes the following:

- a foundational theory of constructive Galois connections that achieves a balance of generality and constructivity amenable to mechanization;
- a library for mechanized abstract interpretation; and
- the first mechanized proofs employing calculational abstract interpretation.

The remainder of the paper is organized as follows. We give a tutorial on abstract interpretation by way of a simple static analyzer, identifying problems with constructivity and the challenges of mechanization (§2). We then develop the intuitions of constructive Galois connections and introduce the monadic structure to achieve generality and extractability of certified artifacts, applying the technique to the simple static analyzer (§3). We then describe the mechanization of Cousot's calculational monograph (§4) and Garcia, Clark, and Tanter's work on gradual typing *via* abstract interpretation (§5). Finally, we develop the formal foundations of constructive Galois connections (§6), relate our work to the literature (§7), and conclude (§8).

2. Designing an Analysis

To motivate constructive Galois connections we sketch the design of an abstract interpreter for `WHILE`, a simple imperative language with loops. In the end, four equivalent statements of soundness for a candidate abstract interpreter are generated systematically through the Galois connection framework. One of these rules in particular is used exclusively in nearly every known result in verified abstract interpretation [2, 6, 16, 25], except for the very first attempt [22], which identified the fundamental difficulty in using any of the others. (For more discussion see Jourdan et al. [16, Sec. 10]; Pichardie [25, Ch. 3.3]; Monniaux [22, Ch. 2.2]; and our Section 2.3.) We observe, as others have before us, that practitioners avoid the three rules because they are notoriously difficult to translate into proof assistants. However, these unused rules have desirable properties which are missed in the current practice of avoiding them all together. We explore these issues in this section, and present our solution to recovering all rules and their benefits in Section 3 through a restricted mode of use of Galois connections, which we call *constructive Galois connections*.

2.1 Concrete Semantics

The syntax for `WHILE` is shown in Figure 1. `WHILE` syntactically distinguished arithmetic, boolean and command expressions. `RAND` is an arithmetic expression which operationally may evaluate to any integer. Syntactic categories $\oplus$, $\otimes$ and $\circledcirc$ range over arithmetic, comparison and boolean operators, and are introduced to simplify the presentation. The `WHILE` language and a large part of our tutorial analysis design is taken directly from Cousot [7].

The concrete semantics of `WHILE` is sketched without full definition in Figure 2. Denotation functions $[\![\_]\!]^a$,

$i \in$	$\mathbb{Z} \coloneqq \{\dots,-1,0,1,\dots\}$	integer literals
$b \in$	$\mathbb{B} \coloneqq \{\text{false},\text{true}\}$	boolean literals
$x \in$	$\text{var} \coloneqq \dots$	variables
$\oplus \in$	$\text{aop} ::= + \mid - \mid \times \mid /$	arithmetic op.
$\olessthan \in$	$\text{cmp} ::= < \mid =$	comparison op.
$\ovee \in$	$\text{bop} ::= \vee \mid \wedge$	boolean op.
$ae \in$	$\text{aexp} ::= i \mid x \mid \texttt{RAND} \mid ae \oplus ae$	arithmetic exp.
$be \in$	$\text{bexp} ::= b \mid ae \olessthan ae \mid be \ovee be$	boolean exp.
$ce \in$	$\text{cexp} ::= \texttt{SKIP} \mid ce; ce \mid x \coloneqq ae$	
	$\mid \texttt{IF}\ be\ \texttt{THEN}\ ce\ \texttt{ELSE}\ ce$	
	$\mid \texttt{WHILE}\ be\ \texttt{DO}\ ce$	command exp.

Figure 1. `WHILE` syntax

$⟦\_⟧^c$ and $⟦\_⟧^b$ give semantics to arithmetic, conditional and boolean operators. The semantics of compound syntactic expressions are given relationally with $\_ \vdash \_ \Downarrow^a \_$, $\_ \vdash \_ \Downarrow^b \_$ and $\_ \mapsto \_$, due to the nondeterminism of `RAND` and nontermination of `WHILE`. These semantics serve as the starting point for designing an abstract interpreter.

2.2 Abstract Semantics with Galois Connections

Following a Galois connection discipline, an abstract semantics for `WHILE` is designed in the following steps:

1. *Abstracting Sets*: An abstraction for each set $\mathbb{Z}$, $\mathbb{B}$ and env.
2. *Abstracting Functions*: An abstraction for each denotation function $⟦\_⟧^a$, $⟦\_⟧^c$ and $⟦\_⟧^b$.
3. *Abstracting Relations*: An abstraction for each semantics relation $\_ \vdash \_ \Downarrow^a \_$, $\_ \vdash \_ \Downarrow^b \_$ and $\_ \mapsto \_$.

Each abstract set must be shown to form a *Galois connection* with its concrete counterpart. Soundness criteria (although not their proofs) are then automatically synthesized for candidate abstract functions and relations, following the Galois connections built for each abstract set. We describe this process for integers and environments (the sets $\mathbb{Z}$ and env), arithmetic operators (the denotation function $⟦\_⟧^a$), and arithmetic expressions (the semantics relation $\_ \vdash \_ \Downarrow^a \_$).

Abstracting Integers To abstract integers, the goal is to design a set with finite elements, where each element "represents" a potentially infinite collection of concrete integers. We choose a simple sign abstract for integers for the purpose of demonstration, but other more complex abstractions are certainly possible [20].

$$i^\sharp \in \mathbb{Z}^\sharp \coloneqq \{\bot, \textbf{NEG}, \textbf{ZER}, \textbf{POS}, \textbf{NEGZ}, \textbf{NZER}, \textbf{POSZ}, \top\}$$

Abstract integers are given meaning through a concretization function γ^z which specifies the set of concrete integers represented by an abstract integer:

$$\begin{array}{lll}
\gamma^z \in \mathbb{Z}^\sharp \to \mathcal{P}(\mathbb{Z}) & \gamma^z(\textbf{ZER}) \coloneqq \{0\} & \gamma^z(\textbf{NEGZ}) \coloneqq \{i \mid i \leq 0\} \\
\gamma^z(\bot) \coloneqq \{\} & \gamma^z(\textbf{POS}) \coloneqq \{i \mid i > 0\} & \gamma^z(\textbf{NZER}) \coloneqq \{i \mid i \neq 0\} \\
\gamma^z(\top) \coloneqq \{i \mid i \in \mathbb{Z}\} & \gamma^z(\textbf{NEG}) \coloneqq \{i \mid i < 0\} & \gamma^z(\textbf{POSZ}) \coloneqq \{i \mid i \geq 0\}
\end{array}$$

$$\rho \in \text{env} \coloneqq \text{var} \rightharpoonup \mathbb{Z} \qquad\qquad \varsigma \in \Sigma ::= \langle \rho, ce \rangle$$

$$\begin{array}{ll}
⟦\_⟧^a \in aop \to \mathbb{Z} \times \mathbb{Z} \rightharpoonup \mathbb{Z} & \_ \vdash \_ \Downarrow^a \_ \in \mathcal{P}(\text{env} \times \text{aexp} \times \mathbb{Z}) \\
⟦\_⟧^c \in cmp \to \mathbb{Z} \times \mathbb{Z} \to \mathbb{B} & \_ \vdash \_ \Downarrow^b \_ \in \mathcal{P}(\text{env} \times \text{bexp} \times \mathbb{B}) \\
⟦\_⟧^b \in bop \to \mathbb{B} \times \mathbb{B} \to \mathbb{B} & \_ \mapsto \_ \in \mathcal{P}(\Sigma \times \Sigma)
\end{array}$$

$$\frac{}{\rho \vdash \texttt{RAND} \Downarrow^a i}\ \textsc{ARand} \qquad \frac{\rho \vdash ae_1 \Downarrow^a i_1 \qquad \rho \vdash ae_2 \Downarrow^a i_2}{\rho \vdash ae_1 \oplus ae_2 \Downarrow^a ⟦\oplus⟧^a(i_1, i_2)}\ \textsc{AOp}$$

$$\frac{\rho \vdash ae \Downarrow^a i}{\langle \rho, x \coloneqq ae \rangle \mapsto \langle \rho[x \leftarrow i], \texttt{SKIP} \rangle}\ \textsc{CAssign}$$

$$\frac{\rho \vdash be \Downarrow^b \text{true}}{\langle \rho, \texttt{WHILE}\ be\ \texttt{DO}\ ce \rangle \mapsto \langle \rho, ce; \texttt{WHILE}\ be\ \texttt{DO}\ ce \rangle}\ \textsc{CWhile-T}$$

$$\frac{\rho \vdash be \Downarrow^b \text{false}}{\langle \rho, \texttt{WHILE}\ be\ \texttt{DO}\ ce \rangle \mapsto \langle \rho, \texttt{SKIP} \rangle}\ \textsc{CWhile-F}$$

Figure 2. `WHILE` concrete semantics

The definition of γ^z induces a partial order $\sqsubseteq^z$ on $\mathbb{Z}^\sharp$ which must coincide with the subset judgment $\subseteq$ on $\mathcal{P}(\mathbb{Z})$ through γ^z, i.e., $i_1^\sharp \sqsubseteq^z i_2^\sharp \iff \gamma^z(i_1^\sharp) \subseteq \gamma^z(i_2^\sharp)$:

$$\_ \sqsubseteq^z \_ \in \mathcal{P}(\mathbb{Z}^\sharp \times \mathbb{Z}^\sharp) \qquad \bot \sqsubseteq^z i^\sharp \sqsubseteq^z i^\sharp \sqsubseteq^z \top$$
$$\textbf{NEG} \sqsubseteq^z \textbf{NEGZ}, \textbf{NZER} \quad \textbf{ZER} \sqsubseteq^z \textbf{NEGZ}, \textbf{POSZ} \quad \textbf{POS} \sqsubseteq^z \textbf{NZER}, \textbf{POSZ}$$

The correctness of γ^z is established by a Galois connection between concrete and abstract integers, notated $\mathcal{P}(\mathbb{Z}) \underset{\alpha^z}{\overset{\gamma^z}{\leftrightarrows}} \mathbb{Z}^\sharp$. To form a Galois connection, γ^z must have a *best abstraction* α^z, which acts as an approximate inverse to γ^z:

$$\alpha^z \in \mathcal{P}(\mathbb{Z}) \to \mathbb{Z}^\sharp \qquad\qquad sign \in \mathbb{Z} \to \mathbb{Z}^\sharp$$
$$\alpha^z(I) \coloneqq \bigsqcup_{i \in I}^{z} sign(i) \qquad sign(i) \coloneqq \begin{cases} \textbf{NEG} & \text{if } i < 0 \\ \textbf{ZER} & \text{if } i = 0 \\ \textbf{POS} & \text{if } i > 0 \end{cases}$$

The definition of α^z uses the join operator ($\_ \sqcup^z \_$) for abstract integers, also called the least upper bound, which selects the smallest element that is larger than both operands. For example, $\textbf{NEG} \sqcup^z \textbf{ZER} = \textbf{NEGZ}$ and $\textbf{ZER} \sqcup^z \textbf{NZER} = \top$.

In addition to their mere existence, γ^z and α^z must be sound and complete, which take the form of $extensive^z$ and $reductive^z$ in the classical Galois connection framework:

$$\begin{array}{l}
extensive^z : \forall (I \in \mathcal{P}(\mathbb{Z})).\ I \subseteq \gamma^z(\alpha^z(I)) \\
reductive^z : \forall (i^\sharp \in \mathbb{Z}^\sharp).\ \alpha^z(\gamma^z(i^\sharp)) \sqsubseteq^z i^\sharp
\end{array}$$

$extensive^z$ states that given a set of integers I, its abstraction $\alpha^z(I)$ must represent at least all of the concrete integers originally present in I. $reductive^z$ states that the abstraction function α^z must provide the most precise results possible, where more precise means lower in the lattice.

Abstracting Environments An abstract environment maps variables to abstract integers rather than concrete integers.

$$\rho^\sharp \in \text{env}^\sharp \coloneqq \text{var} \to \mathbb{Z}^\sharp$$

Abstract environments are given meaning through the concretization function γ^r which specifies the set of concrete environments represented by an abstract environment:

$$\gamma^r \in \text{env}^\sharp \to \mathcal{P}(\text{env})$$
$$\gamma^r(\rho^\sharp) \coloneqq \{\rho \mid \forall(x).\ \rho(x) \in \gamma^z(\rho^\sharp(x))\}$$

That is, an abstract environment represents all concrete environments that agree pointwise with *some* represented concrete integer in the codomain.

The order on abstract environments is the standard pointwise ordering and obeys $\rho_1^\sharp \sqsubseteq^r \rho_2^\sharp \iff \gamma^r(\rho_1^\sharp) \subseteq \gamma^r(\rho_2^\sharp)$:

$$\_ \sqsubseteq^r \_ \in \mathcal{P}(\text{env}^\sharp \times \text{env}^\sharp)$$
$$\rho_1^\sharp \sqsubseteq^r \rho_2^\sharp \iff \forall(x).\ \rho_1^\sharp(x) \sqsubseteq^z \rho_2^\sharp(x)$$

The correctness of γ^r is established by a Galois connection between concrete and abstract environments, notated $\mathcal{P}(\text{env}) \overset{\gamma^r}{\underset{\alpha^r}{\leftrightarrows}} \text{env}^\sharp$. The best abstraction α^r w.r.t. γ^r is:

$$\alpha^r \in \mathcal{P}(\text{env}) \to \mathit{env}^\sharp$$
$$\alpha^r(R)(x) \coloneqq \alpha^z(\{\rho(x) \mid \rho \in R\})$$

and γ^r and α^r are sound and complete:

$$\mathit{sound}^r : \forall(R \in \mathcal{P}(\text{env})).\ R \subseteq \gamma^r(\alpha^r(R))$$
$$\mathit{complete}^r : \forall(\rho^\sharp \in \text{env}^\sharp).\ \alpha^r(\gamma^r(\rho^\sharp)) \sqsubseteq \rho^\sharp$$

Abstracting Functions After designing abstractions and Galois connections for $\mathbb{Z}$ and env we can define what it means for $[\![\_]\!]^{\sharp a}$, some abstract denotation for arithmetic operators, to be a sound abstraction of $[\![\_]\!]^a$, its concrete counterpart. This is done through a *best specification* which is generated by the Galois connection for abstract integers. Because the Galois connection is defined for sets of integers $\mathcal{P}(\mathbb{Z})$ we must lift the denotation function to the world of powersets, which we notate with a subscript $_\mathcal{P}$:

$$[\![\_]\!]^a_\mathcal{P} \in \text{aexp} \to \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$$
$$[\![ae]\!]^a_\mathcal{P}(I_1, I_2) \coloneqq \{[\![ae]\!]^a(i_1, i_2) \mid i_1 \in I_1, i_2 \in I_2\}$$

This is the *nonrelational* powerset lifting, and is strictly weaker than the relational lifting in $\mathcal{P}(\mathbb{Z} \times \mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$ which supports more precise abstractions.

The best specification for $[\![\_]\!]^a$ is constructed by transporting $[\![\_]\!]^a_\mathcal{P}$ to a function in $\mathbb{Z}^\sharp \times \mathbb{Z}^\sharp \to \mathbb{Z}^\sharp$ using γ^z and α^z, which we notate with a subscript $_\mathcal{S}$.

$$[\![\_]\!]^a_\mathcal{S} \in \mathbb{Z}^\sharp \times \mathbb{Z}^\sharp \to \mathbb{Z}^\sharp$$
$$[\![ae]\!]^a_\mathcal{S}(i_1^\sharp, i_2^\sharp) \coloneqq \alpha^z([\![ae]\!]^a_\mathcal{P}(\gamma^z(i_1^\sharp), \gamma^z(i_2^\sharp)))$$

Although $[\![\_]\!]^a_\mathcal{S}$ inhabits $\mathbb{Z}^\sharp \times \mathbb{Z}^\sharp \to \mathbb{Z}^\sharp$, it is not implementable as an algorithm; rather, it is a specification for some other function in $\mathbb{Z}^\sharp \times \mathbb{Z}^\sharp \to \mathbb{Z}^\sharp$ that is. A computable abstraction of $[\![\_]\!]^a$ is then some $[\![\_]\!]^{\sharp a}$ such that:

1. $[\![\_]\!]^{\sharp a}$ approximates the best specification $[\![\_]\!]^a_\mathcal{S}$, and
2. $[\![\_]\!]^{\sharp a}$ is an algorithm.

Property (1) captures the soundness of $[\![\_]\!]^a$, and can be stated simply and formally as $[\![\_]\!]^a_\mathcal{S} \sqsubseteq [\![\_]\!]^{\sharp a}$ using the pointwise ordering on functions. Property (2) is more subtle, and is typically "declared" by observation, rather than proved.

There are four equivalent versions of the soundness judgement $[\![ae]\!]^a_\mathcal{S} \sqsubseteq [\![ae]\!]^{\sharp a}$:

$$\forall(i_1^\sharp, i_2^\sharp).\ \alpha^z([\![ae]\!]^a_\mathcal{P}(\gamma^z(i_1^\sharp), \gamma^z(i_2^\sharp))) \sqsubseteq^z [\![ae]\!]^{\sharp a}(i_1^\sharp, i_2^\sharp) \quad (\alpha\gamma)$$
$$\forall(i_1^\sharp, i_2^\sharp).\ [\![ae]\!]^a_\mathcal{P}(\gamma^z(i_1^\sharp), \gamma^z(i_2^\sharp)) \subseteq \gamma^z([\![ae]\!]^{\sharp a}(i_1^\sharp, i_2^\sharp)) \quad (\gamma\gamma)$$
$$\forall(I_1, I_2).\ [\![ae]\!]^a_\mathcal{P}(I_1, I_2) \subseteq \gamma^z([\![ae]\!]^{\sharp a}(\alpha^z(I_1), \alpha^z(I_2))) \quad (\gamma\alpha)$$
$$\forall(I_1, I_2).\ \alpha^z([\![ae]\!]^a_\mathcal{P}(I_1, I_2)) \sqsubseteq [\![ae]\!]^{\sharp a}(\alpha^z(I_1), \alpha^z(I_2)) \quad (\alpha\alpha)$$

These four equivalent soundness judgements are well known from the literature [7], and there are strategic reasons for proceeding with one over another, which we discuss in Sections 2.4 and 2.5. $\alpha\gamma$ arises simply by unfolding the definition of $[\![\_]\!]^a_\mathcal{S}$. The others are derived by applying α^z, γ^z, $\mathit{extensive}^z$ and $\mathit{reductive}^z$ to both sides. We label each property with one of $\alpha\gamma$, $\gamma\gamma$, $\gamma\alpha$ and $\alpha\alpha$, which corresponds to the appearances of α^z and γ^z in order from left to right. Note that in general, these are only provably equivalent if α^z and γ^z can be shown to form a Galois connection.

Abstracting Relations An abstraction for the semantics relation $\_ \vdash \_ \Downarrow^a \_$ follows a similar design to that of $[\![\_]\!]^a$. First the relation must be lifted to the world of powersets:

$$\Downarrow^a_\mathcal{P} [\_] \in \text{aexp} \to \mathcal{P}(\text{env}) \to \mathcal{P}(\mathbb{Z})$$
$$\Downarrow^a_\mathcal{P} [ae](R) \coloneqq \{i \mid \rho \in R, \rho \vdash ae \Downarrow^a i\}$$

Next, a best specification is induced using the Galois connections for environments and integers:

$$\Downarrow^a_\mathcal{S} [\_] \in \text{aexp} \to \text{env}^\sharp \to \mathbb{Z}^\sharp$$
$$\Downarrow^a_\mathcal{S} [ae](\rho^\sharp) \coloneqq \alpha^z(\Downarrow^a_\mathcal{P} [ae](\gamma^r(\rho^\sharp)))$$

An abstraction of the concrete relation $\_ \vdash \_ \Downarrow^a \_$ is then any abstract function $\Downarrow^{\sharp a} [\_]$ such that $\Downarrow^a_\mathcal{S} [\_] \sqsubseteq \Downarrow^{\sharp a} [\_]$, which yields the four following equivalent judgments:

$$\forall(\rho^\sharp).\ \alpha^z(\Downarrow^a_\mathcal{P} [ae](\gamma^r(\rho^\sharp))) \sqsubseteq \Downarrow^{\sharp a} [ae](\rho^\sharp) \quad (\alpha\gamma)$$
$$\forall(\rho^\sharp).\ \Downarrow^a_\mathcal{P} [ae](\gamma^r(\rho^\sharp)) \subseteq \gamma^z(\Downarrow^{\sharp a} [ae](\rho^\sharp)) \quad (\gamma\gamma)$$
$$\forall(R).\ \Downarrow^a_\mathcal{P} [ae](R) \subseteq \gamma^z(\Downarrow^{\sharp a} [ae](\alpha^r(R))) \quad (\gamma\alpha)$$
$$\forall(R).\ \alpha^z(\Downarrow^a_\mathcal{P} [ae](R)) \sqsubseteq \Downarrow^{\sharp a} [ae](\alpha^r(R)) \quad (\alpha\alpha)$$

2.3 Constructivity Issues and γ-only

Nearly every verified abstract interpreter to date uses $\gamma\gamma$ rules for verification because of issues encoding α^z in a proof assistant [16, 22, 25]. The heart of the issue is the choice of model for the classical powersets. Inhabitants of $\mathcal{P}(\mathbb{Z})$ can be modelled in constructive logic as a *characteristic predicate* $\phi \in \mathbb{Z} \to \mathit{prop}$, where set containment $x \in \phi$ is modelled as a proof of the judgment $\phi(x)$. Using this model, the concretization function $\gamma^z \in \mathbb{Z}^\sharp \to (\mathbb{Z} \to \mathit{prop})$ is easily defined as an inductive relation $i \in \gamma(i^\sharp)$, which proof assistants excel at manipulating:

$$\frac{i<0}{i \in \gamma^z(\mathbf{NEG})} \quad \frac{}{0 \in \gamma^z(\mathbf{ZER})} \quad \frac{i>0}{i \in \gamma^z(\mathbf{POS})}$$
$$\frac{i\leq 0}{i \in \gamma^z(\mathbf{NEGZ})} \quad \frac{i \neq 0}{i \in \gamma^z(\mathbf{NZER})} \quad \frac{i\geq 0}{i \in \gamma^z(\mathbf{POSZ})} \quad \frac{}{i \in \gamma^z(\top)}$$

The issue is the definition of $\alpha^z \in (\mathbb{Z} \to prop) \to \mathbb{Z}^\sharp$, which would need to *compute* the join of all elements represented by an arbitrary characteristic predicate $\phi \in \mathbb{Z} \to prop$. This cannot be done in constructive logic without admission of classical axioms, which is unsatisfactory, as discussed by Pichardie [25] and Monniaux [22]. For these reasons, the state of the art in verified abstract interpretation[2, 6, 16, 25] uses the γ-only approach exclusively.

2.4 Calculational Abstract Interpretation

Traditional works on abstract interpretation [7] emphasize the *calculational* approach to designing abstraction functions. The essence of the approach is to begin with the induced specification, which is not implementable as an algorithm, and apply ordered rewrites until an algorithm is revealed, at which point the result sound by construction. For example, calculating an abstraction for the arithmetic semantics relation would proceed as:

$$\alpha^z(\Downarrow^a_{\mathcal{P}}[ae](\gamma^r(\rho^\sharp))) \sqsubseteq \ldots \sqsubseteq F[ae](\rho^\sharp) \triangleq \Downarrow^{\sharp a}[ae](\rho^\sharp)$$

Here, F is the result of calculating over the initial specification. F is observed to be an algorithm, and then declared to be the definition of $\Downarrow^{\sharp a}[ae](\rho^\sharp)$. The calculational approach relies on manipulating α, which is not possible in existing mechanization approach of γ-only.

2.5 Simplified Proofs and α-only

Often times the $\alpha\alpha$ property is the simplest to use for justifying the soundness of an abstract operation. In fact, verification of functions can be *dramatically* simplified by an $\alpha\alpha$-like property. Consider the following soundness property for $[\![\_]\!]^{\sharp a}$, which is inspired by the original $\alpha\alpha$ rule:

$$\forall(i_1,i_2).\ sign([\![ae]\!]^a(i_1,i_2)) \sqsubseteq [\![ae]\!]^{\sharp a}(sign(i_1),sign(i_2))$$

This rule makes no mention of powersets *at all*, or the powerset lifting of $[\![\_]\!]^a$ for that matter.

This approach appears to fall outside the Galois connection discipline; if it could be related back to the original setup, it likely corresponds to the $\alpha\alpha$ rule. Although $\alpha\alpha$ poses challenges in a constructive setting, this rule does not. We see this as simultaneously a limitation of the classical Galois connection framework, and a path forward to a variant of $\alpha\alpha$ definable in a proof assistant without using axioms.

3. Constructive Galois Connections

Before defining constructive Galois connections, we review their classical counterparts for contrast.

Definition 1. *A classical Galois connection on posets A and B, notated* $A \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} B$*, consists of functions,* $\alpha \in A \nearrow B$ *and* $\gamma \in B \nearrow A$*, satisfying:*

1. *extensive* : $\forall(x).\ x \sqsubseteq \gamma(\alpha(x))$ *and*
2. *reductive* : $\forall(y).\ \alpha(\gamma(y)) \sqsubseteq y$.

The symmetry of classical Galois connections is no accident; they are the natural by-product the more general framework of adjunctions instantiated to the category of posets. Monotonicity properties are pervasive throughout the Galois connection framework and we notate the monotonic function space $\_ \nearrow \_$. We omitted the details of monotonicity in Section 2 to simplify the presentation.

On our way to constructive Galois connections we introduce an intermediate structure which we call *Kleisli Galois connections*. Whereas classical Galois connections are adjunctions between posets where adjoint functors α and γ are functions, Kleisli Galois connections are adjunctions between posets where adjoint functors α^k and γ^k are *monadic* functions from the powerset Kleisli category.

3.1 The Powerset "Specification" Monad

The powerset $\mathcal{P}(A)$ captures a *specification effect* and is understood as the set of "specifications over elements in A". $\mathcal{P}(A)$ is modelled as the *antitonic* $A \searrow prop$ in constructive logic, and has monadic structure, as defined through monad operations $return$ and $extend$ (sometimes called $bind$):

$$\begin{aligned}
&return \in \forall(A).\ A \nearrow \mathcal{P}(A)\\
&return(x) \coloneqq \{y \mid y \sqsubseteq x\}\\
&extend \in \forall(A,B).\ (A \nearrow \mathcal{P}(B)) \nearrow (\mathcal{P}(A) \nearrow \mathcal{P}(B))\\
&extend(f)(X) \coloneqq \{y \mid \exists(x \in X).\ y \in f(x)\}
\end{aligned}$$

We notate $extend(f)(X)$ as $f{*}X$ in the tradition of Moggi [21]. Intuitively $return(x)$ gives the singleton set $\{x\}$, however the monotonic ordering on powersets requires that they be *downward closed*, i.e., $x_1 \in X \wedge x_2 \sqsubseteq x_1 \Rightarrow x_2 \in X$. $return(x)$ then is the downward closure of the singleton set $\{x\}$.

The powerset Kleisli category is generated by the powerset monad and has two elements, a unit element $return$, which we have already defined, and a composition operator, which we notate $\diamond$:

$$\begin{aligned}
&\_ \diamond \_ \in \forall(A,B,C).\ (B \nearrow \mathcal{P}(C)) \nearrow (A \nearrow \mathcal{P}(B)) \nearrow (A \nearrow \mathcal{P}(C))\\
&(g \diamond f)(x) \coloneqq g * f(x)
\end{aligned}$$

Fact 1 (Powerset Monad Laws).✓ *The powerset operator* $\mathcal{P}(\_)$ *and functions* $return$ *and* $extend$ *obey monad laws.*

$$\begin{aligned}
&\textit{left-unit} : \forall(X).\ return * X = X\\
&\textit{right-unit} : \forall(f,x).\ f * return(x) = f(x)\\
&\textit{associative} : \forall(g,f,X).\ g * (f * X) = (g \diamond f) * X
\end{aligned}$$

3.2 Kleisli Galois Connections

Using the powerset monad we re-instantiate the adjunction framework that generated classical Galois connections to a new setting where the adjoint functors α^k and γ^k are members of the powerset Kleisli category. Using classical Galois connections, α and γ live in the standard function space $\_ \nearrow \_$, and the laws *extensive* and *reductive* come from the unit and composition operators for functions: the identify function and standard function composition:

$$\begin{aligned} &extensive : id \sqsubseteq \gamma \circ \alpha \\ &reductive : \alpha \circ \gamma \sqsubseteq id \end{aligned}$$

Kleisli Galois connections differ only in that α^k and γ^k live in the *monadic* function space of the powerset monad $\_ \nearrow \mathcal{P}(\_)$. The laws $extensive^k$ and $reductive^k$ are then analogous to that of classical Galois connections, but with $return$ and $\_ \diamond \_$ substituted for id and $\_ \circ \_$ as the unit and composition operators:

$$\begin{aligned} &extensive^k : return \sqsubseteq \gamma \diamond \alpha \\ &reductive^k : \alpha \diamond \gamma \sqsubseteq return \end{aligned}$$

Unfolding $extensive^k$ and $reductive^k$, we arrive at the definition for Kleisli Galois connections:

Definition 2. *A Kleisli Galois connection on posets A and B, notated $A \underset{\alpha^k}{\overset{\gamma^k}{\leftrightarrows}} B$, consists of functions, $\alpha^k \in A \nearrow \mathcal{P}(B)$ and $\gamma \in B \nearrow \mathcal{P}(A)$, satisfying:*

1. $extensive^k : \forall(x).\ return(x) \subseteq \gamma^k * \alpha^k(x)$ *and*
2. $reductive^k : \forall(y).\ \alpha^k * \gamma^k(y) \subseteq return(y)$.

The use of $return$ as the unit of the powerset Kleisli category is significant: $extensive$ says that $\gamma^k \diamond \alpha^k$ is a pure function *at worst*, and $reductive$ says that $\alpha^k \diamond \gamma^k$ is a pure function *at best*, where pure in this context means "has no specification effect" and is introduced by $return$.

Unfolding the definitions of $return$ and $*$ we arrive at definitions for $sound^k$ and $complete^k$ which are equivalent to $extensive^k$ and $reductive^k$:

$$\begin{aligned} &sound^k : \forall(x).\ \exists(y).\ y \in \alpha^k(x) \wedge x \in \gamma^k(y) \\ &complete^k : \forall(x, y_1, y_2).\ x \in \gamma^k(y_1) \wedge y_2 \in \alpha^k(x) \Rightarrow y_2 \sqsubseteq y_1 \end{aligned}$$

We can already see a more intuitive and constructive setup for abstraction emerging from Kleisli Galois connections. α^k and γ^k are honest about *both* being specifications, which we model as returning powerset elements which have monadic structure. Also $sound^k$ and $complete^k$ give much more intuitive readings: $sound^k$ says "any element in A can be abstracted, and is contained in the meaning of its abstraction". $complete^k$ says "if we abstract any element in the meaning of y_1, it must contain equal or more information than y_1", or in other words, α^k is prohibited from losing any information, which would result in the strict inequality $y_1 \sqsubset y_2$.

3.3 Constructive Galois Connections

Constructive Galois connections arise from the following insight: *The specification effect on α^k is (constructively) provably benign in every case.* That is, there always exists some

Classical

$\alpha : A \nearrow B$

$\gamma : B \nearrow A$

$extensive : \forall(x).\ x \sqsubseteq \gamma(\alpha(x))$

$reductive : \forall(y).\ \alpha(\gamma(y)) \sqsubseteq y$

Kleisli

$\alpha^k : A \nearrow \mathcal{P}(B)$

$\gamma^k : B \nearrow \mathcal{P}(A)$

$extensive^k : \forall(x).\ return(x) \sqsubseteq \gamma^k * \alpha^k(x)$

$reductive^k : \forall(y).\ \alpha^k * \gamma^k(y) \sqsubseteq return(y)$

$sound^k : \forall(x).\ x \in \gamma^k * \alpha^k(x)$

$complete^k : \forall(x, y_1, y_2).\ x \in \gamma^k(y_1) \wedge y_2 \in \alpha^k(x) \Rightarrow y_2 \sqsubseteq y_1$

Constructive

$\eta : A \nearrow B$

$\gamma^c : B \nearrow \mathcal{P}(A)$

$extensive^c : \forall(x).\ return(x) \sqsubseteq \gamma^c * pure(\eta)(x)$

$reductive^c : \forall(y).\ pure(\eta) * \gamma^c(y) \sqsubseteq return(y)$

$sound^c : \forall(x).\ x \in \gamma^c(\eta(x))$

$complete^c : \forall(x, y).\ x \in \gamma^c(y) \Rightarrow \eta(x) \sqsubseteq y$

Figure 3. Classical, Kleisli, and constructive GCs

$\eta \in A \nearrow B$ s.t. $\alpha^k(x) = return(\eta(x))$. This fact follows from $sound^k$ and $complete^k$, the laws that accompany α^k and γ^k to form a Kleisli Galois connection. We explore the proof of *constructing* η from α^k and the isomorphism between Kleisli and constructive Galois connections in Section 6.

For now we assume η is given *a priori*. We rename α to η in the constructive setting following Nielson, Nielson, and Hankin [23] where pure abstraction functions are called "extraction functions" and notated η. In the following definition we use $pure(f)$ where $pure(f)(x) := return(f(x))$:

Definition 3. *A constructive Galois connection on posets A and B, notated $A \underset{\eta}{\overset{\gamma^c}{\leftrightarrows}} B$, consists of functions, $\eta \in A \nearrow B$ and $\gamma^c \in B \nearrow \mathcal{P}(A)$, satisfying:*

1. $extensive^c : \forall(x).\ return(x) \subseteq \gamma^c * return(\eta(x))$ *and*
2. $reductive^c : \forall(y).\ pure(\eta) * \gamma^c(y) \subseteq return(y)$.

This definition is identical to that for Kleisli, except α^k has been replace with $pure(\eta)$. The extensive and reductive laws are the same. The benefit of the constructive setup is twofold: the abstraction function η has no specification effect, and the soundness and completeness properties that follow from $extensive^c$, $reductive^c$ and monad unit laws are much simpler than their Kleisli counterparts:

$$\begin{aligned} &sound^c : \forall(x).\ x \in \gamma^c(\eta(x)) \\ &complete^c : \forall(x, y).\ x \in \gamma^c(y) \Rightarrow \eta(x) \sqsubseteq y \end{aligned}$$

The simplicity of these rule is striking. $sound^c$ says: "given an individual element of A, that element is contained in the meaning of its abstraction." $complete^c$ says "considering any

Constructive Galois Connection for *Integers*

$$\eta^z \in \mathbb{Z} \nearrow \mathbb{Z}^\sharp \qquad \eta^z(i) := \begin{cases} \mathbf{NEG} & \textit{if } i < 0 \\ \mathbf{ZER} & \textit{if } i = 0 \\ \mathbf{POS} & \textit{if } i > 0 \end{cases}$$

$\gamma^{cz} \in \mathbb{Z}^\sharp \nearrow \mathcal{P}(\mathbb{Z})$

$\gamma^{cz}(\bot) := \{\} \qquad \gamma^{cz}(\mathbf{NEGZ}) := \{i \mid i \leq 0\}$

$\gamma^{cz}(\mathbf{NEG}) := \{i \mid i < 0\} \qquad \gamma^{cz}(\mathbf{NZER}) := \{i \mid i \neq 0\}$

$\gamma^{cz}(\mathbf{ZER}) := \{0\} \qquad \gamma^{cz}(\mathbf{POSZ}) := \{i \mid i \geq 0\}$

$\gamma^{cz}(\mathbf{POS}) := \{i \mid i > 0\} \qquad \gamma^{cz}(\top) := \{i \mid i \in \mathbb{Z}\}$

Constructive Galois Connection for *Environments*

$\eta^r \in \text{env} \nearrow \text{env}^\sharp \qquad \eta^r(\rho)(x) := \eta^z(\rho(x))$

$\gamma^{cr} \in \text{env}^\sharp \nearrow \mathcal{P}(\text{env})$

$\gamma^{cr}(\rho^\sharp) := \{\rho \mid \forall(x).\ \eta^z(\rho(x)) \sqsubseteq \rho^\sharp(x)\}$

Soundness Conditions

$\forall(i_1, i_2).\ \eta^z([\![ae]\!]^a(i_1, i_2)) \sqsubseteq [\![ae]\!]^{\sharp a}(\eta^z(i_1), \eta^z(i_2))$

$\forall(\rho).\ pure(\eta^z) * \Downarrow^a [ae](\rho) \sqsubseteq return(\Downarrow^{\sharp a} [ae](\eta^r(\rho)))$

Figure 4. Abstraction design with constructive GCs

abstractions of an element in A, η must give the best abstraction." To easily compare each Galois connection setup, we display each formulation side-by-side in Figure 3.

Because constructive Galois connections are much simpler than classical Galois connections, one might wonder about their relationship with each other in both directions:

1. Which classical Galois connections can be encoded as constructive Galois connections?
2. Can constructive Galois connections be used to recover results stated using classical Galois connections?

We give precise answers to these questions in Section 6.

3.4 Applying Constructive Galois Connections

The soundness conditions for an abstract interpreter developed in Section 2 using classical Galois connections suffered from two drawbacks: only the $\gamma\gamma$ rule could be encoded in constructive logic without axioms, and they could not account for a simplified verification rule inspired by $\alpha\alpha$.

We present the constructive Galois connections for integers and environments $\mathbb{Z} \underset{\eta^z}{\overset{\gamma^{cz}}{\leftrightarrows}} \mathbb{Z}^\sharp$ and env $\underset{\eta^r}{\overset{\gamma^{cr}}{\leftrightarrows}}$ env$^\sharp$, along with the generated $\eta\eta$ soundness rules in Figure 4. In particular, the η definitions are much simpler than their classical α counterparts, and can be defined without classical axioms.

We present definitions for Cousot's abstract interpreter for the arithmetic portion of `WHILE` in Figure 5 along with its soundness proof using the $\eta\eta$ soundness judgments[3]. The simplicity of the soundness proof is striking: in most cases the proof is trivial or follows immediately from the inductive hypothesis, which is not the case for either of the constructive $\eta\gamma$, $\gamma\gamma$ and $\gamma\eta$, or any of the classical soundness setups.

In general, all soundness results using constructive Galois connections can be lifted to their classical counterparts, and

[3] See (redacted for double-blind) for entire mechanization of `WHILE`.

Abstract Interpreter

$\Downarrow^{\sharp a} [i](\rho^\sharp) := \eta^z(i)$

$\Downarrow^{\sharp a} [x](\rho^\sharp) := \rho^\sharp(x)$

$\Downarrow^{\sharp a} [\texttt{RAND}](\rho^\sharp) := \top$

$\Downarrow^{\sharp a} [ae_1 \oplus ae_2](\rho^\sharp) := [\![\oplus]\!]^{\sharp a}(\Downarrow^{\sharp a} [ae_1](\rho^\sharp), \Downarrow^{\sharp a} [ae_2](\rho^\sharp))$

Proof of Soundness

$sound[\Downarrow^{\sharp a}] : pure(\eta^z) * \Downarrow^a [ae](\rho) \sqsubseteq return(\Downarrow^{\sharp a} [ae](\eta^z(\rho)))$

CASE $ae = i$

$goal : \{\eta^z(i') \mid \rho \vdash i \Downarrow^a i'\} \sqsubseteq \{\eta^z(i)\}$

follows by definition of $\Downarrow^a [i]$

CASE $ae = x$

$goal : \{\eta^z(i) \mid \rho \vdash x \Downarrow^a i\} \sqsubseteq \{\eta^r(\rho)(x)\}$

follows by definition of $\Downarrow^a [x]$ and $\eta^r(\rho)$

CASE $ae = \texttt{RAND}$

$goal : \forall(i).\ \eta(i) \sqsubseteq \top$

follows from $\top$ element of lattice

CASE $ae = ae_1 \oplus ae_2$

$IH_n : \eta^z(\Downarrow^a [ae_n](\rho)) \sqsubseteq \Downarrow^{\sharp a} [ae_n](\eta^r(\rho))$

$sound[[\![\oplus]\!]^a] : \forall(i_1,i_2).\ \eta^z([\![\oplus]\!]^a(i_1,i_2)) \sqsubseteq [\![\oplus]\!]^{\sharp a}(\eta^z(i_1),\eta^z(i_2))$

$goal : \eta^z([\![\oplus]\!]^a(\Downarrow^a [ae_1](\rho), \Downarrow^a [ae_2](\rho)))$
$\qquad \sqsubseteq [\![\oplus]\!]^{\sharp a}(\Downarrow^{\sharp a} [ae_1](\eta^r(\rho)), \Downarrow^{\sharp a} [ae_2](\eta^r(\rho)))$

transitive reasoning as follows. . .

$\eta^z([\![\oplus]\!]^a(\Downarrow^a [ae_1](\rho), \Downarrow^a [ae_2](\rho)))$

$\sqsubseteq [\![\oplus]\!]^{\sharp a}(\eta^z(\Downarrow^a [ae_1](\rho)), \eta^z(\Downarrow^a [ae_2](\rho))) \ \wr sound[[\![\oplus]\!]^a] \int$

$\sqsubseteq [\![\oplus]\!]^{\sharp a}(\Downarrow^{\sharp a} [ae_1](\eta^r(\rho)), \Downarrow^{\sharp a} [ae_2](\eta^r(\rho))) \ \wr IH_{1,2} \int \ \blacksquare$

Figure 5. Definitions and soundness proofs using $\eta\eta$

we present a full proof of this fact in Section 6. In this example, the result is the automatic recovery of the classical $\gamma\gamma$ judgment which is used extensively in the state of the art in mechanically verified abstract interpreters, or a version of the $\alpha\alpha$ judgment which does not require classical axioms.

4. Case Study: Calculational AI

To demonstrate the utility and generality of constructive Galois connections we present the first verified abstract interpreter derived through *calculation*, as well as the accompanying proof library for general purpose calculation using Galois connections, which is also a first of its kind.

The setup of calculational abstract interpretation is similar to the design we present in this paper. However, rather than posit an abstract semantics $\Downarrow^{\sharp a} [\_]$ and verify it *a posteriori*, we calculate it from its induced specification.

The best specification for some abstract interpreter $\Downarrow^{\sharp a} [\_]$ was codified by the classical $\alpha\gamma$ soundness rule:

$$\forall(\rho^\sharp).\ \alpha^z(\Downarrow^a_{\mathcal{P}} [ae](\gamma^r(\rho^\sharp))) \sqsubseteq \Downarrow^{\sharp a} [ae](\rho^\sharp) \qquad (\alpha\gamma)$$

The goal is then to *calculate* over the specification using rewrites to obtain a definition for $\Downarrow^{\sharp a} [\_]$. We present selected cases of Cousot's calculation, along with their constructive counterparts in Figure 6.

***Classical* Numeric Literals**

$$\begin{aligned}
&\alpha^z(\Downarrow^a_{\mathcal{P}}[i](\gamma^r(\rho^\sharp))) && \\
&= \alpha^z(\{i' \mid \rho \in \gamma^r(\rho^\sharp) \wedge \rho \vdash i \Downarrow^a i'\}) && \wr \text{ definition of } \Downarrow^a_{\mathcal{P}}[i] \int \\
&= \alpha^z(\{i\}) && \wr \text{ definition of } \rho \vdash i \Downarrow^a i' \int \\
&\triangleq \Downarrow^{\sharp a}[i](\rho^\sharp) && \wr \Downarrow^{\sharp a}[i](\rho^\sharp) \coloneqq \alpha^z(\{i\}) \int
\end{aligned}$$

***Constructive* Numeric Literals**

$$\begin{aligned}
&pure(\eta^z) * (\Downarrow^a [i] * \gamma^r(\rho^\sharp)) && \\
&= pure(\eta^z) * && \\
&\quad (\{i' \mid \rho \in \gamma^r(\rho^\sharp) \wedge \rho \vdash i \Downarrow^a i'\}) && \wr \text{ definition of } \Downarrow^a [i]* \int \\
&= pure(\eta^z) * return(i) && \wr \text{ definition of } \rho \vdash i \Downarrow^a i' \int \\
&= return(\eta^z(i)) && \wr \text{ monad right unit} \int \\
&\triangleq return(\Downarrow^{\sharp a}[i](\rho^\sharp)) && \wr \Downarrow^{\sharp a}[i](\rho^\sharp) \coloneqq \eta^z(i) \int
\end{aligned}$$

***Classical* Variable Reference**

$$\begin{aligned}
&\alpha^z(\Downarrow^a_{\mathcal{P}}[x](\gamma^r(\rho^\sharp))) && \\
&= \alpha^z(\{i \mid \rho \in \gamma^r(\rho^\sharp) \wedge \rho \vdash x \Downarrow^a i\}) && \wr \text{ definition of } \Downarrow^a_{\mathcal{P}}[x] \int \\
&= \alpha^z(\{\rho(x) \mid \rho \in \gamma^r(\rho^\sharp)\}) && \wr \text{ definition of } \rho \vdash x \Downarrow^a i \int \\
&= \alpha^z(\gamma^z(\rho^\sharp(x))) && \wr \text{ definition of } \gamma^r \int \\
&\sqsubseteq \rho^\sharp(x) && \wr\ \alpha^z \circ \gamma^z \text{ reductive} \int \\
&\triangleq \Downarrow^{\sharp a}[x](\rho^\sharp) && \wr \Downarrow^{\sharp a}[x](\rho^\sharp) \coloneqq \rho^\sharp(x) \int
\end{aligned}$$

***Constructive* Variable Reference**

$$\begin{aligned}
&pure(\eta^z) * (\Downarrow^a [x] * \gamma^r(\rho^\sharp)) && \\
&= pure(\eta^z) * && \\
&\quad (\{i \mid \rho \in \gamma^r(\rho^\sharp) \wedge \rho \vdash x \Downarrow^a i\}) && \wr \text{ definition of } \Downarrow^a[x]* \int \\
&= pure(\eta^z) * (\{\rho(x) \mid \rho \in \gamma^r(\rho^\sharp)\}) && \wr \text{ definition of } \rho \vdash x \Downarrow^a i \int \\
&= pure(\eta^z) * \gamma^z(\rho^\sharp(x)) && \wr \text{ definition of } \gamma^r \int \\
&\sqsubseteq return(\rho^\sharp(x)) && \wr\ \eta^z \diamond \gamma^z \text{ reductive} \int \\
&\triangleq return(\Downarrow^{\sharp a}[x](\rho^\sharp)) && \wr \Downarrow^{\sharp a}[x](\rho^\sharp) \coloneqq \rho^\sharp(x) \int
\end{aligned}$$

Figure 6. Case study: classical vs constructive calculation

In both classical calculations, the search is for some definition of $\Downarrow^{\sharp a}[\_]$ through calculation which is an algorithm "by observation". Seen through the lens of a proof assistant, this process can also be viewed as moving from a specification which relies on classical axioms to one which does not. However, the constructive calculations distinguish between specification and algorithm explicitly through the presence of the monadic specification effect $\mathcal{P}(\_)$, whereas the use of classical axioms is implicit in the classical calculation.

In the constructive case, $\Downarrow^{\sharp a}[\_]$ is extracted from its monadic lifting $pure(\Downarrow^{\sharp a}[\_])$, which is derived from the specification induced by constructive Galois connections:

$$pure(\eta^z) * (\Downarrow^a [ae] * \gamma^r(\rho^\sharp)) \sqsubseteq pure(\Downarrow^{\sharp a}[ae])(\rho^\sharp)$$

Calculation proceeds until some expression $return(\ldots)$, which is a pure computation. In the classical setup, the framework guarantees a sound result by construction, but computability is justified externally from the framework. In the constructive setting, the result is both sound *and computable* by construction, because we account for nonconstructivity explicitly with a specification effect which is eliminated during calculation.

***Precise* Simple Type System** | ***Gradual* Simple Type System**

$$\begin{aligned}
&\tau \in type \Coloneqq \bot \mid \mathbb{B} \mid \tau \to \tau && \tau^\sharp \in type^\sharp \Coloneqq \bot \mid \mathbb{B} \mid \tau^\sharp \to \tau^\sharp \mid ? \\
&dom \in type \nearrow type && dom^\sharp \in type^\sharp \nearrow type^\sharp \\
&cod \in type \nearrow type && cod^\sharp \in type^\sharp \nearrow type^\sharp \\
&\_ \sqcap \_ \in type \times type \nearrow type && \_ \sqcap^\sharp \_ \in type^\sharp \times type^\sharp \nearrow type^\sharp \\
&\_ \equiv \_ \in type \times type \searrow prop && \_ \sim \_ \in type^\sharp \times type^\sharp \searrow prop
\end{aligned}$$

$$\frac{\Gamma \vdash e_1 : \mathbb{B} \quad \Gamma \vdash e_2 : \tau_1 \quad \Gamma \vdash e_3 : \tau_2}{\Gamma \vdash \texttt{IF}(e_1)\{e_2\}\{e_3\} : \tau_1 \sqcap \tau_2}\ \textsc{If} \qquad \frac{\Gamma^\sharp \vdash e_1 : \mathbb{B} \quad \Gamma^\sharp \vdash e_2 : \tau_1^\sharp \quad \Gamma^\sharp \vdash e_3 : \tau_2^\sharp}{\Gamma \vdash \texttt{IF}(e_1)\{e_2\}\{e_3\} : \tau_1^\sharp \sqcap^\sharp \tau_2^\sharp}\ \textsc{If}^\sharp$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_2 \equiv dom(\tau_1)}{\Gamma \vdash e_1 e_2 : cod(\tau_1)}\ \textsc{Apply} \qquad \frac{\Gamma^\sharp \vdash e_1 : \tau_1^\sharp \quad \Gamma^\sharp \vdash e_2 : \tau_2^\sharp \quad \tau_2^\sharp \sim dom^\sharp(\tau_1^\sharp)}{\Gamma \vdash e_1 e_2 : cod^\sharp(\tau_1^\sharp)}\ \textsc{Apply}^\sharp$$

$$\frac{\Gamma \vdash e : \tau_1 \quad \tau_1 \equiv \tau_2}{\Gamma \vdash (e :: \tau_2^\sharp) : \tau_2^\sharp}\ \textsc{Coerce} \qquad \frac{\Gamma^\sharp \vdash e : \tau_1^\sharp \quad \tau_1^\sharp \sim \tau_2^\sharp}{\Gamma^\sharp \vdash (e :: \tau_2^\sharp) : \tau_2^\sharp}\ \textsc{Coerce}^\sharp$$

$$\begin{aligned}
&\text{correct}[type^\sharp] : type \underset{\eta}{\overset{\gamma}{\leftrightarrows}} type^\sharp \\
&\text{correct}[dom^\sharp] : \forall(\tau).\ \eta(dom(\tau)) = dom^\sharp(\eta(\tau)) \\
&\text{correct}[cod^\sharp] : \forall(\tau).\ \eta(cod(\tau)) = cod^\sharp(\eta(\tau)) \\
&\text{correct}[\sqcap^\sharp] : \forall(\tau_1,\tau_2).\ \eta(\tau_1 \sqcap \tau_2) = \eta(\tau_1) \sqcap^\sharp \eta(\tau_2) \\
&\text{correct}[\sim] : \forall(\tau_1^\sharp,\tau_2^\sharp).\ \tau_1 \sim \tau_2 \leftrightarrow \exists(\tau_1 \in \gamma(\tau_1^\sharp), \tau_2 \in \gamma(\tau_2^\sharp)).\ \tau_1 \equiv \tau_2
\end{aligned}$$

Figure 7. Case study: deriving gradual type systems

Our Agda calculation strongly resembles the on-paper math. For example, the code for abstracting variable references is:

```
$\alpha[\Downarrow^a]$ (Var $x$) $\rho^\sharp$  =  [proof-mode]
  do [[ (pure · $\eta^z$) *· ($\Downarrow^a$ [ Var $x$ ] *· ($\gamma^{cr}$ · $\rho^\sharp$)) ]]
  ▸ [focus-right [·] $of$ (pure · $\eta^z$) * ] begin
    do [[ $\Downarrow^a$ [ Var $x$ ] *· ($\gamma^{cr}$ · $\rho^\sharp$) ]]
    ▸ $\wr$ $\beta \Downarrow^a$ [Var] $\int$
    ▸ [[ (pure · lookup[ $x$ ]) *· ($\gamma^{cr}$ · $\rho^\sharp$) ]]
    ▸ $\wr$ lookup $\diamond$ $\gamma^{cr} \sqsubseteq \gamma^{cz}$ $\diamond$ lookup$^\sharp$ $\wr$
    ▸ [[ $\gamma^{cz}$ *· (pure · lookup$^\sharp$[ $x$ ] · $\rho^\sharp$) ]]
    end
  ▸ [[ (pure · $\eta^z$) *· ($\gamma^{cz}$ *· (pure · lookup$^\sharp$[ $x$ ] · $\rho^\sharp$)) ]]
  ▸ $\wr$ reductive[$\eta \diamond \gamma$] $\int$
  ▸ [[ return · (lookup$^\sharp$[ $x$ ] · $\rho^\sharp$) ]]
  ▸ [[ return · ($\Downarrow^{\sharp a}$ [ Num $n$ ] · $\rho^\sharp$) ]] ■
```

We complete the full calculation of Cousot's generic abstract interpreter for `WHILE` in Agda, where the resulting interpreter is both sound and computable by construction. We also present a comprehensive calculational framework for arbitrary calculational reasoning using Galois connections as a proof library in Agda. Both artifacts are available as supplemental material to this paper.

5. Case Study: Gradual Type Systems

Recent work in metatheory for gradual type systems by Garcia et al. [15] demonstrates that a Galois connection discipline can guide the design of gradual typing systems. Starting with a Galois connection between precise and gradual types, both the static and dynamic semantics of the gradual type system can be derived systematically.

The essence of the design presented by Garcia et al is to begin with a precise type system, like the simply typed lambda calculus, and add a new type $?$ which functions as the $\top$ element in the lattice of type precision. The precise typing rules are presented with four meta-operators dom, cod, $equate$ and $\equiv$. The gradual type system is then written using abstract versions $dom^\sharp$, $cod^\sharp$, $\sqcap$ and $\sim$, which are proven correct w.r.t. their induced specifications. These examples from the case study are shown in Figure 7.

Because we use the constructive Galois connection framework, our η is much simpler than the α mapping originally presented, and likewise our proof setup is simplified as well. We verify the definitions of $dom^\sharp$, $cod^\sharp$ and $\sqcap^\sharp$ using both the proof originally presented in the paper, as well with the simpler $\eta\eta$ soundness rule. Using the constructive Galois connection setup, η could not be any simpler: it is the identify function embedding into a larger set. The monotonicity of these functions and the existence of a constructive Galois connection $type \underset{\eta}{\overset{\gamma^c}{\leftrightarrows}} type^\sharp$ is all that is needed to recover the soundness proofs originally presented in their paper, which are comparatively more complicated.

For example, in their paper, they calculate the definition of $cod^\sharp(?)$ through the following sequence of reasoning [15, Section 4.4] (translated to our notation):

$$\alpha(cod_{\mathcal{P}}(\gamma(?))) = \alpha(cod_{\mathcal{P}}(type)) = \alpha(type) = ? \triangleq cod^\sharp(?)$$

Using the $\eta\eta$ rule, we need only verify $cod^\sharp(\tau_1 \to \tau_2) = \tau_2$; the correctness of $cod^\sharp(?) = ?$ is then automatic from the monotonicity of cod and Galois connection between η and γ. Our proof correct$[cod^\sharp]$ in Agda couldn't possibly be simpler:

$$\begin{aligned}
&\text{correct}[cod^\sharp]/\eta\eta \;:\; \forall\ (\tau \;:\; type) \to \eta^t\cdot(cod\cdot\tau) \;\equiv\; cod^\sharp\cdot(\eta^t\cdot\tau)\\
&\text{correct}[cod^\sharp]/\eta\eta\ \ \bot \;=\; \text{refl}\\
&\text{correct}[cod^\sharp]/\eta\eta\ \ \langle\mathbb{B}\rangle \;=\; \text{refl}\\
&\text{correct}[cod^\sharp]/\eta\eta\ \ (\tau_1\ \langle\to\rangle\ \tau_2) \;=\; \text{refl}
\end{aligned}$$

Only the cases for precise types need be considered, and each proof follows immediately by reflexivity. No set-theoretic reasoning like $cod_{\mathcal{P}}(type) = type$ is required, and if we are interested in such facts, we can automatically lift correct$[cod^\sharp]$ to classical results to recover them.

In contrast, our proof wich is faithful to the one presented in Garcia et al uses the induced best specification ($\eta\gamma$) and is more involved; it must justify both directions of the equality separately as subset judgements between powersets, one direction of which we show here:

$$\begin{aligned}
&\text{correct}[cod^\sharp]/\eta\gamma : \forall\ \tau^\sharp \to (\text{pure}\cdot\eta^t)* \cdot((\text{pure}\cdot cod)* \cdot(\gamma^{ct}\cdot\tau^\sharp))\\
&\qquad\qquad\qquad\qquad \sqsubseteq \text{pure}\cdot cod^\sharp\cdot\tau^\sharp\\
&\text{correct}[cod^\sharp]/\eta\gamma\ \ \tau^\sharp \;=\; \text{extensionality}[\mathcal{P}]\ \ (Q\ \ \tau^\sharp) \quad \text{where}\\
&\quad Q \;:\; \forall\ \tau_1^\sharp\ \tau_2^\sharp \to \tau_2^\sharp \in (\text{pure}\cdot\eta^t)* \cdot((\text{pure}\cdot cod)* \cdot(\gamma^{ct}\cdot\tau_1^\sharp))\\
&\qquad\qquad \to \tau_2^\sharp \in \text{pure}\cdot cod^\sharp\cdot\tau_1^\sharp\\
&\quad Q\ \_\ .\bot\ (\exists\mathcal{P}\ .\bot\ ,\ (\exists\mathcal{P}\ .\bot\ ,\ \bot\ ,\ \bot)\ ,\ \bot) \;=\; \bot\\
&\quad Q\ .\top\ \_\ (\exists\mathcal{P}\ \_\ ,\ (\exists\mathcal{P}\ \_\ ,\ \top\ ,\ \_)\ ,\ \_) \;=\; \top\\
&\quad Q\ .\langle\mathbb{B}\rangle\ .\bot\ (\exists\mathcal{P}\ .\bot\ ,\ (\exists\mathcal{P}\ .\langle\mathbb{B}\rangle\ ,\ \langle\mathbb{B}\rangle\ ,\ \bot)\ ,\ \bot) \;=\; \bot\\
&\quad Q\ (\_\ \langle\to\rangle\ \tau_2^\sharp)\ \tau_1^\sharp\ (\exists\mathcal{P}\ \tau_1\ ,\ (\exists\mathcal{P}\ (\_\ \langle\to\rangle\ \tau_2)\\
&\qquad ,\ (\_\ \langle\to\rangle\ \tau_2{\in}\gamma[\tau_2^\sharp])\ ,\ \tau_1{\sqsubseteq}\tau_2)\ ,\ \tau_1^\sharp{\sqsubseteq}\eta[\tau_1]) \;=\\
&\qquad \text{complete}^{ct}\ \ \tau_2{\in}\gamma[\tau_2^\sharp] \circledcirc \text{respectful-arg}\ \tau_1{\sqsubseteq}\tau_2 \circledcirc \tau_1^\sharp{\sqsubseteq}\eta[\tau_1]
\end{aligned}$$

The proof considers some element $\tau_2^\sharp$ in $pure(\eta^t)*(pure(cod)*(\gamma^{ct}(\tau_1^\sharp)))$, for which there exists an element τ_1 in $pure(cod)*(\gamma^{ct}(\tau_1^\sharp))$ and some τ_2 in $\gamma^{ct}(\tau_1^\sharp)$, and must show that $\tau_2^\sharp$ is also in $return(cod^\sharp(\tau_1^\sharp))$, i.e. $\tau_2^\sharp \sqsubseteq cod^\sharp(\tau_1^\sharp)$. The proof proceeds by induction on the judgment $\tau_2 \in \gamma^{ct}(\tau_1^\sharp)$ which has four cases: $\tau^\sharp = \top$, $\bot$, $\mathbb{B}$ or $\tau_{11}^\sharp \to \tau_{21}^\sharp$. The first three cases are trivial resulting in either $\top$ or $\bot$, and the $\to$ case uses the proofs of containment for τ_1 and τ_2, as well as the completeness property between η^t and γ^{ct}.

In the end, the $\eta\gamma$ proof is much more involved than the simpler $\eta\eta$ proof we showed prior, both of which are equivalent and generated by the constructive Galois connection. We observe this is true for $\eta\gamma$ and $\eta\eta$ rules in general.

We have mechanized all definitions and theorems for the static semantics defined in their work, demonstrating both the validity of their approach, and the relative simplicity of the correctness proofs in the context of constructive Galois connections. We do many of the proofs twice, both in the style as presented as well as using the $\eta\eta$ soundness rules. The $\eta\eta$ rules were much simpler to prove in every case. As we translated their results into verified Agda definitions, we likewise translated classical Galois connections to constructive ones. We noticed a similar property from our case study mechanizing Cousot's calculated generic abstract interpreter: all of the Galois connections used were computational in nature, and all classical results could be recovered through lifted constructive results.

6. Constructive GC Foundations

In this section we consider the space of constructive Galois connections, and make precise their relationship to classical Galois connections. We do this in two parts, as shown in Figure 8: an isomorphism between Kleisli Galois connections and the subset of classical Galois connections with computational content, and an isomorphism between constructive and Kleisli Galois connections using the constructive embedding of the axiom of choice. We have verified all of the metatheory presented in this section in Agda, including the

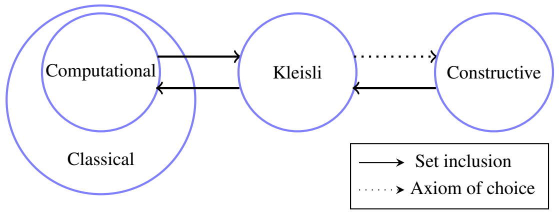

Figure 8. Relationship between classical, Kleisli and constructive Galois connections.

mapping from Kleisli to constructive, which *computes* a constructive Galois connection from a Kleisli one.

Relating Kleisli to classical Galois connections Kleisli Galois connections can be lifted to classical Galois connections through the monadic extension operator. If α^k and γ^k form a Kleisli Galois connection, then $\alpha^k{*}$ and $\gamma^k{*}$ necessarily form a classical Galois connections. In addition to Galois connections themselves, soundness results established w.r.t. Kleisli Galois connections can be lifted to classical results stated with $\alpha^k{*}$ and $\gamma^k{*}$.

Theorem 1 (Kleisli Soundness).✓ *For every Kleisli Galois connection $A \underset{\alpha^k}{\overset{\gamma^k}{\leftrightarrows}} B$ there exists a classical Galois connection $\mathcal{P}(A) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{P}(B)$ where $\alpha \coloneqq \alpha^k{*}$ and $\gamma \coloneqq \gamma^k{*}$, and for every Kleisli proof of abstraction $\gamma \diamond F \diamond \alpha \sqsubseteq F^\sharp$ there exists a classical proof of abstraction $\gamma{*} \circ F{*} \circ \alpha{*} \sqsubseteq F^\sharp{*}$.*

Finally, we identify a subset of classical Galois connections which can be *lowered* to Kleisli Galois connections, which we call *computational Galois connections*. Computational Galois connections are classical Galois connections where α and γ are of the form $\alpha^k{*}$ and $\gamma^k{*}$ for some α^k and γ^k. That is, if a classical Galois connection is constructed with monadic extensions, then the proofs of $reductive$ and $extensive$, as well as classical proofs of abstraction, can be lowered to their Kleisli counterparts as well.

Theorem 2 (Kleisli Relative Completeness).✓ *For every classical Galois connection $\mathcal{P}(A) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{P}(B)$, where α and γ are of the form $\alpha^k{*}$ and $\gamma^k{*}$ for some α^k and γ^k, there exists a Kleisli Galois connection $A \underset{\alpha^k}{\overset{\gamma^k}{\leftrightarrows}} B$, and for every classical proof of abstraction $\gamma^k{*} \circ F{*} \circ \alpha^k{*} \sqsubseteq F^\sharp{*}$ there exists a Kleisli proof of abstraction $\gamma \diamond F \diamond \alpha \sqsubseteq F^\sharp$.*

Relating constructive to Kleisli Galois connections Constructive Galois connections are initially motivated by the need for constructive abstraction functions $\eta \in A \nearrow B$, as opposed to the non-constructive $\alpha^k \in A \nearrow \mathcal{P}(B)$. Aside from this difference, constructive and Kleisli Galois connections are identical. What is surprising is that constructive Galois connections are not a restricted case of Kleisli Galois connections; *all Kleisli Galois connections are constructive*, which we establish through an isomorphism.

The easy direction of the isomorphism is embedding constructive Galois connections into Kleisli ones. It is easy to construct $\alpha^k \in A \nearrow \mathcal{P}(B)$ from $\eta \in A \nearrow B$ where $\alpha^k \coloneqq pure(\eta)$.

Theorem 3 (Constructive Soundness).✓ *For every constructive Galois connection $A \underset{\eta}{\overset{\gamma^c}{\leftrightarrows}} B$ there exists a Kleisli Galois connection $A \underset{\alpha^k}{\overset{\gamma^k}{\leftrightarrows}} B$ where $\alpha^k \coloneqq pure(\eta)$ and $\gamma^k \coloneqq \gamma^c$, and for every constructive proof of abstraction $\gamma^c \diamond F \diamond pure(\eta) \sqsubseteq F^\sharp$ there exists a Kleisli proof of abstraction $\gamma^k \diamond F \diamond \alpha^k \sqsubseteq F^\sharp$.*

The surprising fact is the other direction: embedding Kleisli Galois connections into constructive ones. Although we verify all theorems in Agda, we include a sketch of the proof for this direction for the interested reader.

Theorem 4 (Constructive Completeness).✓ *For every Kleisli Galois connection $A \underset{\alpha^k}{\overset{\gamma^k}{\leftrightarrows}} B$ there exists a constructive Galois connection $A \underset{\eta}{\overset{\gamma^c}{\leftrightarrows}} B$ where $\gamma^c \coloneqq \gamma^k$ and η is constructed such that $pure(\eta) = \alpha^k$, and for every Kleisli proof of abstraction $\gamma^k \diamond F \diamond \alpha^k \sqsubseteq F^\sharp$ there exists a constructive proof of abstraction $\gamma^c \diamond F \diamond pure(\eta) \sqsubseteq F^\sharp$.*

Proof. To construct η we apply $choice$, the "axiom" of choice which is definable as a theorem in constructive logic, to $sound^k$, the soundness property between α^k and γ^k for Kleisli Galois connections:

$$sound^k : \forall(x).\ \exists(y).\ y \in \alpha^k(x) \wedge x \in \gamma^k(y)$$
$$choice : (\forall(x).\ \exists(y).\ P(x,y)) \Rightarrow (\exists(f).\ \forall(x).\ P(x,f(x)))$$

η is then definable:

$$\exists\eta \mid \forall(x).\ \eta(x) \in \alpha^k(x) \wedge x \in \gamma^k(\eta(x)) \ \wr \text{ by } choice(sound^k)\ \int$$

You can see immediately that $sound^c$ holds for η:

$$sound^c : \forall(x).\ x \in \gamma^c(\eta(x))$$

which follows from the second fact $x \in \gamma^k(\eta(x))$ from $choice(sound^k)$ and $\gamma^k = \gamma^c$. $complete^c$ similarly holds:

$$complete^c : \forall(x,y).\ x \in \gamma^c(y) \Rightarrow \eta(x) \sqsubseteq y$$

The conclusion $\eta(x) \sqsubseteq y$ holds by applying $complete^k$ to premise $x \in \gamma^c(y)$, the first fact $\eta(x) \in \alpha^k(x)$ from $choice(sound^k)$, and $\gamma^c = \gamma^k$. We redisplay $complete^k$ here for reference:

$$complete^k : \forall(x,y_1,y_2).\ x \in \gamma^k(y_1) \wedge y_2 \in \alpha^k(x) \Rightarrow y_2 \sqsubseteq y_1$$

Finally we prove $pure(\eta) = \alpha^k$ as from the fact that α^k and γ^k uniquely determine one another—the Kleisli version of a well known fact about classical Galois connections:

Lemma 1 (Unique Abstraction).✓ *Given two Kleisli Galois connections* $A \underset{\alpha_1^k}{\overset{\gamma_1^k}{\leftrightarrows}} B$ *and* $A \underset{\alpha_1^k}{\overset{\gamma_1^k}{\leftrightarrows}} B$*,* $\alpha_1^k = \alpha_2^k$ *iff* $\gamma_1^k = \gamma_2^k$*.*

$pure(\eta)$ is then equal to α^k as a corollary and $\gamma^c = \gamma^k$. □

Recovering Nonconstructive Galois Connections Constructive Galois connections recover classical ones through embeddings which do not rely on classical axioms. The result is that the original Galois connection between integers from Section 2 is not recovered exactly per se from the lifted constructive Galois connection, rather this version is:

$$\begin{array}{ll} pure(\eta^z)* \in \mathcal{P}(\mathbb{Z}) \nearrow \mathcal{P}(\mathbb{Z}^\sharp) & \gamma^{cz}* \in \mathcal{P}(\mathbb{Z}^\sharp) \nearrow \mathcal{P}(\mathbb{Z}^\sharp) \\ pure(\eta^z)* \coloneqq \{\eta^z(i) \mid i \in I\} & \gamma^{cz}*(I^\sharp) \coloneqq \{i \mid i^\sharp \in I^\sharp \wedge i \in \gamma^z(i^\sharp)\} \end{array}$$

To recover the original Galois connection we compose the lifted constructive Galois connection with the following "joining" Galois connection, which *does* require classical axioms, and which we do not mechanize:

$$\begin{array}{ll} \alpha^{\sqcup^z} \in \mathcal{P}(\mathbb{Z}^\sharp) \nearrow \mathbb{Z}^\sharp & \gamma^{\sqcup^z} \in \mathbb{Z}^\sharp \nearrow \mathcal{P}(\mathbb{Z}^\sharp) \\ \alpha^{\sqcup^z}(I^\sharp) \coloneqq \bigsqcup_{i^\sharp \in I^\sharp} i^\sharp & \gamma^{\sqcup^z}(i^\sharp) \coloneqq \{i^{\sharp\prime} \mid i^{\sharp\prime} \sqsubseteq i^\sharp\} \end{array}$$

The original Galois connection is then recovered as:

$$\mathcal{P}(\mathbb{Z}) \underset{\alpha^{\sqcup^z} \circ\, pure(\eta^z)*}{\overset{\gamma^{cz}* \,\circ\, \gamma^{\sqcup^z}}{\leftrightarrows}} \mathbb{Z}$$

We have thus factored the original Galois connection into two parts: one part fully constructive, as lifted from a constructive Galois connection, and one part classical, which would require the use of classical axioms. Note that $\gamma^{\sqcup^z}$ is just $return$ from the powerset monad, and adds no information due to its cancellation under the monad unit laws.

Recovering Noncomputational Galois Connections Constructive Galois connections are isomorphic to a restricted subset of classical Galois connections which we called computational Galois connections. The classical Galois connections outside this set, i.e., the *non*computational ones, are still useful, and can be manipulated in harmony with constructive ones, even without the use of axioms.

For example, the classical independent attributes Galois connection splits a relations between sets A and B into independent properties on each set, and is defined:

$$\begin{array}{l} \alpha^{IA} : \mathcal{P}(A \times B) \nearrow \mathcal{P}(A) \times \mathcal{P}(B) \\ \alpha^{IA}(XY) \coloneqq \langle \{x \mid \langle x, y\rangle \in XY\}, \{y \mid \langle x, y\rangle \in XY\}\rangle \\ \gamma^{IA} : \mathcal{P}(A) \times \mathcal{P}(B) \nearrow \mathcal{P}(A \times B) \\ \gamma^{IA}(\langle X, Y\rangle) \coloneqq \{\langle x, y\rangle \mid x \in X \wedge y \in Y\} \end{array}$$

Both Galois connection mappings are "honest" about being specifications and can be directly encoded in constructive logic. We define noncomputational Galois connections like independent attributes in our framework without the use of axioms, and it is instrumental in our calculation of Cousot's generic abstract interpreter for `WHILE` where it interacts directly with constructive Galois connections.

7. Related Work

This work connects two long strands of research: abstract interpretation and dependently typed programming. The former is founded on the pioneering work of Cousot and Cousot [10, 11]; the latter on that of Martin-Löf [18], embodied in Norell's Agda [24]. A key technical insight of ours is to use a monadic structure for Galois connections, following the example of Moggi [21] for the λ-calculus.

Calculational abstract interpretation Cousot describes calculational abstract interpretation by example in his lecture notes [8] and monograph [7], and recently introduced a unifying calculus for Galois connections [13]. Our work mechanizes Cousot's calculations and provides a foundation for mechanizing other instances of calculational abstract interpretation (e.g. [19, 26]). We expect our work to have applications to the mechanization of calculational program design [3, 4] by employing only Galois *retractions*, i.e. $\alpha \circ \gamma$ is an identity [13]. There is prior work on mechanized program calculation [28], but it is not based on abstract interpretation.

Verified static analyses Verified abstract interpretation has many promising results [1, 5, 6, 25], scaling up to large-scale real-world static analyzers [16]. Mechanized abstract interpretation has yet to benefit from being built on a compositional Galois connection framework. Until now, approaches use classical axioms or "γ-only" encodings of soundness and (sometimes) completeness. Our techniques for isolating specification effects should complement these approaches.

Monadic abstract interpretation Monads in abstract interpretation have recently been applied to good effect for modularity [14, 27]. However, that work uses monads to structure the semantics, not the Galois connections and proofs.

Future directions There have been recent developments on compositional abstract interpretation frameworks [14] where abstract interpreters and their proofs of soundness are systematically derived side-by-side. That framework relies on correctness properties transported by *Galois transformers*, which we posit would benefit from mechanization since they hold both computational and specification content.

8. Conclusions

This paper realizes the vision of mechanized and constructive Galois connections foreshadowed by Cousot [7, p. 85], giving the first mechanically verified proof by calculational abstract interpretation; once for his generic static analyzer and once for the semantics of gradual typing. Our proofs by calculus closely follow the originals. The primary discrepancy is the use of monads to isolate *specification effects*. By maintaining this discipline, we are able to verify calculations by Galois connections *and* extract computational content from pure results. The resulting artifacts are correct-by-verified-construction, thereby avoiding known bugs.[4]

[4] `di.ens.fr/~cousot/aisoftware/Marktoberdorf98/Bug_History`

Acknowledgments

We thank Ron Garcia and Éric Tanter for discussions of their work. Éric also helped with our French translation. We thank the Colony Club in D.C. and the Board & Brew in College Park for providing a fruitful environment in which to work.

References

[1] G. Barthe, D. Pichardie, and T. Rezk. A certified lightweight non-interference java bytecode verifier. In R. D. Nicola, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007.

[2] F. Besson, D. Cachera, T. Jensen, and D. Pichardie. Certified static analysis by abstract interpretation. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 223–257. Springer Berlin Heidelberg, 2009.

[3] R. Bird and O. de Moor. *The Algebra of Programming*. Prentice Hall, 1996.

[4] R. S. Bird. A calculus of functions for program derivation. In D. A. Turner, editor, *Research Topics in Functional Programming*. Addison-Wesley, 1990. Also available as Technical Monograph PRG-64, from the Programming Research Group, Oxford University.

[5] S. Blazy, V. Laporte, A. Maroneze, and D. Pichardie. Formal verification of a c value analysis based on abstract interpretation. In F. Logozzo and M. Fähndrich, editors, *Static Analysis*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.

[6] D. Cachera and D. Pichardie. A certified denotational abstract interpreter. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010.

[7] P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.

[8] P. Cousot. MIT 16.399: Abstract interpretation, 2005.

[9] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *2nd International Symposium on Programming*, 1976.

[10] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. ACM, 1977.

[11] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '79. ACM, 1979.

[12] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretations. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92. ACM, 1992.

[13] P. Cousot and R. Cousot. A Galois connection calculus for abstract interpretation. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14. ACM, 2014.

[14] D. Darais, M. Might, and D. Van Horn. Galois transformers and modular abstract interpreters. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 552–571. ACM, 2015.

[15] R. Garcia, A. M. Clark, and É. Tanter. Abstracting gradual typing. In *Proceedings of the 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016)*, St Petersburg, FL, USA, Jan. 2016. ACM Press. To appear.

[16] J. H. Jourdan, V. Laporte, S. Blazy, X. Leroy, and D. Pichardie. A Formally-Verified c static analyzer. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15. ACM, 2015.

[17] X. Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 2009.

[18] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[19] J. Midtgaard and T. Jensen. A calculational approach to Control-Flow analysis by abstract interpretation. In M. Alpuente and G. Vidal, editors, *Static Analysis*, Lecture Notes in Computer Science, chapter 23. Springer Berlin Heidelberg, 2008.

[20] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 2006.

[21] E. Moggi. An abstract view of programming languages. Technical report, Edinburgh University, 1989.

[22] D. Monniaux. Réalisation mécanisée d'interpréteurs abstraits. Rapport de DEA, Université Paris VII, 1998. French.

[23] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2004.

[24] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, Sept. 2007.

[25] D. Pichardie. *Interprétation abstraite en logique intuitionniste: extraction d'analyseurs Java certifiés*. PhD thesis, Université Rennes, 2005.

[26] I. Sergey, J. Midtgaard, and D. Clarke. Calculating graph algorithms for dominance and shortest path. In J. Gibbons and P. Nogueira, editors, *Mathematics of Program Construction*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.

[27] I. Sergey, D. Devriese, M. Might, J. Midtgaard, D. Darais, D. Clarke, and F. Piessens. Monadic abstract interpreters. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2013.

[28] J. Tesson, H. Hashimoto, Z. Hu, F. Loulergue, and M. Takeichi. Program calculation in Coq. In M. Johnson and D. Pavlovic, editors, *Algebraic Methodology and Software Technology*, Lecture Notes in Computer Science, chapter 10. Springer Berlin Heidelberg, 2011.