Structure-Preserving Binary Relations for Program Abstraction

David A. Schmidt[⋆]

Computing and Information Sciences Department
Kansas State University
Manhattan, KS 66506 USA.

Abstract. An *abstraction* is a property-preserving contraction of a program's model into a smaller one that is suitable for automated analysis. An abstraction must be sound, and ideally, complete. Soundness and completeness arguments are intimately connected to the abstraction process, and approaches based on homomorphisms and Galois connections are commonly employed to define abstractions and prove their soundness and completeness.
This paper develops Mycroft and Jones's proprosal that an abstraction should be stated as a form of structure-preserving binary relation. Mycroft-Jones-style relations are defined, developed, and employed in characterizations of the homomorphism and Galois-connection approaches to abstraction.

1 Introduction

In program analysis (*static analysis*) [6]), the term, "abstraction," describes a property-preserving contraction of a program's model into one that is smaller and more suitable for automated analysis. Abstraction-generated models are used by compilers to perform data-flow analysis [1, 16], by type-checkers to verify type safety [29], and by model checkers to validate safety properties of systems of processes [3].

An abstraction must be *sound*, in the sense that properties proved true of the abstracted model must hold true of the original program model. Ideally, the abstract model should be most precise, or *complete*, with regards to the properties of interest. Soundness and completeness arguments are intimately connected to the abstraction process.

Our intuitions tell us that a program's model is a kind of algebra, and the abstraction process is a kind of homomorphism, and indeed, the *homomorphism approach* provides a simple means of devising sound abstractions [4].

A crucial insight from the abstract interpretation research of Cousot and Cousot [7–9] is that an abstraction can be defined in terms of a homomorphism and its "inverse" function, giving a *Galois connection.* The inverse function gives us a tool for arguing the completeness of an abstraction.

⋆ Supported by NSF CCR-9970679, INT-9981558, ITR-0085949, and ITR-0086154.

Galois connections possess a rich collection of properties that facilitate correctness proofs of static analyses. But as Mycroft and Jones noted in a seminal paper in 1985 [26], there is a price to be paid for possessing these properties: Program models employing data of compound and higher-order types require Galois connections that operate on powerdomains of overwhelming complexity. Mycroft and Jones suggest that an alternative framework, based on binary relations, can be used to avoid much of this complexity.[1] Then, reasoning techniques based on logical relations [24, 27, 28, 31, 32] apply to the compound and higher-typed values.

The intuition behind Mycroft and Jones's proposal is simple: Given the set of source program states, C, and the set of abstract program states, A, define a binary relation, $\mathcal{R} \subseteq C \times A$, based on the intuition that $c\ \mathcal{R}\ a$ if a is an acceptable modelling of c in the abstract model. For example, for a type-checking abstraction, where concrete data values are abstracted to data-type tags, we might assert that $3\ \mathcal{R}\ \mathit{int}$ and also $3\ \mathcal{R}\ \mathit{real}$, but not $3\ \mathcal{R}\ \mathit{bool}$. In practice, it is relation $\mathcal{R}$ that is crucial to a proof of safety of abstraction.

But this seems too simple—can the homomorphism and Galois-connection approaches be discarded so readily? And can *any* binary relation between C and A define a meaningful property for a static analysis? Mycroft and Jones state that a useful binary relation, $\mathcal{R} \subseteq C \times A$, must possess *LU-closure*: For all $c, c' \in C$, $a, a' \in A$,

$$c' \sqsubseteq_C c,\ c\ \mathcal{R}\ a,\ \text{and } a \sqsubseteq_A a' \text{ imply } c'\ \mathcal{R}\ a'$$

But is this strong enough for proving soundness and completeness? Is it too strong? Does LU-closure characterize homomorphisms or Galois connections?

This paper addresses these and other fundamental questions regarding the homomorphism, Galois connection, and binary-relations approaches to abstraction, relating all three in terms of "structure preserving" binary relations.

The paper's outline goes as follows: Section 2 defines the format of program model—it is a *Kripke structure*, which is an automaton whose states possess local properties of interest that can be preserved or discarded by abstraction. To develop intuition, the Section gives examples of models and abstractions based on Kripke structures.

Next, Section 4 demonstrates how to employ homomorphisms to abstract one Kripke structure by another. The underlying notion of *simulation* is used first to explain the homomorphism approach and next to relate homomorphisms to Galois connections, which are developed in Section 6. Applications of Galois connections to completeness proofs and synthesis of abstract Kripke structures are developed in detail.

Finally, Section 7 introduces Mycroft-Jones-style binary relations and lists crucial structure-preserving properties. These properties are used to state equivalences between structure-preserving forms of binary relations, homomorphisms, and Galois connections.

[1] And in response, Cousot and Cousot proposed a variant, *higher-order abstract interpretation* [10], that sidesteps most of the powerdomain machinery.

Fig. 1. Kripke structure of a process that uses a resource

$$\mathcal{K} = \langle \{s_0, s_1, s_2\}, \{(s_0, s_1), (s_1, s_0), (s_1, s_2), (s_2, s_0), (s_2, s_2)\}, \mathcal{I}_K \rangle$$
$$\text{where } \mathcal{I}_K(s_0) = \{\}, \mathcal{I}_K(s_1) = \{own\}, \mathcal{I}_K(s_2) = \{own, use\}$$

$$\begin{tikzcd}
s_0 \arrow[dr, bend right] & & s_2^{own,use} \arrow[ll, bend right] \arrow[loop right] \\
& s_1^{own} \arrow[ul, bend right] \arrow[ur] &
\end{tikzcd}$$

2 Kripke Structures

We represent a program model as a *Kripke structure*. Simply stated, a Kripke structure is a finite-state automaton whose states are labelled with atomic, primitive "properties" that "hold true" at the states [3, 25]. Kripke structures neatly present program models based on operational semantics, such as flowcharts, data-flow frameworks [16], statecharts [14], and process-algebra processes [23].

Definition 1. *1. A* state-transition system *is a pair,* $\langle \Sigma_K, \rightarrow_K \rangle$, *where*

(a) Σ_K *is a set of states.*

(b) $\rightarrow_K \subseteq \Sigma_K \times \Sigma_K$ *is the transition relation. We write* $s \longrightarrow s'$ *to denote* $(s, s') \in \rightarrow_K$.

2. A state-transition system is labelled *if its second component is revised to* $\rightarrow_K \subseteq \Sigma_K \times \mathcal{L} \times \Sigma_K$, *where* $\mathcal{L}$ *is some fixed, finite set. We write* $s \xrightarrow{\ell} s'$ *to denote* $(s, \ell, s') \in \rightarrow_K$.

3. A state-transition system is a Kripke structure *if it is extended to a triple,* $\langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$, *where* $\mathcal{I}_K : \Sigma_K \rightarrow \mathcal{P}(Atom)$ *associates a set of atomic properties,* $\mathcal{I}_K(s) \subseteq Atom$, *to each* $s \in \Sigma_K$, *for some fixed, finite set, Atom.*

In this paper, we use (unlabelled) Kripke structures; Figure 1 presents a process that requests ownership of and uses a resource. The Kripke structure is presented as an automaton whose states are labelled by atomic properties from $Atom = \{use, own\}$.

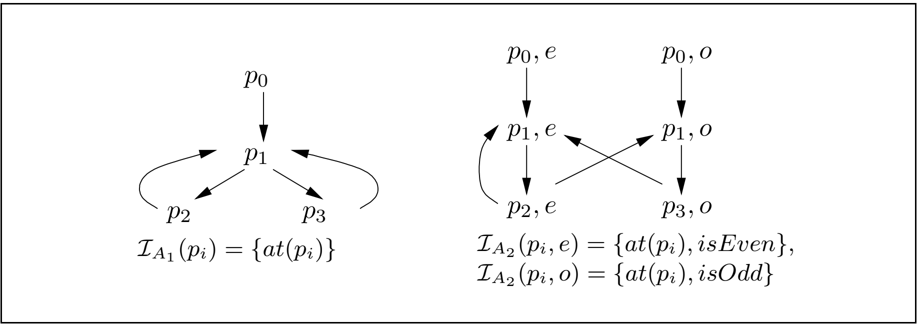
A program's operational semantics can also be modelled as a Kripke structure; Figure 2 presents an infinite-state structure that results from a program that computes upon an input, `x`. A state, (p, n), remembers the value of variable `x` on entry to program point, p. The program's transfer functions are coded as the transition relation.

Atom lists the properties of interest to the model. Here, it is helpful to visualize each $p \in Atom$ as naming some $S_p \subseteq \Sigma_K$ such that $p \in \mathcal{I}_K(s)$ iff $s \in S_p$, e.g., $isEven = \{(p, 2n) \mid n \geq 0\}$. Note that $\mathcal{P}(Atom)$ contains all possible combinations of the properties of interest—when we abstract the Kripke structure, we will use a subset of $\mathcal{P}(Atom)$ as the state space for an abstract model.

```
p0 : while (true) do
   p1 : if isEven(x)
      p2 : then x:= x div2
      p3 : else x:= x + 1
od
```

$$
\begin{aligned}
&\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle\\
&\text{where } \Sigma_K = \mathit{ProgramPoint} \times \mathit{Nat}\\
&\quad \mathit{ProgramPoint} = \{p_0, p_1, p_2, p_3\}\\
&\quad \rightarrow_K = \{\ p_0, n \longrightarrow p_1, n,\\
&\qquad\qquad p_1, 2n \longrightarrow p_2, 2n,\\
&\qquad\qquad p_1, 2n+1 \longrightarrow p_3, 2n+1,\\
&\qquad\qquad p_2, n \longrightarrow p_1, n/2,\\
&\qquad\qquad p_3, n \longrightarrow p_1, n+1\\
&\qquad\qquad \mid\ n \in \mathit{Nat}\}\\
&\quad \mathit{Atom} = \{\mathit{at}(p) \mid p \in \mathit{ProgramPoint}\}\\
&\qquad\qquad \cup \{\mathit{isEven}, \mathit{isOdd}\}\\
&\quad \mathcal{I}_K(p, 2n) = \{\mathit{at(p)}, \mathit{isEven}\}\\
&\quad \mathcal{I}_K(p, 2n+1) = \{\mathit{at(p)}, \mathit{isOdd}\}
\end{aligned}
$$

Fig. 2. Infinite-state Kripke structure of a sequential program


$$\begin{tikzcd}
& p_0 \arrow[d] & \\
& p_1 \arrow[dl] \arrow[dr] & \\
p_2 \arrow[ur, bend left] & & p_3 \arrow[ul, bend right]
\end{tikzcd}$$

$\mathcal{I}_{A_1}(p_i) = \{at(p_i)\}$

$$\begin{tikzcd}
p_0, e \arrow[d] & p_0, o \arrow[d] \\
p_1, e \arrow[d] & p_1, o \arrow[d] \\
p_2, e \arrow[u, bend left] \arrow[ur] & p_3, o \arrow[ul]
\end{tikzcd}$$

$\mathcal{I}_{A_2}(p_i, e) = \{at(p_i), isEven\}$,
$\mathcal{I}_{A_2}(p_i, o) = \{at(p_i), isOdd\}$

Fig. 3. Two abstractions of a program

Abstractions of a Kripke structure Because of its infinite state set, the structure defined in Figure 2 cannot be conveniently drawn pictorially nor can it be easily analyzed mechanically. We must abstract the structure into a manageably sized, finite-state Kripke structure, using the mapping, $\mathcal{I}_K$, to guide us. Figure 3 shows two possible abstractions, represented pictorially. The first abstraction, called a *control abstraction*, defines $\Sigma_{A_1} = \mathit{ProgramPoint}$—data values are forgotten, as are the properties, *isEven* and *isOdd*. Control abstraction generates the control-flow graph used by a compiler.

The second abstraction is a mixture of data and control abstraction: It simplifies the program's data domain (that is, x's value) to $\mathit{EvenOdd} = \{e, o\}$ and defines $\Sigma_{A_2} = \mathit{ProgramPoint} \times \mathit{EvenOdd}$. In consequence, more precise knowledge is presented about the outcomes of the test at program point p_1 (cf. [38]) in contrast to a control abstraction.

The state sets for both abstractions in Figure 3 are just subsets of $\mathcal{P}(\mathit{Atom})$ from Figure 2: $\Sigma_{A_2} = \mathcal{I}_K(\Sigma_K)$, and $\Sigma_{A_1} = \mathcal{I}_K(\Sigma_K) \setminus \{\mathit{isEven}, \mathit{isOdd}\}$, where $X \setminus Y$ denotes the sets of $X \subseteq \mathcal{P}(\mathit{Atom})$ with occurrences of $a \in Y$ removed.

The characterization of each $a \in \Sigma_{A_i}$ as $a \subseteq Atom$ induces the abstract structure's property map—it is simply, $\mathcal{I}_{A_i}(a) = a$.

Derivation of abstract transitions Given the abstract state set, Σ_A, we must derive a transition relation, $\rightarrow_A$; we begin with examples for needed intuition.

The intuition behind the second structure in Figure 3 is that *Nat*, the set of natural numbers, was abstracted to $EvenOdd = \{e, o\}$, such that even-valued naturals are represented by e and odd-valued naturals are represented by o. The addition and division-by-two operations, encoded in the transition relation in Figure 2, must be restated to operate on the new set. By necessity, the semantics of division-by-two is nondeterministic:

$$\begin{array}{lll} e+1 \mapsto o & e/2 \mapsto e & o/2 \mapsto e \\ o+1 \mapsto e & e/2 \mapsto o & o/2 \mapsto o \end{array}$$

The operations above generate $\rightarrow_{A_2}$—in particular, we have these crucial transition rules for the decision point at p_1:

$$\begin{array}{l} p_1, e \longrightarrow p_2, e \\ p_1, o \longrightarrow p_3, o \end{array}$$

The rules retain information about the properties of the data that arrive at program points p_2 and p_3.

The above abstraction generates a finite set of states for the Kripke structure in Figure 2. But the finite cardinality might be too huge for practical analysis, and a more severe abstraction might be required. For this, one can partially order the abstract-data domain—the partial ordering helps us merge similar abstract states and so reduce the cardinality of the state set. For example, we might add the data values, $\bot$ ("no value at all") and $\top$ ("any value at all") to the *EvenOdd* set and partially order the result as follows:

$$EvenOdd_\bot^\top = \begin{tikzcd} & \top \arrow[dl, dash] \arrow[dr, dash] & \\ e \arrow[dr, dash] & & o \arrow[dl, dash] \\ & \bot & \end{tikzcd}$$

The ordering suggests precision-of-information-content—e.g., e is more precise than $\top$ about the range of numbers it represents. (To rationalize this explanation, $\bot$ is the most "precise" of all, because it precisely notes no value at all.) This particular partial ordering is no accident: It comes from our belief that abstract values and abstract states are just sets of properties. Here, $EvenOdd_\bot^\top$ is isomorphic to $\mathcal{P}\{isEven, isOdd\}$, ordered by superset inclusion.

The arithmetic operations must be revised for the new domain:

$$\begin{array}{ll} e/2 \mapsto \top & e+1 \mapsto o \\ o/2 \mapsto \top & o+1 \mapsto e \\ \top/2 \mapsto \top & \top+1 \mapsto \top \end{array}$$

$$\begin{tikzcd}
p_0, \top \arrow[d] & \\
p_1, \top \arrow[r] \arrow[d] & p_3, o \arrow[d] \\
p_2, e \arrow[u, bend left] & p_1, e \arrow[l]
\end{tikzcd}$$

$\mathcal{I}_{A_3}(p_i, e) = \{at(p_i), isEven\}$
$\mathcal{I}_{A_3}(p_i, o) = \{at(p_i), isOdd\}$
$\mathcal{I}_{A_3}(p_i, \top) = \{at(p_i)\}$

Fig. 4. Kripke structure generated with partially ordered data domain

(Operations on $\perp$ are ignored.) In particular, division by 2 produces $\top$ as its result. Now, program states have the form, *ProgramPoint* $\times$ $\mathit{EvenOdd}_{\perp}^{\top}$, where $p, \perp$ denotes a program point that is never reached during execution ("dead code") and $p, \top$ represents a program point that may be reached with both even- and odd-valued numbers. The presence of $\top$ suggests these transition rules for the program's decision point, p_1:[2]

$$p_1, \top \longrightarrow p_2, e$$
$$p_1, \top \longrightarrow p_3, o$$

We use $\mathit{EvenOdd}_{\perp}^{\top}$ with the above transition rules for p_1 to generate a new Kripke structure for the example in Figure 2; see Figure 4. The resulting structure has one fewer state than its counterpart in Figure 3.

State merging We can generate a smaller structure from Figure 4 by merging the two states, (p_1, e) and $(p_1, \top)$. This merge yields the result state, $(p_1, \top)$, because $e \sqcup \top = \top$ in the ordering on the data values. The transitions to and from (p_1, e) are "folded" into $(p_1, \top)$, and the resulting structure looks like this:

$$\begin{tikzcd}
& p_0, \top \arrow[d] & \\
& p_1, \top \arrow[dl] \arrow[dr] & \\
p_2, e \arrow[ur, bend left] & & p_3, o \arrow[ul, bend right]
\end{tikzcd}$$

In data-flow analysis, it is common to perform state-merging "on the fly" while generating the Kripke structure: Σ_K is defined as a subset of *ProgramPoint* $\times$ *DataFlowInformation* such that, if $(p_k, d_1) \in \Sigma_K$ and $(p_k, d_2) \in \Sigma_K$, then $d_1 = d_2$; that is, there is at most one state per program point. When the Kripke structure is generated by executing the source program with the data domain, *DataFlowInformation*, newly generated states of form (p_k, d_i) are merged with the existing state, (p_k, d), giving $(p_k, d_i \sqcup d)$.

[2] The following transition rules for p_1 would also be acceptable, but they lose too much precision: $p_1, \top \longrightarrow p_2, \top$ and $p_1, \top \longrightarrow p_3, \top$.

Monotone Kripke Structures The structure in Figure 4 has "related" states: $(p_1, \top)$ and (p_1, e) are related because $e \sqsubseteq \top$. To formalize this, we partially order the state set, Σ_{A_3}, such that $p_i, d_i \sqsubseteq_{A_3} p_j, d_j$ iff $p_i = p_j$ and $d_i \sqsubseteq_{EvenOdd_\bot^\top} d_j$.

An expected consequence of this relationship should be that, if $p_1, e \longrightarrow q$ then also $p_1, \top \longrightarrow q'$, where $q \sqsubseteq_{A_3} q'$—the less precise state should be consistent with the more precise one. Similar thinking causes us to demand, if $p_1, e \sqsubseteq_{A_3} p_1, \top$, then $\mathcal{I}_{A_3}(p_1, e) \supseteq \mathcal{I}_{A_3}(p_1, \top)$. These are *monotonicity* properties:

Definition 2. *For Kripke structure, $\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$ such that Σ_K is partially ordered by $\sqsubseteq_K \subseteq \Sigma_K \times \Sigma_K$,*

1. *$\rightarrow_K$ is* monotone (with respect to $\sqsubseteq_K$) *iff for all $s, s' \in \Sigma_K$, $s \sqsubseteq_K s'$ and $s \longrightarrow s_1$ imply there exists $s_2 \in \Sigma_K$ such that $s' \longrightarrow s_2$ and $s_1 \sqsubseteq_K s_2$:*

$$\begin{tikzcd}
s \arrow[r, phantom, "\sqsubseteq_K"] \arrow[d] & s' \arrow[d] \\
s_1 \arrow[r, phantom, "\sqsubseteq_K"] & s_2
\end{tikzcd}$$

2. *$\mathcal{I}_K$ is* monotone (with respect to $\sqsubseteq_K$) *iff $s \sqsubseteq_K s'$ implies $\mathcal{I}_K(s) \supseteq \mathcal{I}_K(s')$.*

A Kripke structure whose state set is partially ordered is monotone *if both its transition relation and property map are monotone.*

Since a Kripke structure's transitions encode a program's transfer functions, it is natural to demand that the transition relation be monotone. The property map must be monotone to ensure that the partial ordering on states is sensible. We work exclusively with monotone Kripke structures—the monotonicity is crucial to proofs of soundness and completeness of abstractions.

3 Relating Models

Abstraction must relate a "detailed" Kripke structure with a "less detailed" counterpart. We call the detailed Kripke structure the *concrete structure* and name it $\mathcal{C}$, and we call the less detailed structure the *abstract structure* and name it $\mathcal{A}$. When we compare two Kripke structures, $\mathcal{C}$ and $\mathcal{A}$, *we assume they use the same set, Atom, as the codomain of their respective property maps.* This facilities easy definitions of soundness and completeness.

4 Homomorphisms

The classic tool for relating one algebraic structure to another is the homomorphism; here is the variant for Kripke structures:

Definition 3. *For Kripke structures $\mathcal{C} = \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle$ and $\mathcal{A} = \langle \Sigma_A, \rightarrow_A, \mathcal{I}_A \rangle$, a* homomorphism from $\mathcal{C}$ to $\mathcal{A}$ *is a function, $h : \Sigma_C \rightarrow \Sigma_A$, such that*

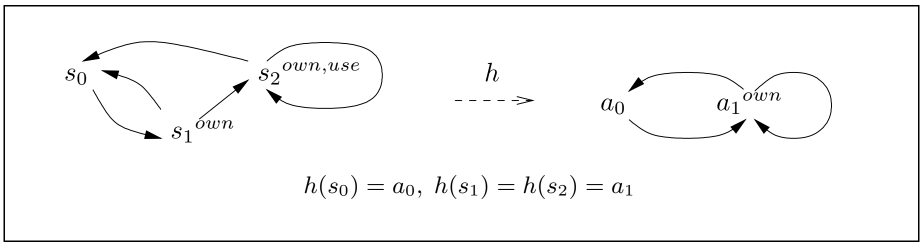


Fig. 5. Example homomorphism

1. *for all $c \in \Sigma_C$, $c \longrightarrow c'$ implies there exists a transition, $h(c) \longrightarrow a'$ such that $h(c') \sqsubseteq_A a'$:*

$$\begin{tikzcd}
c \arrow[r, dotted, "h"] \arrow[d] & h(c) \arrow[d] \\
c' \arrow[r, dotted, "h"] & h(c') \sqsubseteq_A a'
\end{tikzcd}$$

2. *for all $c \in \Sigma_C$, $\mathcal{I}_C(c) \supseteq \mathcal{I}_A(h(c))$.*

Figure 5 displays a simple example of two structures and their relationship by homomorphism. The Figure shows a homomorphism that preserves transitions up to equality. But the Definition does not require this—$h(c') \sqsubseteq a'$ is stated, rather than $h(c') = a'$, to handle the situation where Σ_A is nontrivially partially ordered. For example, reconsider Figures 2 and 4, where Σ_C is $\mathit{ProgramPoint} \times \mathit{Nat}$, Σ_A is $\mathit{ProgramPoint} \times \mathit{EvenOdd}_{\perp}^{\top}$, and consider the transition, `x := x div 2`:

$$\begin{tikzcd}
p_2, 6 \arrow[r, dotted, "h"] \arrow[d, "\texttt{x:= x div2}"'] & p_2, e \arrow[d, "\texttt{x:= x div2}"] \\
p_1, 3 \arrow[r, dotted, "h"] & p_1, o \sqsubseteq p_1, \top
\end{tikzcd}$$

where $h(\mathit{pgmpoint}, n) = (\mathit{pgmpoint}, \mathit{polarity}(n))$ and $\mathit{polarity}(2n) = e$ and $\mathit{polarity}(2n+1) = o$.

Given a concrete (monotone) Kripke structure, $\mathcal{C} = \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle$, we can use its property map, $\mathcal{I}_C : \Sigma_C \rightarrow \mathcal{P}(\mathit{Atom})$, as a homomophism onto the monotone Kripke structure, $A_{\mathcal{I}_C} = \langle \Sigma_A, \rightarrow_A, \mathcal{I}_A \rangle$:

$$\begin{aligned}
&\Sigma_A = \mathcal{I}_C(\Sigma_C) \\
&\rightarrow_A = \{a \rightarrow_A \mathcal{I}_C(c') \mid c \rightarrow_C c',\ \ a \subseteq \mathcal{I}_C(c)\} \\
&\mathcal{I}_A(a) = a
\end{aligned}$$

As suggested earlier, we use the range of $\mathcal{I}_C$ as the states for the abstract structure; the definition can be proved to be "complete" [3] for state set, $\mathcal{I}_C(\Sigma_C)$ [8, 12].

[3] The formalization of "complete" must wait until the section on Galois connections, where needed technical machinery is provided.

Standard uses of homomorphisms to relate concrete and abstract structures can be found in papers by Clarke, Grumberg, and Long [5] and Clarke, et al. [2], among others.

5 Simulations

A homomorphism between Kripke structures preserves the ability to transit between states. This notion is better known as a *simulation.* Let $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$ be a binary relation on the states of structures $\mathcal{C}$ and $\mathcal{A}$, and write $c\,\mathcal{R}\,a$ when $(c, a) \in \mathcal{R}$.

Definition 4. *For Kripke structures $\mathcal{C} = \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C\rangle$ and $\mathcal{A} = \langle \Sigma_A, \rightarrow_A, \mathcal{I}_A\rangle$, a binary relation, $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$, is a* simulation *of $\mathcal{C}$ by $\mathcal{A}$, written $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$, iff for every $c \in \Sigma_C$, $a \in \Sigma_A$, if $c\,\mathcal{R}\,a$, then*

1. *if $c \longrightarrow c'$, then there exists $a' \in \Sigma_A$ such that $a \longrightarrow a'$ and $c'\,\mathcal{R}\,a'$:*

$$\begin{tikzcd} c \arrow[r, dotted, no head, "\mathcal{R}"] \arrow[d] & a \arrow[d] \\ c' \arrow[r, dotted, no head, "\mathcal{R}"] & a' \end{tikzcd}$$

2. *$\mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)$.*

In the above situation, we also say that $\mathcal{A}$ *simulates* $\mathcal{C}$.[4] When $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$, then $c\,\mathcal{R}\,a$ asserts that a is an acceptable abstraction of c—properties true for a hold true for c, and a's transitions mimick all transitions made by c. More generally, any sequence of transitions in $\mathcal{C}$ can be mimicked by a sequence in $\mathcal{A}$.[5]

If we reexamine Figure 5, we readily see the underlying simulation between the concrete and abstract structures: $\mathcal{R} = \{(s_0, a_0), (s_1, a_1), (s_2, a_1)\}$. And, when we study the program flowgraph in Figure 2 and its two even-odd abstractions in Figures 3 and 4, we see this simulation relates states in the concrete structure to those in the abstract structures:

$$\begin{aligned} \mathcal{R} = \{\ & ((p_i, 2n), (p_i, e)), \\ & ((p_i, 2n+1), (p_i, o)), \\ & ((p_i, n), (p_i, \top))\ \mid\ i \geq 0, n \in \mathit{Nat}\} \end{aligned}$$

We demand that simulations are *left total*:

Definition 5. *A binary relation, $\mathcal{R} \subseteq S \times T$, is* left total *if for all $s \in S$, there exists some $t \in T$ such that $s\,\mathcal{R}\,t$. The relation is* right total *if for all $t \in T$, there exists some $s \in S$ such that $s\,\mathcal{R}\,t$.*

[4] If labelled transitions are employed, the simulation must respect the labels: $c\,\mathcal{R}\,a$ and $c \xrightarrow{\ell} c'$ implies that $a \xrightarrow{\ell} a'$ and $c'\,\mathcal{R}\,a'$.

[5] Thus, properties that hold true for all *paths* starting at a also hold true for all paths starting at c. This supports sound verification of LTL- and ACTL-coded properties of temporal-logic on the abstract Kripke structure [4, 12, 34, 37].

A left-total simulation ensures that every state in the concrete structure can be modelled in the abstract structure. Right totality ensures that there are no superfluous abstract states.

Simulations play a crucial role in equivalence proofs of interpreters [20]: Each execution step of interpreter, $\mathcal{C}$, for a source-programming language is mimicked by a (sequence of) execution steps of an interpreter, $\mathcal{A}$, for the target programming language, and mathematical induction justifies that sequences of transitions taken by $\mathcal{C}$ can be mimicked by $\mathcal{A}$. Coinductive reasoning [23, 30] extends the proof to sequences of infinite length. More recently, correctness proofs of hardware circuits, protocols, and systems of processes has been undertaken by means of (bi)simulations [23].

It is easy to extract the simulation embedded within a homomorphism: For $h : \Sigma_C \rightarrow \Sigma_A$, define

$$c\,\mathcal{R}_h\,a \text{ iff } h(c) \sqsubseteq_A a$$

As before, $h(c) \sqsubseteq_A a$ is stated, rather than $h(c) = a$, to take into account the partial ordering, if any, upon Σ_A.

Since we can extract a useful simulation from a homomorphism, we might ask if the dual is possible: The answer is "no"—a homomorphism is a function, and it takes little work to invent simulation relations that are not functions.

It is possible to synthesize a simulation for two Kripke structures, $\mathcal{C}$ and $\mathcal{A}$, from scratch: Define this hierarchy of binary relations, $\mathcal{R}_i \subseteq \Sigma_C \times \Sigma_A$, for $i \geq 0$:

$$\begin{array}{l}\mathcal{R}_0 = \Sigma_C \times \Sigma_A \\ \mathcal{R}_{i+1} = \{(c, a) \mid \mathcal{I}_C(c) \supseteq \mathcal{I}_A(a), \text{ and also} \\ \qquad\qquad\qquad\quad \text{if } c \longrightarrow c', \text{then there exists } a' \in \Sigma_A \text{ such that} \\ \qquad\qquad\qquad\quad a \longrightarrow a' \text{ and } c'\ \mathcal{R}_i\ a'\}\end{array}$$

$c\ \mathcal{R}_i\ a$ asserts that a mimicks c for up to i transitions, so we define the limit relation, $\mathcal{R}_\infty = \bigcap_{i \geq 0} \mathcal{R}_i$. As shown by Park [30] and Milner [22], for finite-image Kripke structures $\mathcal{C}$ and $\mathcal{A}$, $\mathcal{R}_\infty$ defines a simulation, and indeed, it is the largest possible simulation on $\mathcal{C}$ and $\mathcal{A}$: If $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$, then $\mathcal{R} \subseteq \mathcal{R}_\infty$.

Alas, the synthesis technique cannot produce a left-total simulation when one is impossible: Consider $\mathcal{C} = \langle \{c_0, c_1\}, \{c_0 \longrightarrow c_1\}, \mathcal{I}_C \rangle$ and $\mathcal{A}\langle \{a_0\}, \{\}, \mathcal{I}_A \rangle$—we calculate $\mathcal{R}_\infty = \{(c_1, a_0)\}$, which means there is no way to abstract c_0 within $\mathcal{A}$—the abstract structure is unsuitable. In this fashion, the synthesis technique can be used to decide whether one finite-state structure can be simulated by another.

Simulations give the foundation for reasoning about the relationships between structures. We next consider a tool that lets us reason about "complete" simulations.

6 Galois connections

Widely used for abstraction studies [7, 8, 17, 21], Galois connections are a form of "invertible homomorphism" that come with powerful techniques for analyzing the precision of abstractions.

Definition 6. *For partially ordered sets P and Q, a* Galois connection *is a pair of functions, $(\alpha\colon P \to Q, \gamma\colon Q \to P)$, such that for all $p \in P$ and $q \in Q$, $p \sqsubseteq_P \gamma(q)$ iff $\alpha(p) \sqsubseteq_Q q$:*

$$\begin{tikzcd}
\gamma(q) & q \arrow[l, dotted, "\gamma"'] \\
\sqcup|_P & \sqcup|_Q \\
p \arrow[r, dotted, "\alpha"] & \alpha(p)
\end{tikzcd}$$

Equivalently, α, γ form a Galois connection iff

1. *α and γ are monotone*
2. *$\alpha \circ \gamma \sqsubseteq id_Q$*
3. *$id_P \sqsubseteq \gamma \circ \alpha$.*

We say that α is the lower adjoint *and γ is the* upper adjoint *of the Galois connection.*

A plethora of properties can be proved about Galois connections; here are a few:

Theorem 7. *Say that $(\alpha\colon P \to Q, \gamma\colon Q \to P)$ is a Galois connection:*

1. *The adjoints uniquely determine each other, that is, if (α, γ') is also a Galois connection, then $\gamma = \gamma'$ (similarly for γ, α, and α').*
2. *$\alpha(p) = \sqcap\gamma^{-1}(\uparrow p)$ and $\gamma(q) = \sqcup\alpha^{-1}(\downarrow q)$, where $\uparrow p = \{p' \in P \mid p \sqsubseteq_P p'\}$ and $\downarrow q = \{q' \in Q \mid q' \sqsubseteq_Q q\}$*
3. *$\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$.*
4. *α is one-one iff γ is onto iff $\gamma \circ \alpha = id_P$; γ is one-one iff α is onto iff $\alpha \circ \gamma = id_Q$.*
5. *$\alpha[P]$ and $\gamma[Q]$ are isomorphic partially ordered sets.*
6. *α preserves all joins, and γ preserves all meets*
7. *If P and Q are complete lattices, then so are $\alpha[P]$ and $\gamma[Q]$, but they need not be sublattices.*

Proofs of these results can be found in [21] as well as in many other sources.

Result (ii) implies that one can validate when a function α (dually, γ) is a component of a Galois connection by merely checking the wellformedness of the definition of its partner: α is the lower adjoint of a Galois connection iff $\alpha^{-1}(\downarrow q)$ is a principal ideal in P, for every $q \in Q$. (A *principal ideal* is a set, $S \subseteq P$, such that $S = \downarrow p$, for some $p \in P$.)

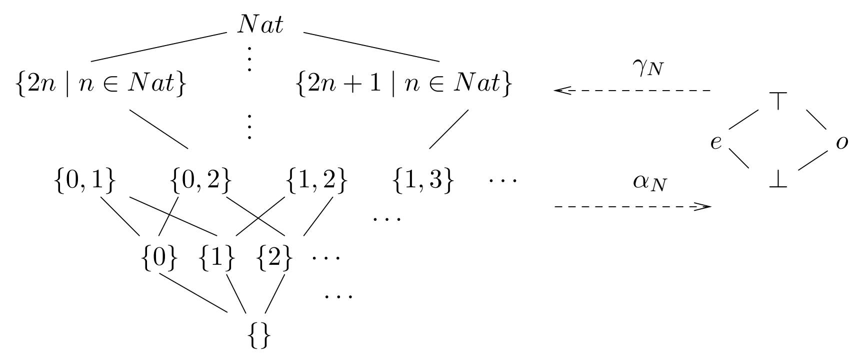
Say that an abstract Kripke structure, $\mathcal{A}$, has a partially ordered state set, $(\Sigma_A, \sqsubseteq_A)$. (For simplicity, say that it is a complete lattice.) We relate the concrete structure, $\mathcal{C}$, to $\mathcal{A}$, by means of a Galois connection of this form:[6]

$$\langle \alpha : (\mathcal{P}(\Sigma_C), \subseteq) \to (\Sigma_A, \sqsubseteq_A),\ \gamma : (\Sigma_A, \sqsubseteq_A) \to (\mathcal{P}(\Sigma_C), \subseteq)\rangle$$

[6] Assume that Σ_C has no partial ordering of its own or that we ignore it [10].

We say that α is the *abstraction map* and γ is the *concretization map* [7]—$\alpha(S)$ maps a set of states, $S \subseteq \Sigma_C$, to its most precise approximation in Σ_A—this is ideal for performing control abstraction, where several states must be merged into one. Of course, $\alpha\{c\}$ maps a single state, c, to its abstract counterpart, like a homomorphism would do. Dually, $\gamma(a)$ maps an abstract state to the set of concrete states that a represents.

Figure 6 displays a Galois connection between the data values used in the structures in Figures 2 and 4. In the Figure, a Galois connection is first defined



A Galois connection between the above sets:

$$
\begin{aligned}
&\alpha_N : \mathcal{P}(Nat) \to \mathit{EvenOdd}_\bot^\top\\
&\quad \alpha_N(S) = \bigsqcup(\{e \mid \text{exists } n \in Nat, 2n \in S\}\\
&\qquad\qquad\qquad \cup\{o \mid \text{exists } n \in Nat, 2n+1 \in S\})\\
&\gamma_N : \mathit{EvenOdd}_\bot^\top \to \mathcal{P}(Nat)\\
&\quad \gamma_N(a) = \{2n \mid n \in Nat, e \sqsubseteq a\} \cup \{2n+1 \mid n \in Nat, o \sqsubseteq a\}
\end{aligned}
$$

Examples: $\alpha_N\{2,6\} = e, \alpha_N\{2,3,4\} = \top, \gamma_N(e) = \{0,2,4,\cdots\}, \gamma_N(\top) = Nat$.
Using (α_N, γ_N) to define a Galois connection between $\mathcal{P}(\Sigma_C)$ and Σ_A:

$$
\begin{aligned}
&\Sigma_C = \mathit{ProgramPoint} \times Nat \qquad \Sigma_A = (\mathit{ProgramPoint} \times \mathit{EvenOdd}_\bot^\top)_\bot^\top\\
&\alpha : \mathcal{P}(\Sigma_C) \to \Sigma_A\\
&\quad \alpha(T) = \begin{cases} \bot, & \text{if } T = \{\}\\ (p_k, \alpha_N(T \downarrow 2)), & \text{if } T \downarrow 1 = \{p_k\}\\ \top, & \text{if } p_i, p_j \in T \downarrow 1, i \not= j \end{cases}\\
&\quad \text{where } T \downarrow 1 = \{p \mid (p,n) \in T\}, \quad T \downarrow 2 = \{n \mid (p,n) \in T\}\\
&\gamma : \Sigma_A \to \mathcal{P}(\Sigma_C)\\
&\quad \gamma(p,a) = \{(p,n) \mid n \in \gamma_N(a)\}\\
&\quad \gamma(\bot) = \{\}\\
&\quad \gamma(\top) = \Sigma_C
\end{aligned}
$$

Fig. 6. Example Galois connection

between the sets of data values, $\mathcal{P}(Nat)$ and $EvenOdd_{\bot}^{\top}$, used by the concrete and abstract structures, respectively. Then, this Galois connection is used to define a Galois connection between the state sets of the two structures. (To make a complete lattice, Σ_A is augmented by $\bot$ and $\top$. As a technicality, we assume $(\top, \top) \in \rightarrow_A$.)

The Galois connection in the Figure makes clear that the abstract structure can remember at most one program point in its state. One might devise an abstract structure that remembers more than one program point at a time, e.g., $\Sigma_A = \mathcal{P}(ProgramPoint) \times EvenOdd_{\bot}^{\top}$, or better still, $\Sigma_A = \mathcal{P}(ProgramPoint \times EvenOdd_{\bot}^{\top})$. Such state sets would be useful for defining abstract structures that model nondeterministic computations.

Extracting the simulation within a Galois connection Thanks to Section 5, we know that we relate Kripke structures by simulations. Given the concrete Kripke structure, $\mathcal{C}$, an abstract structure, $\mathcal{A}$, and Galois connection, $(\alpha\colon \mathcal{P}(\Sigma_C) \to \Sigma_A, \gamma\colon \Sigma_A \to \mathcal{P}(\Sigma_C))$[7], we must prove a simulation exists, in the sense of Definition 4. To do this, we extract the following relation, $\mathcal{R}_\gamma \subseteq \Sigma_C \times \Sigma_A$, from the Galois connection:

$$c \,\mathcal{R}_\gamma\, a \text{ iff } c \in \gamma(a)$$

(Or, equivalently stated, $c \,\mathcal{R}_\gamma\, a$ iff $\alpha\{c\} \sqsubseteq_A a$.) For example, it is easy to prove that the relation extracted from the Galois connection in Figure 6 is a simulation.

Synthesizing an Abstract Kripke Structure from a Concrete Structure and an Abstract Domain In practice, we use a Galois connection to *synthesize* a "best" abstract Kripke structure for a concrete one: Say that we have $\mathcal{C} = \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle$, and a complete lattice A, and a monotone property map, $\mathcal{I}_A : A \to \mathcal{P}(Atom)$.[8] We synthesize a Galois connection, and from it, an abstract Kripke structure that is sound (there exists a simulation) and complete (any other abstract structure that uses A as its state set and simulates $\mathcal{C}$ computes less precise properties).

First, consider which element in A may approximate some $c \in \Sigma_c$; the element must come from this set:

$$S_c = \{a' \in A \mid \mathcal{I}_C(c) \supseteq \mathcal{I}_A(a')\}$$

Define $\beta(c) = \sqcap S_c$; now, *if for every $c \in \Sigma_C$, $\beta(c)$ is an element of S_c, we can define a Galois connection* between $\mathcal{P}(\Sigma_C)$ and A (c.f. Theorem 7(2)):

$$\begin{aligned}&\alpha(S) = \sqcup_{s \in S}\beta(s)\\&\gamma(a) = \{c \mid \mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)\}\end{aligned}$$

[7] We omit the partial orderings to save space.

[8] Of course, if A is a sublattice of $\mathcal{P}(Atom)$, we take $\mathcal{I}_A(a) = a$.

We use the Galois connection to define this transition relation on A:

$$\rightarrow_A = \{a \rightarrow_A \alpha\{c'\} \mid \text{there exists } c \rightarrow_C c' \text{ and } c \in \gamma(a)\}$$

The abstract Kripke structure is defined as $\mathcal{A}_\gamma = \langle A, \rightarrow_A, \mathcal{I}_A \rangle$. The definition of $\rightarrow_A$ ensures that $\mathcal{A}_\gamma$ can simulate $\mathcal{C}$, that is, $\mathcal{C} \lhd_{\mathcal{R}_\gamma} \mathcal{A}_\gamma$, where $c\, \mathcal{R}_\gamma\, a$ iff $c \in \gamma(a)$, as usual. Note that $\mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)$ iff $\beta(c) \sqsubseteq_A a$.

We can prove that $\mathcal{A}_\gamma$ is complete in the following sense: Say that there is another Kripke structure, $\mathcal{A}' = \langle A, \rightarrow'_A, \mathcal{I}_A \rangle$ such that $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}'$, where $c\, \mathcal{R}\, a$ iff $\mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)$. *But* $\mathcal{R}$ *is just* $\mathcal{R}_\gamma$. The Galois connection between $\mathcal{P}(\Sigma_C)$ and A lets us prove that $\mathcal{A}_\gamma \lhd_{\mathcal{R}_{\sqsubseteq_A}} \mathcal{A}'$, where $a\, \mathcal{R}_{\sqsubseteq_A}\, a'$ iff $a \sqsubseteq_A a'$, implying $\mathcal{I}_A(a) \supseteq \mathcal{I}_A(a')$. Hence, $\mathcal{A}_\gamma$ is most precise.[9]

Defining a Kripke Structure from a Galois connection Given a state-transition system, $\langle \Sigma_C, \rightarrow_C \rangle$, we can use a Galois connection to define an appropriate mapping, $\mathcal{I}_C$, to make the state-transition system into a Kripke structure.

Say we have a Galois connection, $(\alpha{:}\, \mathcal{P}(\Sigma_C) \rightarrow A, \gamma{:}\, A \rightarrow \mathcal{P}(\Sigma_C))$, where A is a complete lattice. A serves as the collection of expressible properties—use $\mathcal{I}_C(c) = \uparrow \alpha\{c\} = \{a' \mid \alpha\{c\} \sqsubseteq a'\}$ for $c \in \Sigma_C$. Furthermore, we use the techniques from the previous section to define a best abstraction of $\mathcal{C}$:

$$\mathcal{A} = \langle A, \{a \rightarrow_A \alpha(c') \mid c \rightarrow_C c', c \in \gamma(a)\}, \lambda a.\uparrow a \rangle$$

Analyzing Nondeterminism with Galois Connections Galois connections are well suited for studying the transitions that might be taken from a *set* of concrete states. We can readily synthesize the appropriate Kripke structure that represents the nondeterministic version of $\mathcal{C}$: For $S, S' \subseteq \Sigma_C$, write $S \overset{\bullet}{\longrightarrow} S'$ to assert that for every $c' \in S'$, there exists some $c \in S$ such that $c \longrightarrow c'$. Next, we define the monotone Kripke structure, $\mathcal{PC}$, into which $\mathcal{C}$ embeds:

$$\mathcal{PC} = \langle \mathcal{P}(\Sigma_C), \overset{\bullet}{\rightarrow}, \mathcal{I}_{PC} \rangle, \text{ where } \mathcal{I}_{PC}(S) = \cap\{\mathcal{I}_C(c) \mid c \in S\}$$

Given a Galois connection, $(\alpha : \mathcal{P}(\Sigma_C) \rightarrow \Sigma_A, \gamma : \Sigma_A \rightarrow \mathcal{P}(\Sigma_C))$, we readily define this relation, $\mathcal{R}_{(\alpha,\gamma)} \subseteq \mathcal{P}(\Sigma_C) \times \Sigma_A$:

$$S\, \mathcal{R}_{(\alpha,\gamma)}\, a \text{ iff } \alpha(S) \sqsubseteq_A a$$

or equivalently,

$$S\, \mathcal{R}_{(\alpha,\gamma)}\, a \text{ iff } S \subseteq \gamma(a)$$

To verify that $\mathcal{R}_{(\alpha,\gamma)}$ is a simulation, we must "complete" the abstract structure, $\mathcal{A}$, with all the abstract transitions it needs to model transitions from sets of concrete states to sets of concrete states:

[9] This reasoning is entirely analogous to the reasoning done with Galois connections on abstractions of functional definitions to prove that the abstract transfer function, $\alpha \circ f \circ \gamma$, is the most precise abstraction of the concrete transfer function, f.

Definition 8. *Given a monotone Kripke structure, $\mathcal{A} = \langle \Sigma_A, \rightarrow_A, \mathcal{I}_A \rangle$, such that Σ_A is a complete lattice, we define its* closure *as follows:*

$$\begin{array}{l} closure(\mathcal{A}) = \langle \Sigma_A, \rightarrow_{CA}, \mathcal{I}_A \rangle \\ \quad where\ \rightarrow_{CA} = \ \{ a \rightarrow \sqcup T \ \mid \ T \subseteq \{ a' \ \mid \ a \rightarrow_A a' \} \ \} \end{array}$$

For example, if $a_0 \rightarrow_A a_1$ and $a_0 \rightarrow_A a_2$ in $\mathcal{A}$, then these transitions, as well as $a_0 \rightarrow_{CA} (a_1 \sqcup a_2)$, appear in *closure*($\mathcal{A}$). The closure construction builds a monotone Kripke structure and shows us how the relation on elements of Σ_C naturally lifts into a relation on *subsets* of Σ_C:

Theorem 9. *If $\mathcal{A}$ is monotone, then $\mathcal{C} \lhd_{\mathcal{R}_\gamma} \mathcal{A}$ implies $\mathcal{PC} \lhd_{\mathcal{R}_{(\alpha,\gamma)}}$ closure($\mathcal{A}$).*

Lifting a homomorphism into a Galois connection An elegant alternative to writing a Galois connection directly is using a homomorphism as an intermediary: Given Kripke structures $\mathcal{C}$ and $\mathcal{A}$, prove that $h : \Sigma_C \rightarrow \Sigma_A$ is a homomorphism, so that we have $\mathcal{C} \lhd_{\mathcal{R}_h} \mathcal{A}$. Next, "lift" h into the Galois connection, $(\alpha_h : \mathcal{P}(\Sigma_C) \rightarrow \Sigma_A, \gamma_h : \Sigma_A \rightarrow \mathcal{P}(\Sigma_C))$, as follows [27]:

$$\begin{array}{l} \alpha_h(S) = \sqcup\{h(c) \ \mid \ c \in S\} \\ \gamma_h(a) = \{c \ \mid \ h(c) \sqsubseteq_A a\} \end{array}$$

Practitioners often find it convenient to map each state in Σ_C to the state in Σ_A that best approximates it. Once function h is defined this way and proved a homomorphism, then it "lifts" into a Galois connection.

Of course, one recovers $h : \Sigma_C \rightarrow \Sigma_A$ from the Galois connection, (α, γ), by defining $h(c) = \alpha\{c\}$.

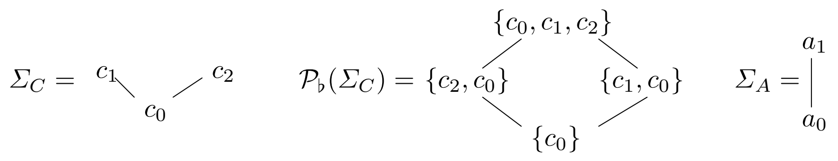
Galois connections on lower powerdomains Stepwise abstractions might require that Σ_C be nontrivially partially ordered and that its ordering be incorporated in the construction of $\mathcal{P}(\Sigma_C)$. In such a case, we suggest employing the *lower powerdomain* construction, $\mathcal{P}_\flat$ [13]: For a complete partially ordered set, $(P, \sqsubseteq_P)$,

$$\begin{array}{l} \mathcal{P}_\flat(P, \sqsubseteq_P) = (\{ \mathit{ScottClosure}(S) \ \mid \ S \subseteq P, S \neq \{\} \ \}, \subseteq) \\ \text{where } \mathit{ScottClosure}(S) = \downarrow S \cup \{\sqcup C \ \mid \ C \subseteq S \text{ is a chain}\} \\ \text{and } \downarrow S = \{c \ \mid \ \text{exists } c' \in S, c \sqsubseteq_P c'\} \end{array}$$

(Recall that a *chain* is a sequence, $c_0 \sqsubseteq_P c_1 \sqsubseteq_P \cdots \sqsubseteq_P c_i \sqsubseteq_P \cdots$.) The elements of the lower powerdomain are nonempty *Scott-closed* sets, which are those nonempty subsets of P that are closed downwards and closed under least-upper bounds of chains. An example of a lower powerdomain and its appearance in a Galois connection appears in Figure 7.

When using a lower powerdomain with the constructions in this section, we require that a homomorphism, $h : \Sigma_C \rightarrow \Sigma_A$, be a *Scott-continuous function* (that is, it preserves limits of chains: $h(\sqcup C) = \sqcup_{c \in C} h(c)$ for all chains, $C \subseteq \Sigma_C$).

Also, all the set-theoretic expressions in this section must be understood as Scott-closed sets—as necessary, apply *ScottClosure* to a set to make it Scott-closed.



$$\alpha : \mathcal{P}_\flat(\Sigma_C) \to \Sigma_A \qquad \alpha(S) = \begin{cases} a_0, & \text{if } S = \{c_0\} \\ a_1, & \text{otherwise} \end{cases}$$

$$\gamma : \Sigma_A \to \mathcal{P}_\flat(\Sigma_C) \qquad \gamma(a_0) = \{c_0\} \qquad \gamma(a_1) = \{c_0, c_1, c_2\}$$

Fig. 7. Lower powerdomain and Galois connection

7 Properties of Binary Relations

The previous sections showed that there are natural binary relations that underlie homomorphisms and Galois connections: In the case of a homomorphism, $h : \Sigma_C \to \Sigma_A$, we extract the relation, $\mathcal{R}_h \subseteq \Sigma_C \times \Sigma_A$:

$$c \; \mathcal{R}_h \; a \text{ iff } h(c) \sqsubseteq_A a$$

and in the case of a Galois connection, $(\alpha : \mathcal{P}(\Sigma_C) \to \Sigma_A, \gamma : \Sigma_A \to \mathcal{P}(\Sigma_C))$, we can extract the relation, $\mathcal{R}_\gamma \subseteq \Sigma_C \times \Sigma_A$:

$$c \; \mathcal{R}_\gamma \; a \text{ iff } c \in \gamma(a)$$

In both cases, the intuition behind $c \; \mathcal{R} \; a$ is that a is an approximation of c or that c is a possible refinement of a.

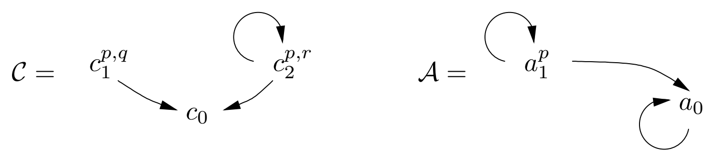
It is worthwhile to "disassemble" these and other binary relations and examine the properties that make the relations well behaved. Initial efforts in this direction were taken by Hartmanis and Stearns [15], Mycroft and Jones [26], Cousot and Cousot [9], Schmidt [33], and Loiseaux, et al. [19].[10] In the developments that follow, we employ the usual monotone Kripke structures, $\mathcal{C} = \langle \Sigma_C, \to_C, \mathcal{I}_C \rangle$ and $\mathcal{A} = \langle \Sigma_A, \to_A, \mathcal{I}_A \rangle$.

Here are four fundamental properties on relations on partially ordered sets:

Definition 10. *For partially ordered sets P and Q, a binary relation, $\mathcal{R} \subseteq P \times Q$, is*

- L-closed *("lower closed") iff for all $p \in P$, $q \in Q$, $p \; \mathcal{R} \; q$ and $p' \sqsubseteq_P p$ imply $p' \; \mathcal{R} \; q$*
- U-closed *("upper closed") iff for all $p \in P$, $q \in Q$, $p \; \mathcal{R} \; q$ and $q \sqsubseteq_Q q'$ imply $p \; \mathcal{R} \; q'$*
- G-closed *("greatest-lower-bound closed") iff for all $p \in P$, $\sqcap\{q \mid p \; \mathcal{R} \; q\}$ exists, and $p \; \mathcal{R} \; (\sqcap\{q \mid p \; \mathcal{R} \; q\})$*

[10] P. Cousot has remarked that the groundwork was laid by Shmuelli, but the appropriate references have not been located as of this date.



$\mathcal{R} = \{(c_0, a_0), (c_0, a_1), (c_1, a_1), (c_2, a_1)\}$

Fig. 8. Binary simulation relation

– inclusive *iff for all chains* $\{p_i\}_{i \geq 0} \subseteq P$, $\{q_i\}_{i \geq 0} \subseteq Q$, *if for all* $i \geq 0$, $p_i \mathcal{R} q_i$, *then both* $\sqcup_{i \geq 0} p_i$ *and* $\sqcup_{i \geq 0} q_i$ *exist, and* $\sqcup_{i \geq 0} p_i \mathcal{R} \sqcup_{i \geq 0} q_i$.

L- and U-closure define monotonicity and antimonotonicity properties of the binary relation [26]. G-closure states that the relation determines a function from P to Q,[11] and inclusive-closure states that the relation is Scott-continuous.

For the concrete and abstract state sets in Figure 7, Figure 8 displays two structures that use these state sets and defines a binary relation between them. One can readily verify that the binary relation in the Figure is a simulation that is LUG-(and trivially inclusive)-closed. Homomorphisms and Galois connections supply exactly these properties:

Proposition 11. *Let* $h : \Sigma_C \to \Sigma_A$ *be a function, and define* $c \mathcal{R}_h a$ *iff* $h(c) \sqsubseteq_A a$. *Then,*

1. $\mathcal{R}_h$ *is UG-closed;*
2. *if* h *is monotonic, then* $\mathcal{R}_h$ *is L-closed;*
3. *if* h *is Scott-continuous, then* $\mathcal{R}_h$ *is inclusive-closed.*

Proposition 12. *Let* $(\alpha : \mathcal{P}(\Sigma_C) \to \Sigma_A, \gamma : \Sigma_A \to \mathcal{P}(\Sigma_C))$ *be a Galois connection, and define* $c \mathcal{R}_\gamma a$ *iff* $c \in \gamma(a)$. *Then* $\mathcal{R}_\gamma$ *is LUG-inclusive-closed.*

Momentarily, we will see that these properties let one lift a binary relation into a function or Galois connection, but first we examine the properties one by one and learn their value to proving simulations.

First, we must emphasize that *the properties in Definition 10 do not ensure that a relation is a simulation.* Here is a trivial counterexample:

$$\mathcal{C} = \langle \{c_0\}, \{c_0 \longrightarrow c_0\}, \mathcal{I}_C \rangle$$
$$\mathcal{A} = \langle \{a_0\}, \{\}, \mathcal{I}_A \rangle$$

[11] Cousot and Cousot [9] note that L-closure and U-closure can be viewed as duals, meaning there is a dual to G-closure, which requires existence of least-upper bounds; this dual (and the dual to inclusivity) will not be developed here.

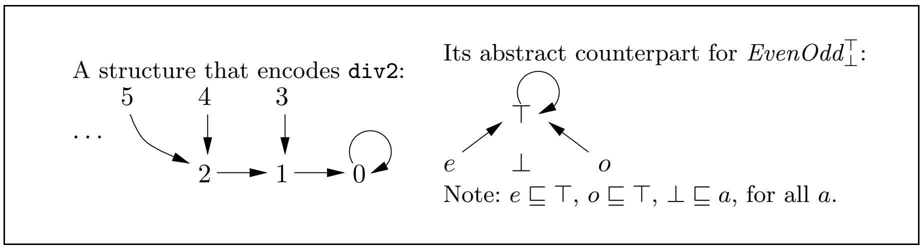

A structure that encodes div2:
5
4
3
. . .
2
1
0
Its abstract counterpart for $EvenOdd_\bot^\top$:
$\top$
e
$\bot$
o
Note: $e \sqsubseteq \top$, $o \sqsubseteq \top$, $\bot \sqsubseteq a$, for all a.

Fig. 9. Why U-closure is helpful for proving a simulation

The relation, $c_0\ \mathcal{R}\ a_0$, is LUG-inclusive-closed, but $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$ does *not* hold. Regardless of the Definition 10-properties of a relation, $\mathcal{R}$, we must perform the usual simulation proof with the relation.[12]

So, we must ask: How do the properties in Definition 10 aid us? To begin, consider an arbitrary relation, $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$, where we hope to prove that $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$. The guiding intuition behind $c\ \mathcal{R}\ a$ is that a can act as an approximation of c and that c can act as a refinement of a.

If it is the case that Σ_A is nontrivially partially ordered, the simulation proof will almost certainly require that $\mathcal{R}$ be U-closed, as suggested by the example in Figure 9. Within its transitions, the Figure's first structure encodes the `div2` operation on the natural numbers, and similarly, the second structure encodes the abstraction of `div2` on the tokens e ("even"), o ("odd'), and $\top$ ("any value"). To complete the proof that the second structure simulates the first, the obvious binary relation, $2n\ \mathcal{R}\ e$, $2n+1\ \mathcal{R}\ o$, must be made U-closed by adding $n\ \mathcal{R}\ \top$, for all $n \in \mathit{Nat}$.

We have this simple but useful property:

Proposition 13. *If $\mathcal{R}$ is U-closed, then $\gamma_{\mathcal{R}}(a) = \{c \mid c\ \mathcal{R}\ a\}$ is monotonic.*

The intuition behind $\gamma_{\mathcal{R}}(a)$ is that it identifies the elements in Σ_C that are represented by $a \in \Sigma_A$. Monotonicity ensures that, as abstract states become less precise in $\sqsubseteq_A$, they represent more and more concrete states in Σ_C.

In an obvious, dual way, if one considers the refinement of a structure, $\mathcal{A}$, into an implementation, $\mathcal{C}$, where Σ_C is nontrivially partially ordered, then L-closure is likely to be needed. Figure 10 gives one example, where the abstract state, a_1, is refined into the pair of states, c_1 and c_2, such that $c_1 \sqsubseteq_C c_2$. The refinement directs that the simulation relation be $c_0\ \mathcal{R}\ a_0$, and $c_i\ \mathcal{R}\ a_1$, for $i \in 1..2$. This relation is L-closed, and crucially so, in order to prove the simulation.

Proposition 14. *If $\mathcal{R}$ is L-closed, then $\beta_{\mathcal{R}}(c) = \{a \mid c\ \mathcal{R}\ a\}$ is antimonotonic.*

The intuition is that $\beta_{\mathcal{R}}(c)$ identifies the states in Σ_A that might approximate c. The antimonotoncity result ensures that the more precisely defined concrete states possess smaller, "more precise" sets of possible approximations from Σ_A.

[12] But a relation that lifts to a Galois connection *defines* a simulation that is sound and complete.

Refine a_1 into c_1 and c_2, where $c_1 \sqsubseteq_C c_2$:

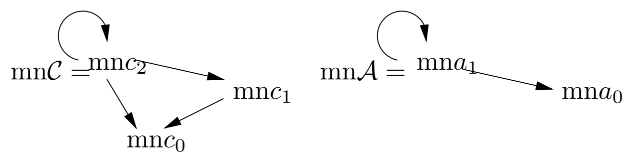


Fig. 10. Why L-closure is helpful for proving a simulation

The $\beta_{\mathcal{R}}$ map gives good intuition about the behaviour of $\mathcal{R}$, but practitioners desire to map a concrete state, $c \in \Sigma_C$ to the "most precise" state that approximates it. G-closure yields these important results:

Proposition 15. *For partially ordered sets P and Q, and binary relation, $\mathcal{R} \subseteq P \times Q$, define $\hat{\beta}_{\mathcal{R}}(c) = \sqcap\{a \mid c\,\mathcal{R}\,a\}$:*

1. *if $\mathcal{R}$ is G-closed, then $\hat{\beta}_{\mathcal{R}} : P \to Q$ is a well-defined function;*
2. *if $\mathcal{R}$ is LG-closed, then $\hat{\beta}_{\mathcal{R}}$ is monotonic;*
3. *if $\mathcal{R}$ is LG-inclusive-closed, then $\hat{\beta}_{\mathcal{R}}$ is Scott-continuous.*

This tells us that a G-closed relation identifies, for each $c \in \Sigma_C$, the element in Σ_A that best approximates it, namely, $\sqcap\{a \mid c\,\mathcal{R}\,a\}$.

Corollary 16. *If $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$ is G-closed and $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$, then $\hat{\beta}_{\mathcal{R}}$ is a homomorphism from Σ_C to Σ_A.*

The interaction of U- and G-closure brings us to Galois connections. For $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$, define $\gamma_{\mathcal{R}} : \Sigma_A \to \mathcal{P}(\Sigma_C)$ and $\alpha_{\mathcal{R}} : \mathcal{P}(\Sigma_C) \to \Sigma_A$ as follows:

$$\begin{aligned}&\gamma_{\mathcal{R}}(a) = \{c \mid c\,\mathcal{R}\,a\}\\ &\alpha_{\mathcal{R}}(S) = \sqcup_{c \in S}\hat{\beta}_{\mathcal{R}}(c)\\ &\quad \text{where } \hat{\beta}_{\mathcal{R}}(c) = \sqcap\{a \mid c\,\mathcal{R}\,a\}\end{aligned}$$

Theorem 17. *1. If $\mathcal{R}$ is UG-closed, then $(\alpha_{\mathcal{R}}, \gamma_{\mathcal{R}})$ form a Galois connection between $\mathcal{P}(\Sigma_C)$ and Σ_A;*
2. If $\mathcal{R}$ is LUG-inclusive-closed and Σ_C is nontrivially partially ordered, then $(\alpha_{\mathcal{R}}, \gamma_{\mathcal{R}})$ form a Galois connection between $\mathcal{P}_\flat(\Sigma_C)$ and Σ_A.

Since Galois connections synthesize simulations that are sound and complete in the sense of Section 6, we conclude that *UG- (or LUG-)closed relations are best for abstraction studies.* If G-closure is impossible, then soundness can be ensured by U- (or LU-)closure, as noted by Mycroft and Jones [26] and Cousot and Cousot [9].

8 Conclusion

We have seen how standard approaches for defining abstractions of program models can be understood in terms of structure-preserving binary relations. The crucial topics that follow the present work are: *(i)* the *extraction* of properties from an abstract Kripke structure and *(ii)* the *application* of properties to transformation of the corresponding concrete Kripke structure. Topic (i) can be best understood as validating temporal-logic properties by model checking [34–37]; Topic (ii) is surprisingly underdeveloped, although Lacey, Jones, Van Wyk, and Frederiksen [18] and Cousot and Cousot [11] give recent proposals.

Acknowledgements: I thank Neil Jones for many years of inspiring work. The current paper evolved from research presented at the Workshop on Refinement and Abstraction, Amagasaaki, Japan, in 1999. I thank Yoshiki Kinoshita and the other workshop organizers for their encouragement. Anindya Banerjee, Michael Huth, Radha Jagadeesan, Markus Müller-Olm, and Hiroshi Watanabe supplied useful comments and corrections.

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison Wesley, 1986.
2. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. Computer-Aided Verification 2000*, Lecture Notes in Computer Science. Springer, 2000.
3. E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.
4. E.M. Clarke, O. Grumberg, and D.E. Long. Verification tools for finite-state concurrent systems. In J.W. deBakker, W.-P. deRoever, and G. Rozenberg, editors, *A Decade of Concurrency: Reflections and Perspectives*, number 803 in Lecture Notes in Computer Science, pages 124–175. Springer, 1993.
5. 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.
6. P. Cousot, editor. *Static Analysis, 8th International Symposium.* Lecture Notes in Computer Science 2126, Springer, Berlin, 2001.
7. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs. In *Proc. 4th ACM Symp. on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 269–282. ACM Press, 1979.
9. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
10. P. Cousot and R. Cousot. Higher-order abstract interpretation. In *Proc. IEEE Int'l. Conf. Programming Languages.* IEEE Press, 1994.
11. P. Cousot and R. Cousot. Systematic design of program transformations by abstract interpretation. In *Proc. 29th ACM Symp. on Principles of Prog. Languages.* ACM Press, 2002.
12. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM TOPLAS*, 19:253–291, 1997.

13. C. Gunter. *Semantics of Programming Languages.* MIT Press, Cambridge, MA, 1992.
14. D. Harel. Statecharts: a visual formalization for complex systems. *Science of Computer Programming*, 8, 1987.
15. J. Hartmanis and R. Streans. Pair algebras and their application to automata theory. *Information and Control*, 7:485–507, 1964.
16. M. Hecht. *Flow Analysis of Computer Programs.* Elsevier, 1977.
17. N. Jones and F. Nielson. Abstract interpretation: a semantics-based tool for program analysis. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 4*, pages 527–636. Oxford Univ. Press, 1995.
18. D. Lacey, N.D. Jones, E. Van Wyk, and C. Frederiksen. Proving correctness of compiler optimizations by temporal logic. In *Proc. 29th ACM Symp. on Principles of Prog. Languages.* ACM Press, 2002.
19. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–44, 1995.
20. C. McGowan. An inductive proof technique for interpreter equivalence. In R. Rustin, editor, *Formal Semantics of Programming Languages*, pages 139–148. Prentice-Hall, 1972.
21. A. Melton, G. Strecker, and D. Schmidt. Galois connections and computer science applications. In *Category Theory and Computer Programming*, pages 299–312. Lecture Notes in Computer Science 240, Springer-Verlag, 1985.
22. R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag, Lecture Notes in Computer Science 92, 1980.
23. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
24. J.C. Mitchell. *Foundations for Programming Languages.* The MIT Press, Cambridge, Massachusetts, 1996.
25. M. Müller-Olm, D.A. Schmidt, and B. Steffen. Model checking: A tutorial introduction. In G. Filé and A. Cortesi, editors, *Proc. 6th Static Analysis Symposium.* Springer LNCS, 1999.
26. A. Mycroft and N.D. Jones. A relational framework for abstract interpretation. In *Programs as Data Objects*, pages 156–171. Lecture Notes in Computer Science 217, Springer-Verlag, 1985.
27. F. Nielson. Two-level semantics and abstract interpretation. *Theoretical Computer Science*, 69(2):117–242, 1989.
28. F. Nielson and H. R. Nielson. *Two-Level Functional Languages.* Cambridge University Press, 1992.
29. F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer Verlag, 1999.
30. D. Park. Concurrency and automata in infinite strings. Lecture Notes in Computer Science 104, pages 167–183. Springer, 1981.
31. G. D. Plotkin. Lambda-definability in the full type hierarchy. In J. Seldin and J. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–374. Academic Press, 1980.
32. J. Reynolds. Types, abstraction, and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing '83*, pages 513–523. North-Holland, Amsterdam, 1983.
33. D.A. Schmidt. Natural-semantics-based abstract interpretation. In A. Mycroft, editor, *Static Analysis Symposium*, number 983 in Lecture Notes in Computer Science, pages 1–18. Springer-Verlag, 1995.

34. D.A. Schmidt. Data-flow analysis is model checking of abstract interpretations. In *Proc. 25th ACM Symp. on Principles of Prog. Languages*. ACM Press, 1998.
35. D.A. Schmidt. Binary relations for abstraction and refinement. *Workshop on Refinement and Abstraction, Amagasaki, Japan, Nov. 1999. Elsevier Electronic Notes in Computer Science*, to appear.
36. D.A. Schmidt and B. Steffen. Data-flow analysis as model checking of abstract interpretations. In G. Levi, editor, *Proc. 5th Static Analysis Symposium*. Springer LNCS 1503, 1998.
37. B. Steffen. Generating data-flow analysis algorithms for modal specifications. *Science of Computer Programming*, 21:115–139, 1993.
38. B. Steffen. Property-oriented expansion. In R. Cousot and D. Schmidt, editors, *Static Analysis Symposium: SAS'96*, volume 1145 of *Lecture Notes in Computer Science*, pages 22–41. Springer-Verlag, 1996.