Binary Relations for Abstraction and Refinement

David A. Schmidt [1]

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

Abstract

By employing Kripke structures as a common framework for system specifications, implementations, and abstractions, we study the standard means for relating a specification to its refinement and for relating an implementatation to its abstraction. The classic tools of homomorphism and Galois connection are dissasembled and characterized in terms of binary simulation relations that possess desirable structural properties.

Because specifications, implementations, and abstractions possess logical properties as well, we study sound subsets of temporal logic (more specifically, modal mu-calculus) that can be used for stating necessarily-true propositions and possibly-true propositions about specifications and abstractions. By extending Kripke structures to modal-transition systems, we are able to employ full modal mu-calculus as a sound logic for necessarily- and possibly-true propositions, and we can characterize a modal-transition system by the logical propositions that hold true for it.

Most of the paper's technical development is scattered throughout the research literature, and the paper's main contribution is assembling the technical material into a coherent, useful methodology for system refinement and abstraction.

1 Introduction

A software system must be specified, then implemented, then analyzed. Working from the specification to the implementation is called *refinement*; converting the implementation into a format suitable for analysis is called *abstraction*.

[1] Ph: +1-785-532-6350. `schmidt@cis.ksu.edu`. Supported by NSF CCR-9633388, CCR-9970679, INT-9981558, and NASA NAG-2-1209.

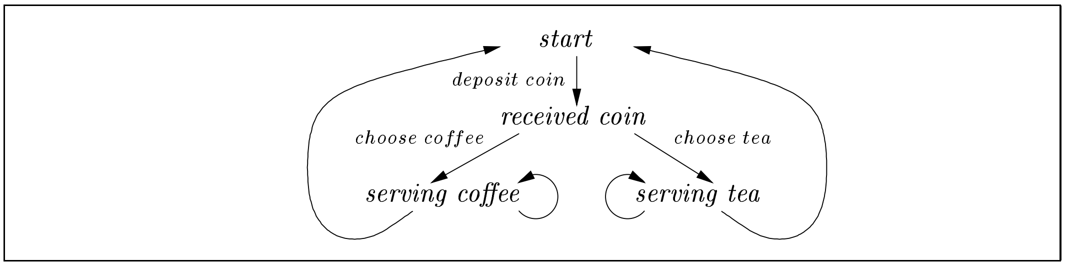
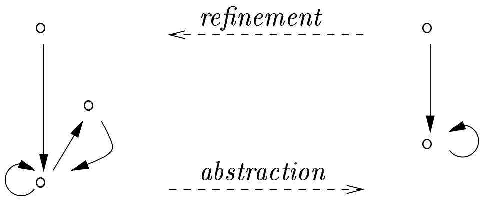


Fig. 1. A specification of a vending machine in state-diagram style

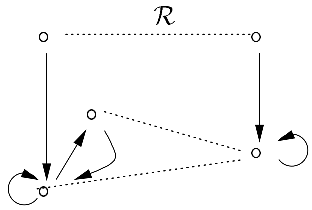
In an ideal world, the same format that one uses to write a specification should be used to express its refinement also, and this same format should also be acceptable for deriving an implementation's abstraction. In practice, this is rarely the case: Specifications are written with class diagrams or sequence diagrams or logical pre- and post-conditions; and implementations are sequences of source code. Abstractions tend to be control-flow graphs, program-dependency graphs, or interprocedural block diagrams. But it *is* possible to restrict specifications, implementations, and abstractions so that just behavioral properties are expressed; this gives a common, automaton-like, format.

In this paper, we consider program specifications that are written as state diagrams [17] or statecharts [19]; see Figure 1 for an example. Implementations are expressed as automata (or as "trees" of all possible execution traces), and abstractions are flowcharts or flow graphs (e.g., Figure 4, seen later in this paper). All these formats can be expressed in the framework of a (*labelled*) *Kripke structure* [5,36,44]. By using the Kripke-structure framework, we can consider refinement and abstraction as transformations from one Kripke structure to another. Indeed, the processes can be viewed as duals:



The previous diagram makes us wonder about the formal relationship between one structure and its refinement (or, its abstraction). This paper considers two classical tools for relating structures—homomorphisms and Galois connections—and explains in what formal sense a homomorphism or Galois connection establishes that one structure refines/abstracts another. The underlying notion of *simulation* of one structure's transitions by another's is exposed as the central notion: A structure "abstracts" an implementation if the former simulates the moves of the latter. Dually, a structure "refines" a specification if the latter simulates the former (that is, the refinement is more deterministic than its specification).

Both a homomorphism and a Galois connection define a binary relation, $\mathcal{R}$, between the states (nodes) of the two structures that are being related:



We spend some effort formalizing crucial properties of these binary relations and show how such properties help prove existence of simulations. Also, we characterize homomorphisms and Galois connections in terms of these properties.

Constructions like homomorphisms and Galois connections define structural relationships between two Kripke structures, but one must not forget that a practitioner cares about *logical properties* as well—indeed, a specification is sometimes written as a set of logical propositions! In the case of a specification expressed as a Kripke structure, we demand a logic that states logical properties of the Kripke structure, and we demand that a *refinement preserves the logical properties*. Dually, we expect that a Kripke structure that is an abstraction of an implementation possess only those logical properties that hold true of the implementation as well.

What is the appropriate logic for Kripke structures? This paper pursues temporal logic [5,44]. Perhaps surprisingly, the temporal logics in common use must be restricted in significant ways to ensure the desirable behavior stated in the previous paragraph. We survey standard approaches for restricting temporal logic and discover two possible subsets of temporal logic—one for stating properties that *necessarily* hold true of a refinement (or implementation) and one that states properties that *possibly* hold true.

Neither logic is entirely satisfactory, and we repair the situation by proposing an improvement of Kripke structure into *mixed* and then *modal* transition systems. Along with an improved notion of simulation, we find that the standard temporal logic can be used to express logical properties of specifications, implementations, and abstractions, even to the point that the logical propositions that hold true for a modal-transition system *characterize* the system (up to an equivalence).

Most of the technical development that supports the above is scattered throughout the research literature; the contribution of this paper is to assemble the technical material into a whole, coherent, useful methodology for refinement and abstraction.

The paper's structure goes as follows: Section 2 defines Kripke structures and gives examples of abstractions and refinements of them by means of data- and control-transformation techniques. Next, Section 3 shows how one might relate one Kripke structure to another by means of homomorphisms, Galois connections, and finally, by means of a binary relation between the states of the two structures. The notion of "simulation" is defined and applied to the relational forms.

Finally, Section 4 introduces the modal mu-calculus as the temporal logic for expressing properties of Kripke structures. Examples of modal-mu-calculus propositions are used to motivate why the full logic cannot be used to express properties of refinements and abstractions, so the "necessarily" and "possibly" subsets of the logic are introduced. Then, incremental extensions to these sublogics return us to the full modal mu-calculus, in the process, requiring the extension of Kripke structure into mixed- and modal- transition systems, where a structure's transitions are labelled with "modalities" that state whether the transition *may* be preserved in a refinement or *must* be preserved. The characterization of a modal-transition system in terms of the propositions that hold true for it concludes the paper.

2 Kripke Structures

We use *Kripke structures* to represent the variety of specifications, data-flow analysis models, state charts, etc., that arise in abstraction and refinement studies. Crudely stated, a Kripke structure is a nondeterministic finite-state automaton where each state is labelled with atomic, primitive "properties" that "hold true" at it [5,36]. The formalization is

Definition 1 *A* Kripke structure *is a triple,* $\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$*, where*

- Σ_K *is a set of states.*
- $\rightarrow_K \subseteq \Sigma_K \times \Sigma_K$ *is the transition relation. We write* $s \longrightarrow s'$ *to denote* $(s, s') \in \rightarrow_K$*. Futher, we require* $\rightarrow_K$ *has* finite image*: for every* $s \in \Sigma_K$*, the set* $\{s' \mid s \longrightarrow s'\}$ *is finite.*
- $\mathcal{I}_K : \Sigma_K \rightarrow \mathcal{P}(Atom)$ *associates a set of atomic properties,* $\mathcal{I}_K(s) \subseteq Atom$*, to each* $s \in \Sigma_K$*.*

Figure 2 defines a Kripke structure and draws it as a finite-state automaton whose states are labelled by their atomic properties. The structure describes the state chart of a process that requests ownership and use of a resource. Such a structure is a kind of specification that can be refined into an implementation; the refinement might itself be a Kripke structure.

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

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

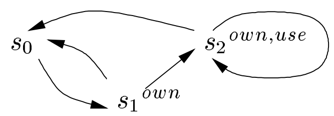


Fig. 2. Kripke structure of a process that uses a resource

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

$$\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K\rangle$$

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

Fig. 3. Infinite-state Kripke structure of a sequential program

A Kripke structure might be written so that its transitions, $s \longrightarrow s'$, are labelled, e.g., $s \xrightarrow{\ell} s'$; Figure 1 shows such an example. For simplicity, we will omit labelled transitions from the formal development in this paper, but they can be included with no harm to any of the technical results that will follow [28,35,44].

A source program's semantics can also be depicted as a Kripke structure; Figure 3 defines the 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.

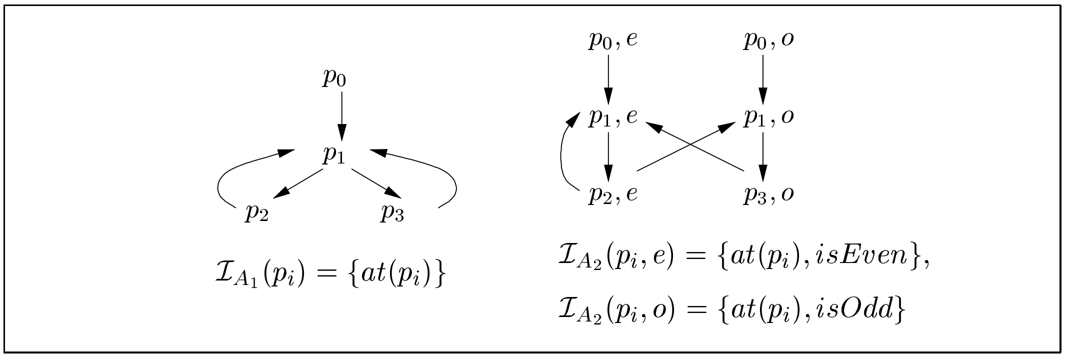


Fig. 4. Two abstractions of a program

2.0.1 Abstractions of a Kripke structure

Because of its infinite state set, the structure defined in Figure 3 cannot be conveniently drawn pictorially nor can it be easily analyzed mechanically. For these reasons, we might try to simplify or "abstract" the structure into a manageably sized, finite-state Kripke structure. Figure 4 shows two possible abstractions, represented pictorially. The first abstraction, traditionally called *control abstraction*, defines $\Sigma_{A_1} = \mathit{ProgramPoint}$—data values are forgotten during structure generation. Control abstraction generates the classic control-flow graph used by a compiler, and the classic *data-flow analysis* is a technique for attaching data information to the states of the control-flow graph [1,20].

The second abstraction is a mixture of data and control abstraction: It simplifies the program's data domain (namely, `x`'s value) to $\mathit{EvenOdd} = \{e, o\}$ and defines $\Sigma_{A_2} = \mathit{ProgramPoint} \times \mathit{EvenOdd}$. The domain generates a structure that embeds both control and data information into the graph, "unfolding" the program (that is, some program points are replicated). In consequence, more precise knowledge is presented about the outcomes of the test at program point p_1 (cf. Steffen's "property oriented abstraction" [43]).

To support our claim that the structures in Figure 4 are abstractions of the structure in Figure 3, we must formally relate the two Figures. The means for doing so are explained in the next section, but more machinery is needed first.

2.0.2 Data abstraction

The second structure in Figure 4 was generated by abstracting the set, *Nat*, of natural numbers into the simpler set, $\mathit{EvenOdd} = \{e, o\}$, such that even-valued naturals are represented by e and odd-valued naturals are represented by o. The addition and division-by-two operations must be restated to operate on the new set:

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

By necessity, the semantics of division-by-two is nondeterministic.

The data abstraction and its operations are inserted into the definition of $\rightarrow_K$, generating $\rightarrow_{A_2}$. In particular, we have these crucial transition rules for the decision point at p_1:

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

which let us retain crucial information about the properties of the data that arrive at program points p_2 and p_3.

The even-odd data abstraction generates a finite set of states for the Kripke structure in Figure 3. Of course, the finite cardinality might still be too huge for practical analysis. In such a case, more severe abstraction is required. For this, one can partially order the data domain. The partial ordering is used to merge similar states and so reduce the cardinality of the state set. For example, we might add the data values, $\perp$ ("no value at all") and $\top$ ("any value at all") to the *EvenOdd* set and partially order the result as follows:

$$\mathit{EvenOdd}_{\perp}^{\top} = \begin{array}{ccccc} & & \top & & \\ & / & & \backslash & \\ e & & & & o \\ & \backslash & & / & \\ & & \perp & & \end{array}$$

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, $\perp$ is the most "precise" of all, because it precisely notes no value at all.)

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. 5. Kripke structure generated with partially ordered data domain

(Operations on $\bot$ are ignored.) In particular, division by 2 produces $\top$ as its result.

Now, program states have the form, $\mathit{ProgramPoint} \times \mathit{EvenOdd}_\bot^\top$, where $p, \bot$ 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:

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

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

We use $\mathit{EvenOdd}_\bot^\top$ with the above transition rules for p_1 to generate a new Kripke structure for the example in Figure 3; see Figure 5. The resulting structure has one fewer state than its counterpart in Figure 4.

Finally, note that the following transition rules for state p_1 would also be acceptable (but they lose too much precision):

$$p_1, \top \longrightarrow p_2, \top$$

$$p_1, \top \longrightarrow p_3, \top$$

2.0.3 Partial orderings that respect structure

The structure in Figure 5 has two "related" states: $(p_1, \top)$ and (p_1, e) are related because $e \sqsubseteq \top$. (For the example, we define $p_i, d_i \sqsubseteq_K p_j, d_j$ iff $p_i = p_j$ and $d_i \sqsubseteq_{\mathit{EvenOdd}_\bot^\top} d_j$.)

An expected consequence of this relationship should be that, if $p_1, e \longrightarrow q$ then $p_1, \top \longrightarrow q'$, where $q \sqsubseteq_K q'$, also—the less precise state should be consistent with the more precise one. This notion is so crucial to proofs of correctness of abstraction and refinement that we name it specially:

Definition 2 *For Kripke structure,* $\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$ *and partial ordering* $\sqsubseteq_K \subseteq \Sigma_K \times \Sigma_K$*,* $\sqsubseteq_K$ reflects $\rightarrow_K$ *iff for all* $s, s' \in \Sigma_K$*,* $s \sqsubseteq_K s'$ *and* $s \longrightarrow s_1$ *implies there exists* $s_2 \in \Sigma_K$ *such that* $s' \longrightarrow s_2$ *and* $s_1 \sqsubseteq_K s_2$*:*

$$\begin{array}{ccc} s & \sqsubseteq_K & s' \\ \downarrow & & \downarrow \\ s_1 & \sqsubseteq_K & s_2 \end{array}$$

Of course, we might encode the transition information in a dual fashion, although this is rarely seen in practice: *For Kripke structure,* $\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$ *and partial ordering* $\sqsubseteq_K \subseteq \Sigma_K \times \Sigma_K$*,* $\sqsubseteq_K$ preserves $\rightarrow_K$ *iff for all* $s, s' \in \Sigma_K$*,* $s \sqsubseteq_K s'$ *and* $s' \longrightarrow s_2$ *implies there exists* $s_1 \in \Sigma_K$ *such that* $s \longrightarrow s_1$ *and* $s_1 \sqsubseteq_K s_2$. To keep the paper manageable, we will employ only those examples where $\sqsubseteq_K$ reflects $\rightarrow_K$.

In the next section, we will see that transition reflection and preservation are examples of *simulations*.

A partial ordering on states might respect the states' atomic properties in a similar way. There are two ways of doing this.

First, we can reasonably argue that whatever atomic properties are *necessarily true* of $(p_1, \top)$ should also be necessarily true of (p_1, e), because the latter is more precise than the former. But occasionally, the dual approach is argued: because it is less precise, $(p_1, \top)$ has more atomic properties that are *possibly true* of it than does (p_1, e). This generates two dual, useful notions:

Definition 3 *For Kripke structure,* $\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$ *and partial ordering* $\sqsubseteq_K \subseteq \Sigma_K \times \Sigma_K$*,*

(1) $\sqsubseteq_K$ reflects $\mathcal{I}_K$ *iff* $s \sqsubseteq_K s'$ *implies* $\mathcal{I}_K(s) \supseteq \mathcal{I}_K(s')$*.*
(2) $\sqsubseteq_K$ preserves $\mathcal{I}_K$ *iff* $s \sqsubseteq_K s'$ *implies* $\mathcal{I}_K(s) \subseteq \mathcal{I}_K(s')$*.*

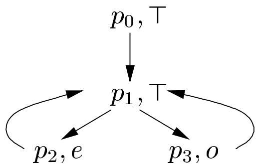
In practice, it is common to see a partial ordering, $\sqsubseteq_K$, reflect both $\rightarrow_K$ and $\mathcal{I}_K$, but later in this paper we encounter other situations.

2.0.4 Control abstraction by state merging

As noted earlier, the abstraction that sets $\Sigma_K = \mathit{ProgramPoint}$ is known as *control abstraction*. But control abstraction can also merge the states of an existing Kripke structure in some consistent way.

Here is an example: We can generate a smaller structure from Figure 5 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 $(p_1, \top)$. The resulting structure looks like this:



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 $(p_k, d_i \sqcup d)$.

If this approach is taken with the source program in Figure 3 and the data-flow-information domain, $EvenOdd_{\perp}^{\top}$, the resulting structure is exactly that seen above.

2.0.5 Refinements of a Kripke structure

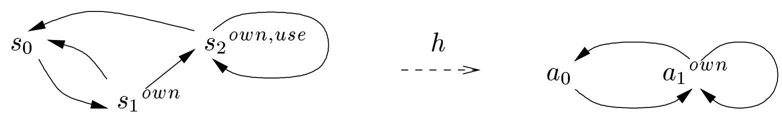
In our formulation, refinement is a dual notion to abstraction, so one might "reverse" the developments in the previous subsections to employ "data refinement" and "control refinement" to a specification written as a Kripke structure.

3 Relating Models

Abstraction and refinement are dual processes, but they both share the problem of relating a "detailed" structure with a "less detailed" counterpart. We call the detailed structure the *concrete structure* (i.e., the "implementation" or "refinement") and name it $\mathcal{C}$, and we call the less detailed structure the *abstract structure* (i.e., the "abstraction" or "specification") and name it $\mathcal{A}$.

3.1 Homomorphisms

The classic tool for relating one structure to another is the homomorphism, which maps a state from a concrete structure into the state in the abstract



$h(s_0) = a_0,\ h(s_1) = h(s_2) = a_1$

Fig. 6. Example homomorphism

structure that "models" it:

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* homomorphism from $\mathcal{C}$ to $\mathcal{A}$ *is a function, $h : \Sigma_C \rightarrow \Sigma_A$, such that for all $c \in \Sigma_C$, $c \longrightarrow c'$ implies there exists a transition, $h(c) \longrightarrow a'$ such that $h(c') \sqsubseteq_A a'$:*

$$\begin{array}{ccc} c & \xrightarrow{h} & h(c) \\ \downarrow & & \downarrow \\ c' & \xrightarrow{h} & h(c') \sqsubseteq_A a' \end{array}$$

Figure 6 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. As noted in the previous section, data-flow analysis problems often require that Σ_A be nontrivially partially ordered.

For example, reconsider Figures 3 and 5, where Σ_C is $\mathit{ProgramPoint} \times \mathit{Nat}$, Σ_A is $\mathit{ProgramPoint} \times \mathit{EvenOdd}_{\bot}^{\top}$, and consider the transition, `x := x div 2`:

$$\begin{array}{ccc} p_2, 6 & \xrightarrow{h} & p_2, e \\ \texttt{x:= x div2} \downarrow & & \downarrow \texttt{x:= x div2} \\ p_1, 3 & \xrightarrow{h} & p_1, o \sqsubseteq p_1, \top \end{array}$$

where

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

If Σ_A is partially ordered by $\sqsubseteq_A$, then $\sqsubseteq_A$ should reflect $\rightarrow_A$. This ensures that the definition of homomorphism in Definition 4 is consistent with $\sqsubseteq_A$. (If

$\sqsubseteq_A$ preserves $\rightarrow_A$, then we might alter Definition 4 so that $a' \sqsubseteq h(c')$ holds instead—but we won't pursue this issue!)

Because of the $\mathcal{I}_C$ and $\mathcal{I}_A$ components within $\mathcal{C}$ and $\mathcal{A}$, respectively, a homomorphism should respect atomic properties as well. There are two obvious ways that this might be enforced:

Definition 5 *A homomorphism, h, from $\mathcal{C}$ to $\mathcal{A}$,*

- reflects properties *if* $\mathcal{I}_C(c) \supseteq \mathcal{I}_A(h(c))$, *for all* $c \in \Sigma_C$.
- preserves properties *if* $\mathcal{I}_C(c) \subseteq \mathcal{I}_A(h(c))$, *for all* $c \in \Sigma_C$.

In most abstraction and refinement case studies, homomorphism h reflects properties, because the abstract structure is used to validate desirable properties that must be *necessarily true* of the concrete structure when they are true of the abstract structure. When we employ such a property-reflecting homomorphism, we expect that $\sqsubseteq_A$ reflect $\mathcal{I}_A$ as well.

But a homomorphism that preserves properties can also be useful, because the abstract structure can then be analyzed for the presence of undesirable properties—if an undesirable property is validated on the abstract structure, then there is cause for concern, because the property might be *possibly true* of the concrete structure; if an undesirable property is refuted, then the property is not possibly true of the concrete structure. (And, we would expect that $\sqsubseteq_A$ preserves $\mathcal{I}_A$.)

The notions of "necessarily true" and "possibly true" are developed more completely within temporal logics in a later section of this paper.

Because homomorphisms "point" from the concrete structure to the abstract one, they are especially useful for data-flow analysis studies. Indeed, the abstract structure, $\mathcal{A}$, might be synthesized from $\mathcal{C}$ and a function, $h : \Sigma_C \rightarrow \Sigma_A$. The function is made into a homomorphism by defining

$$a \longrightarrow a' \in \rightarrow_A \text{ iff there exists } c \in h^{-1}(a), c' \in h^{-1}(a') \text{ such that } c \longrightarrow c'$$

and h is made property reflecting by defining

$$\mathcal{I}_A(a) = \cap\{\mathcal{I}_C(c) \mid h(c) = a, c \in \Sigma_C\}$$

(A property-preserving homomorphism is defined dually with set union. Note that we need not enforce a partial ordering on Σ_A, but if we desire one, an obvious choice is the preordering, $a \sqsubseteq_A a'$ iff $\mathcal{I}_A(a) \supseteq \mathcal{I}_A(a')$, quotiented to make it a partial ordering.)

Standards uses of homomorphisms to relate concrete and abstract structures can be found in papers by Clarke, Grumberg, and Long [7] and Clarke, et al. [4], among others.

3.2 Simulations

A homomorphism is a "structure preserving" mapping, but lying inside it is a more fundamental notion—to relate two Kripke structures, one must show that all the transitions in one structure are "mimicked" by the other. The formalization of this idea is that of *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 6 *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 $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$, if for every $c \in \Sigma_C$, $a \in \Sigma_A$,*

$$\begin{aligned} &\textit{if } c \, \mathcal{R} \, a \textit{ and } c \longrightarrow c', \textit{then there exists } a' \in \Sigma_A \textit{ such that} \\ &\quad a \longrightarrow a' \textit{ and } c' \, \mathcal{R} \, a' : \end{aligned}$$

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

In the above situation, we also say that $\mathcal{A}$ *simulates* $\mathcal{C}$, and we write $\mathcal{C} \lhd \mathcal{A}$ if $\mathcal{R}$ is unimportant. Of course, the intuition is that any transition in $\mathcal{C}$ can be mimicked by one in $\mathcal{A}$, and this mimicry can be extended to a sequence of transitions.

(Note: If labelled transitions are employed, as in Figure 1, the simulation must respect the labels: $c \, \mathcal{R} \, a$ and $c \xrightarrow{\ell} c'$ implies that $a \xrightarrow{\ell} a'$ and $c' \, \mathcal{R} \, a'$.)

If we reexamine Figure 6, 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 3 and its two even-odd abstractions in Figures 4 and 5, we see this simulation relates states in the concrete structure to those in the abstract structures:

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

In practice, we expect that simulations are *left total* and *right total*:

Definition 7 *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 $s\ \mathcal{R}\ t$.*

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 should respect the atomic properties of the related structures:

Definition 8 *If $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$ holds, then*

(1) $\mathcal{R}$ is property reflecting *if $c\ \mathcal{R}\ a$ implies $\mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)$, for all $c \in \Sigma_C$ and $a \in \Sigma_A$.*
(2) $\mathcal{R}$ is property preserving *if $c\ \mathcal{R}\ a$ implies $\mathcal{I}_C(c) \subseteq \mathcal{I}_A(a)$, for all $c \in \Sigma_C$ and $a \in \Sigma_A$.*

As with homomorphisms, a property-reflecting simulation sets the stage for validating properties of the abstract structure that are necessarily true of the concrete structure, and a property-preserving simulation lets us discover properties of the abstract structure that are possibly true of the concrete structure.

It is common to embed the requirements of property reflection and property preservation into the definition of simulation itself:

Proposition 9 *$\mathcal{R} \subseteq \mathcal{C} \times \mathcal{A}$ is a property-reflecting simulation iff for every $c \in \Sigma_C$, $a \in \Sigma_A$, if $c\ \mathcal{R}\ a$, then*

$$\mathcal{I}_C(c) \supseteq \mathcal{I}_A(a), and$$

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

Property-preserving simulations can be characterized similarly.

Simulations played a crucial role in equivalence proofs of interpreters defined by operational semantics definitions [32], where each execution step of an 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 vice versa). Mathematical induction justifies that every finite sequence of transitions taken within $\mathcal{C}$ can be mimicked by $\mathcal{A}$. Coinductive reasoning [39,35] extends the same proof concept to infinite sequences of transitions.

More recently, correctness proofs of hardware circuits, protocols, and systems of processes has been undertaken by means of simulations; see Milner [35] for a starting point.

Definition 2 states, for a structure, $\mathcal{K} = \langle \Sigma_K, \rightarrow_K, \mathcal{I}_K \rangle$, how a partial ordering, $\sqsubseteq_K \subseteq \Sigma_K \times \Sigma_K$, might reflect $\rightarrow_K$; based on Definition 6, we now realize this is a simulation: $\sqsubseteq_K$ reflects $\rightarrow_K$ iff $\mathcal{K} \lhd_{\sqsubseteq_K} \mathcal{K}$. If $\sqsubseteq_K$ reflects $\mathcal{I}_K$, then $\lhd_{\sqsubseteq_K}$ is a property-reflecting simulation

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.

When Σ_A has no partial ordering on it, the above relation degenerates to just $c \; \mathcal{R}_h \; a$ iff $h(c) = a$, and it is trivial to prove that $\mathcal{R}_h$ is a simulation. But when Σ_A is nontrivially partially ordered, we demand that $\sqsubseteq_A$ reflect $\rightarrow_A$, then it is straightforward to prove that $\mathcal{R}_h$ is a simulation.

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

Although a homomorphism is a powerful practical tool for defining simulations, 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 $i \geq 0$:

$$\begin{aligned}
\mathcal{R}_0 &= \Sigma_C \times \Sigma_A \\
\mathcal{R}_{i+1} &= \{(c, a) \mid \text{if } c \longrightarrow c', \text{then there exists } a' \in \Sigma_A \text{ such that} \\
&\qquad\qquad a \longrightarrow a' \text{ and } c' \; \mathcal{R}_i \; a'\}
\end{aligned}$$

Of course, $c \; \mathcal{R}_i \; a$ asserts that a mimicks c for up to i transitions. Therefore, we define the limit relation, $\mathcal{R}_\infty = \bigcap_{i \geq 0} \mathcal{R}_i$. As shown by Park [39] and Milner [34], 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 $\mathcal{R} \subseteq \mathcal{R}_\infty$. One can define a property-reflecting or property-preserving simulation, $\mathcal{R}_\infty$, in this way as well, by using the characterization of property-preserving (reflecting) simulation found in Proposition 9.

Alas, the synthesis technique cannot produce a left-total or right-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 represent 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 prove to be a useful foundation for reasoning about the relationships between structures, and we will use them frequently in the sequel. But we next consider another powerful practical tool for defining relationships between structures.

3.3 Galois connections

A homomophism is a clumsy tool for refinement, because it "points" in the "wrong" direction—an invertible mapping would be better. Also, because a homomorphism is a function, it gives incomplete insight about how one might perform "state merging"—that is, how one determines the appropriate abstract state that corresponds to a *set* of concrete states. (This proves useful in practice, both for conducting an analysis where execution might start in one of many possible concrete states and for studying "control abstraction," as described in the previous section, where a set of states are merged into one.)

These issues are addressed by generalizing from a homomorphism function to a Galois connection [9,10,24,33]:

Definition 10 *For partially ordered sets P and Q, a* Galois connection *is a pair of functions, $(\alpha: P \to Q, \gamma: Q \to P)$, such that for all $p \in P$ and $q \in Q$, $p \sqsubseteq_P \gamma(q)$ iff $\alpha(p) \sqsubseteq_Q q$:*

$$\begin{array}{ccc} \gamma(q) & \overset{\gamma}{\longleftarrow} & q \\ \sqcup|_P & & \sqcup|_Q \\ p & \overset{\alpha}{\longrightarrow} & \alpha(p) \end{array}$$

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 is a list of some of the more interesting.

Theorem 11 *Say that* $(\alpha: P \to Q, \gamma: Q \to P)$ *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.*

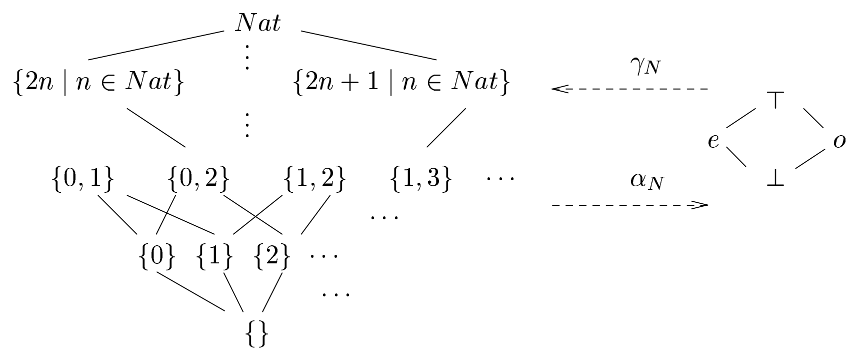
Proofs of these results can be found in [33] as well as in many other sources.

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 we have Kripke structures $\mathcal{C}$ and $\mathcal{A}$. The obvious way of using a Galois connection to relate the two structures is by constructing the complete lattices $(\mathcal{P}(\Sigma_C), \subseteq)$ and $(\Sigma_A, \sqsubseteq_A)$ and formulating α and γ. (For the moment, we assume that Σ_C has no partial ordering, or, we ignore it [11]. We assume that $\sqsubseteq_A$ is given, and for simplicity, we assume $(\Sigma_A, \sqsubseteq_A)$ is a complete lattice.)

In such a case, we say that $\alpha : \mathcal{P}(\Sigma_C) \to \Sigma_A$ is the *abstraction map* and $\gamma : \Sigma_A \to \mathcal{P}(\Sigma_C)$ is the *concretization map* [9]. The intuition is that $\alpha(S)$ maps the state set, S, 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 homorphism would do. Dually, $\gamma(a)$ maps an abstract state to the set of concrete states that a represents—$\gamma(a)$ defines the set of potential "refinements" of a.

Figure 7 displays a simple Galois connection between the data values used in the structures in Figures 3 and 5. In the Figure, a Galois connection is first defined between the sets of data values, $\mathcal{P}(Nat)$ and $EvenOdd_\perp^\top$, 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. (Note that the abstract state set, Σ_A, is augmented with the artificial states,



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:

$$
\begin{aligned}
&\Sigma_C = \mathit{ProgramPoint} \times Nat \qquad \Sigma_A = (\mathit{ProgramPoint} \times \mathit{EvenOdd}_\bot^\top)_\bot^\top \\
&\alpha : \mathcal{P}(\Sigma_C) \to \Sigma_A \\
&\quad \alpha(T) = \begin{cases} \bot, & \text{if } T = \{\} \\ (p_k, \alpha_N(T \downarrow 2)), & \text{if } T \downarrow 1 = \{p_k\} \\ \top, & \text{if } p_i, p_j \in T \downarrow 1, i \neq j \end{cases} \\
&\quad \text{where } T \downarrow 1 = \{p \mid (p,n) \in T\}, \quad T \downarrow 2 = \{n \mid (p,n) \in T\} \\
&\gamma : \Sigma_A \to \mathcal{P}(\Sigma_C) \\
&\quad \gamma(p,a) = \{(p,n) \mid n \in \gamma_N(a)\} \\
&\quad \gamma(\bot) = \{\} \\
&\quad \gamma(\top) = \Sigma_C
\end{aligned}
$$

Fig. 7. Example Galois connection

$\bot$ and $\top$, to make a complete lattice. As a technicality, we must assume $\top \longrightarrow \top \in \to_A$.)

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., $\Sigma_A = \mathcal{P}(\mathit{ProgramPoint}) \times \mathit{EvenOdd}_\perp^\top$, or better still, $\Sigma_A = \mathcal{P}(\mathit{ProgramPoint} \times \mathit{EvenOdd}_\perp^\top)$. Such state sets would be useful for defining abstract structures that model nondeterministic computations.

In the usual case, we expect the concretization map to be property preserving and the abstraction map to be property reflecting; these results should be proved directly or should be forced to hold by defining, for all $a \in \Sigma_A$,

$$\mathcal{I}_A(a) = \cap\{\mathcal{I}_C(c) \mid c \in \gamma(a)\}$$

3.3.1 Constructing a simulation with a Galois connection

Our goal has been to construct simulations between abstract structures, $\mathcal{A}$, and concrete structures, $\mathcal{C}$. In the case of a Galois connection, $(\alpha\colon \mathcal{P}(\Sigma_C) \to \Sigma_A, \gamma\colon \Sigma_A \to \mathcal{P}(\Sigma_C))$, we extract the relation, $\mathcal{R}_\gamma \subseteq \Sigma_C \times \Sigma_A$, as follows:

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

The intuition behind $c\, \mathcal{R}_\gamma\, a$ is that a is an approximation of c or that c is a possible refinement of a. $\mathcal{R}_\gamma$ is the obvious candidate to prove a simulation.

But one motivation for defining a Galois connection is studying the transitions that might be taken from a *set* of concrete states! So, we generalize the notion of "transition" as follows:

Definition 12 *For $S, S' \subseteq \Sigma_C$, we write $S \overset{\bullet}{\longrightarrow} S'$ to assert that for every $c' \in S'$, there exists some $c \in S$ such that $c \longrightarrow c'$.*

This definition of transition is used in the obvious way to define a structure, $\mathcal{P}C$, into which $\mathcal{C}$ embeds:

$$\mathcal{P}C = \langle \mathcal{P}(\Sigma_C), \overset{\bullet}{\to}, \mathcal{I}_{PC} \rangle$$

where $\mathcal{I}_{PC}(S) = \cap\{\mathcal{I}_C(c) \mid c \in S\}$.

Given a Galois connection, $(\alpha : \mathcal{P}(\Sigma_C) \to \Sigma_A, \gamma : \Sigma_A \to \mathcal{P}(\Sigma_C))$, 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,

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

and we are left with the task of proving that $\mathcal{R}_{(\alpha,\gamma)}$ is a simulation relation. To do so, 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:

Definition 13 *Given a 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:*

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

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 shows us how the relation on elements of Σ_C naturally lifts into a relation on *subsets* of Σ_C:

Theorem 14 *If $\sqsubseteq_A$ reflects $\rightarrow_A$, then $\mathcal{C} \lhd_{\mathcal{R}_\gamma} \mathcal{A}$ implies that $\mathcal{P}C \lhd_{\mathcal{R}_{(\alpha,\gamma)}} closure(\mathcal{A})$.*

If we have the luxury of defining Kripke structure, $\mathcal{A}$, from scratch, we can force $\mathcal{R}_\gamma$ to be a simulation by defining $\rightarrow_A$ in the obvious, sufficient way:

$$a \longrightarrow a' \in \rightarrow_A \ \text{ iff there exists } c \in \gamma(a), c' \in \gamma(a') \text{ such that } c \longrightarrow c'$$

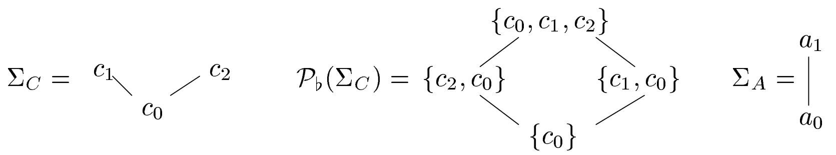
Of course, one is not always allowed to define $\mathcal{A}$ from scratch, so we consider the situation further.

3.3.2 Lifting a homomorphism into a Galois connection

An elegant alternative to proving a simulation directly with a Galois connection is to use a homomorphism as an intermediary: Given structures $\mathcal{C}$ and $\mathcal{A}$, prove that $h : \Sigma_C \rightarrow \Sigma_A$ is a homomorphism in the fashion described earlier, 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 [38]:

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

This lifting technique is useful, because 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 in this way and is proved to be a homomorphism, then it is lifted "automatically" into a Galois connection.



$$\alpha : \mathcal{P}_\flat(\Sigma_C) \to \Sigma_A \qquad\qquad \gamma : \Sigma_A \to \mathcal{P}_\flat(\Sigma_C)$$

$$\alpha(S) = \begin{cases} a_0, & \text{if } S = \{c_0\} \\ a_1, & \text{otherwise} \end{cases} \qquad \begin{array}{l} \gamma(a_0) = \{c_0\} \\ \gamma(a_1) = \{c_0, c_1, c_2\} \end{array}$$

Fig. 8. Lower powerdomain and Galois connection

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

Say that we have $\mathcal{C} \lhd_{\mathcal{R}_h} \mathcal{A}$, and we use h to define (α_h, γ_h). Does this induce a simulation? With the usual assumption that $\sqsubseteq_A$ reflect $\to_A$ and the previous theorem, the answer is "yes":

Corollary 15 *For Kripke structures $\mathcal{C} = \langle \Sigma_C, \to_C, \mathcal{I}_C \rangle$ and $\mathcal{A} = \langle \Sigma_A, \to_A, \mathcal{I}_A \rangle$, let $h : \Sigma_C \to \Sigma_A$ be a homomorphism—hence $\mathcal{C} \lhd_{\mathcal{R}_h} \mathcal{A}$.*

Say that the Galois connection, (α_h, γ_h), is defined from h, as above. If $\sqsubseteq_A$ reflects $\to_A$, then $\mathcal{PC} \lhd_{\mathcal{R}_{(\alpha_h,\gamma_h)}} closure(\mathcal{A})$ holds, where

$$S \; \mathcal{R}_{(\alpha_h,\gamma_h)} \; a \textit{ iff } \alpha_h(S) \sqsubseteq_A a \textit{ iff } S \subseteq \gamma_h(a).$$

3.3.3 Galois connections on lower powerdomains

Stepwise abstractions or refinements 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, $\mathcal{P}_\flat$ [18]: For a complete partially ordered set, $(P, \sqsubseteq_P)$,

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

(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 8.

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. We will not develop further the theory of lower powerdomains but merely note here that it can be applied to the developments in this paper.

3.4 Properties of Binary Relations

We must define a binary relation between two structures so to prove that one structure simulates the other. The previous subsections showed that there are natural binary relations that one can extract from homomorphisms and Galois connections. In the case of a homomorphism, $h : \Sigma_C \rightarrow \Sigma_A$, 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, $(\alpha\colon \mathcal{P}(\Sigma_C) \rightarrow \Sigma_A, \gamma\colon \Sigma_A \rightarrow \mathcal{P}(\Sigma_C))$, 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.

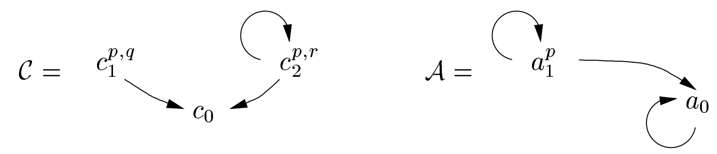
It is worthwhile to "disassemble" these and other binary relations and examine the properties that make the relations well behaved. Initial efforts in this direction were taken by Mycroft and Jones [37], Schmidt [42], and Loiseaux, et al. [31]. (P. Cousot has remarked that the groundwork was laid by Shmuelli, but the appropriate references have not been located as of this date.)

In the developments that follow, we assume the usual Kripke structures, $\mathcal{C} = \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle$ and $\mathcal{A} = \langle \Sigma_A, \rightarrow_A, \mathcal{I}_A \rangle$.

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

Definition 16 *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 $p' \, \mathcal{R} \, q$*



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

Fig. 9. Binary simulation relation

- 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* $p \in P$, $p \,\mathcal{R}\, (\sqcap\{q \mid p \,\mathcal{R}\, q\})$
- 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* $\sqcup_{i \geq 0} p_i \,\mathcal{R}\, \sqcup_{i \geq 0} q_i$.

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

For the concrete and abstract state sets in Figure 8, Figure 9 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 17 *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 18 *Let* $(\alpha\colon \mathcal{P}(\Sigma_C) \to \Sigma_A, \gamma\colon \Sigma_A \to \mathcal{P}(\Sigma_C))$ *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 for abstractions and refinements.

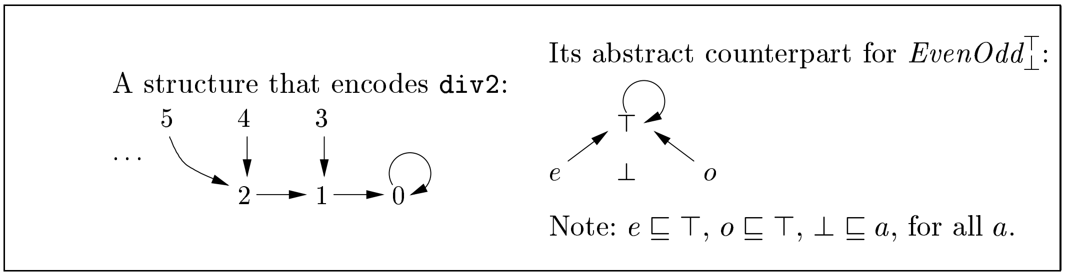


Fig. 10. Why U-closure is helpful for proving a simulation

First, we must emphasize that *the properties in Definition 16 do not ensure that a relation is a simulation.* Here is a trivial counterexample:

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

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 16-properties of a relation, $\mathcal{R}$, we must perform the usual proof that

for all $c \in \Sigma_C, a \in \Sigma_A$,
  $c \; \mathcal{R} \; a$ and $c \longrightarrow c'$ imply
    there exists $a' \in \Sigma_A$ such that $a \longrightarrow a'$ and $c' \; \mathcal{R} \; a'$.

So, we must ask: How do the properties in Definition 16 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 likely require that $\mathcal{R}$ be U-closed (and that $\sqsubseteq_A$ reflect $\rightarrow_A$), as suggested by the example in Figure 10. 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 19 *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.

Refine a_1 into c_1 and c_2, where $c_1 \sqsubseteq_{\mathcal{C}} c_2$:

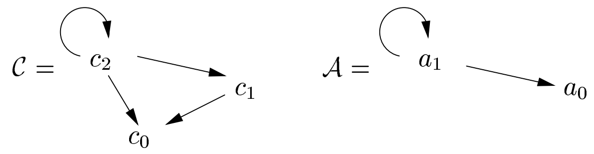


Fig. 11. Why L-closure is helpful for proving a simulation

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 11 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 20 *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.

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 21 *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 22 *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 23 *(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.

Although it is possible to prove a simulation between two structures where the simulation relation has none of the properties studied in this section, useful simulations possess some or all the properties presented here.

4 Properties of Structures

By definition, a homomorphism or simulation respects the transitions of the two structures it relates—$\mathcal{C} \lhd \mathcal{A}$ implies that every transition taken in the concrete structure, $\mathcal{C}$, is mimicked or preserved in the abstract structure, $\mathcal{A}$. And as noted in the previous section, a homomorphism or simulation should also reflect or preserve the atomic properties attached to the states of the respective structures. The usual relationship is *reflection*:

$$c\ \mathcal{R}\ a \text{ implies } \mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)$$

for $c \in \Sigma_C$ and $a \in \Sigma_A$, but we might alternatively or additionally desire *preservation*:

$$c\ \mathcal{R}\ a \text{ implies } \mathcal{I}_C(c) \subseteq \mathcal{I}_A(a)$$

But when we analyze an abstract or concrete structure, we wish to validate more than just atomic properties of states—we want to validate propositions about the sequences of transitions embedded within the structures. Examples of such propositions are

- "starting from state a_0, there exists a sequence of two transitions to a state where property p holds true"
- "starting from state a_0, property p holds true now and at every state reachable from a_0."

- "starting from state a_0, there is a finite sequence of transitions that leads to a state where property p holds true."

Such propositions are elegantly expressed within *temporal logic*. Momentarily, we survey some temporal logics.

If a simulation reflects (or preserves) atomic properties, then we expect that reflection (preservation) "lifts" to the temporal logic in which the atomic properties are embedded: Let $\mathcal{L}_{Atom}$ be a temporal logic whose syntax includes the atomic properties in *Atom*. In the case that $\mathcal{R}$ reflects properties, we expect a similar reflection result for $c \in \Sigma_C$, $a \in \Sigma_A$, and $\phi \in \mathcal{L}_{Atom}$:

$$c\,\mathcal{R}\,a \text{ and } a \models \phi \text{ imply } c \models \phi$$

where $c \models \phi$ denotes that ϕ holds true for (the transition sequences that begin at) c. (This notion is formalized in the next section.) This is called *weak preservation* [12,13,31]).

And, when $\mathcal{R}$ preserves properties, we demand the dual:

$$c\,\mathcal{R}\,a \text{ and } c \models \phi \text{ imply } a \models \phi$$

or, more tellingly expressed in the contrapositive,

$$c\,\mathcal{R}\,a \text{ and } a \not\models \phi \text{ imply } c \not\models \phi$$

When a temporal logic possesses both weak preservation and the above preservation property, this is called *strong preservation* [12,13,31]).

The remainder of this paper is devoted to understanding the forms of temporal logic that reflect and preserve propositions in the presence of simulations.

5 Temporal Logics

As noted by Emerson in his excellent survey [16], temporal logic is a variant of modal logic for expressing the changing truth of propositions with respect to time. A temporal logic might be structured as *linear time* or *branching time*. In the former case, the logic expresses properties of an execution path, that is, of a single sequence of transitions. In the latter case, the logic expresses properties of the transitions ("branchings") that might occur from a state in a Kripke structure.

$$\pi \in \Sigma_C^\infty \qquad \psi \in \mathcal{L}_{Atom} \qquad q \in Atom$$

$$\psi \ ::= \ q \ | \ \neg\psi \ | \ \psi_1 \vee \psi_2 \ | \ \mathsf{X}\psi \ | \ \mathsf{F}\psi \ | \ \mathsf{G}\psi \ | \ \psi_1 \mathsf{U} \psi_2$$

For $\pi = s_0 s_1 \cdots s_i \cdots$ and $k \geq 0$,

define $\pi_k = s_k$ and $tl_k(\pi) = s_k s_{k+1} \cdots$

$\pi \models q$ iff $q \in \mathcal{I}_C(\pi_0)$

$\pi \models \neg\psi$ iff $\pi \not\models \psi$

$\pi \models \psi_1 \vee \psi_2$ iff $\pi \models \psi_1$ or $\pi \models \psi_2$

$\pi \models \mathsf{X}\psi$ iff $tl_1(\pi) \models \psi$

$\pi \models \mathsf{F}\psi$ iff there exists $k \geq 0$ such that $tl_k(\pi) \models \psi$

$\pi \models \mathsf{G}\psi$ iff for all $k \geq 0, tl_k(\pi) \models \psi$

$\pi \models \psi_1 \mathsf{U} \psi_2$ iff there exists $k \geq 0$ such that for all $0 \leq j < k$,
$tl_j(\pi) \models \psi_1$, and $tl_k(\pi) \models \psi_2$

Fig. 12. LTL syntax and semantics

The sections that follow present one linear-time logic and two branching-time logics.

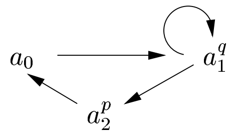
5.1 LTL

Given a Kripke structure, $\mathcal{C}$, we call a sequence of states, $s_0 s_1 \cdots s_i \cdots$, for $s_j \in \Sigma_C$, $j \geq 0$, an *execution path*. Unless stated otherwise, we assume that an execution path has infinite length.

The formulas of LTL [16,40] express properties of execution paths; LTL's syntax and semantics appear in Figure 12. LTL is a propositional logic of atomic propositions extended by the four modalities, X ("neXt"), F ("in the Future"), G ("Globally"), and U ("Until").

We can best understand the modalities by means of examples. Say that we have the Kripke structure pictured in Figure 13, and we wish to study properties of execution paths starting at state a_0. The three propositions stated in English in the earlier section might be phrased in LTL as follows:

- $\mathsf{XX}p$: "along the execution path, two transitions lead to a state where p holds true." This assertion can be checked for any path we choose; for example, $a_0 a_1 a_2 Z \models \mathsf{XX}p$ holds true, for any suffix $Z \in \mathit{PathsFrom}(a_0)$, but



Let $P = a_0 a_1^* a_2$

$PathsFrom(a_0) = P^\infty \cup P^* a_0 a_1^\infty$

Fig. 13. Execution paths within a Kripke structure

$a_0 a_1 a_1 a_2 Z \not\models \mathsf{XX}p$. Thus, the property holds for some, but not all the execution paths starting at a_0.
- $\mathsf{G}p$: "along the execution path, property p holds globally true." This property fails to hold for all the execution paths starting at a_0; indeed, only $a_1^\infty \models \mathsf{G}p$ holds true of all the executions contained in Figure 13.
- Fp: "along the execution path, there is a future where p holds." We have that $a_0 a_1^* a_2 Z \models Fp$, for all $Z \in PathsFrom(a_0)$, but not all paths starting from a_0 satisfy the property, e.g., $a_0 a_1^\infty$ does not.

As for the modality, U, we might check whether atomic proposition q holds until p does; we write this as $q\mathsf{U}p$. This result clearly holds for every execution starting at state a_1, and it trivially holds for every execution starting at a_2, but it fails for every execution starting at a_0 (since q does not hold true at a_0 itself).

The modalities can be nested, e.g., $\mathsf{FG}q$ states that at some future in an execution path, q holds globally true. This property holds for the path, $a_2 a_0 a_1^\infty$.

In passing, we note that the F and G modalities are redundant: $\mathsf{F}\psi$ is equivalent to $true\mathsf{U}\psi$, and $\mathsf{G}\psi$ is equivalent to $\neg\mathsf{F}(\neg\psi)$. Also, if transitions in a Kripke structure possess labels, ℓ, then the notion of execution path can be extended to include the labels, e.g., $s_0 \ell_0 s_1 \ell_1 \cdots$, and the modalities can refer to the labels as needed, e.g., $\mathsf{X}_{\ell_1}\mathsf{F}_{\ell_2}\psi$ states that the initial transition is an ℓ_1-labelled transition and that there is a sequence of ℓ_2-labelled transitions that lead to a point where ψ holds.

In practice, properties coded in LTL are checked for all the paths that begin at a particular state, s, and one writes $s \models \forall\psi$ if every path starting at s makes ψ hold true.

5.2 CTL

Correctness properties of a Kripke structure are often stated in terms of the state from which execution begins, and CTL ("computation tree logic") [3,16] expresses properties of possible executions from a specific state.

$$s \in \Sigma_C \qquad \pi \in \Sigma_C^{\infty} \qquad \phi \in \mathcal{L}_{Atom} \qquad q \in Atom$$

$$\begin{array}{lcl} \phi & ::= & q \;\mid\; \neg\phi \;\mid\; \phi_1 \vee \phi_2 \;\mid\; \mathsf{E}\psi \;\mid\; \mathsf{A}\psi \\ \psi & ::= & \mathsf{X}\phi \;\mid\; \mathsf{G}\phi \;\mid\; \phi_1 \mathsf{U} \phi_2 \end{array}$$

$s \models q$ iff $q \in \mathcal{I}_C(s)$

$s \models \neg\phi$ iff $s \not\models \phi$

$s \models \phi_1 \vee \phi_2$ iff $s \models \phi_1$ or $s \models \phi_2$

$s \models \mathsf{E}\psi$ iff there exists a path, π, within $\mathcal{C}$ such that $\pi_0 = s$ and $\pi \models \psi$

$s \models \mathsf{A}\psi$ iff for all paths, π, within $\mathcal{C}$ such that $\pi_0 = s, \pi \models \psi$

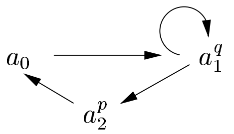
$\pi \models \mathsf{X}\phi$ iff $\pi_1 \models \phi$

$\pi \models \mathsf{G}\phi$ iff for all $k \geq 0, \pi_k \models \phi$

$\pi \models \phi_1 \mathsf{U} \phi_2$ iff there exists $k \geq 0$ such that for all $0 \leq j < k$,

$\pi_j \models \phi_1$, and $\pi_k \models \phi_2$

Fig. 14. CTL syntax and semantics



$a_i \models \mathsf{EX}(\mathsf{EX}p), i \in 0..1$

$a_2 \not\models \mathsf{EX}(\mathsf{EX}p)$

$a_j \not\models \mathsf{AG}p, j \in 0..2$

$a_j \models \mathsf{E}(true\mathsf{U}p), j \in 0..2$

Fig. 15. Examples of CTL properties

The syntax and semantics of CTL appears in Figure 14. The Figure shows that CTL has a "two level" syntactic structure, where LTL-like formulas can be quantified by A ("for All paths") and E ("there Exists a path"). The syntax, in effect, generates just these six modalities: AX, AG, AU, EX, EG, and EU.

Although CTL appears to be "quantified LTL," it is *not*—the semantics of the path modalities, X, G, and U are different from what was stated in Figure 12: In Figure 14, each path modality examines the states embedded in a path and *not* the subpaths of the path. This makes the semantics of the CTL modalities into a variation of reachability analysis, as shown in the examples that follow.

Figure 15 redisplays the example Kripke structure and its properties coded in CTL. We note that

- $a_0 \models \mathsf{EX}(\mathsf{EX}p)$, because the semantics of $\mathsf{EX}(\mathsf{EX}p)$ states that a_0 has a path whose next state, a_1, itself has a path whose next state (namely, a_2) makes

p hold—note that *two* paths are used to validate the claim. In this sense, CTL is fundamentally different from LTL, which validates a proposition on exactly one path at a time. This is the reason why CTL is a branching-time logic and why the CTL formula can be read as "starting from state a_0, there exists a sequence of two transitions to a state where p holds true."

Contrast the CTL formula with this one: $\mathsf{AX}(\mathsf{AX}p)$, which requires that all possible two-step transitions lead to a state where p holds. This fails for a_0.

- $a_0 \not\models \mathsf{AG}p$. The semantics of A and G imply that, along all paths starting at a_0, all reachable states must make p hold. This matches the English narrative: "starting from state a_0, property p holds true now and at every state reachable from a_0." Although the semantics is expressed in terms of "all paths," a state-reachability analysis readily checks formulas of the form $\mathsf{AG}\phi$—verify that every state reachable from a_0 makes ψ hold true.
- $a_0 \models \mathsf{E}(true\mathsf{U}p)$ holds, because "starting from state a_0, there is a finite sequence of transitions that leads to a state where property p holds true." Similar to the previous two examples, one validates the formula by locating a state, a', such that a' is reachable from a_0 and $a' \models p$.

  It is common to abbreviate the form, $\mathsf{E}(true\mathsf{U}\psi)$, as $\mathsf{EF}\psi$.

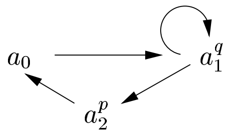
The previous examples make clear that a branching-time logic emphasizes states and the transitions between them, rather than complete paths. For this reason, we next extend CTL to a logic based totally on states and the transitions between them.

5.3 Modal mu-calculus

We use *modal mu-calculus* [26,27,44] for the remainder of the paper because it is simple to define, highly expressive, and most other branching-time temporal logics (e.g., CTL) easily translate into it. Modal mu-calculus consists of *Atom* (the atomic properties of states) extended by propositional logic, and the modalities $\Diamond$ (diamond) and $\Box$ (box), and recursive definitions.

$\Diamond\phi$ can be read as the assertion, "from the current state, there exists a transition to a next state where ϕ holds." Similarly, $\Box\phi$ can be read as, "from the current state, all transitions lead to next states where ϕ holds." Therefore, the example propositions stated at the beginning of this section can be defined in modal-mu calculus as follows:

- $a_0 \models \Diamond\Diamond p$: "starting from state a_0, there exists a sequence of two transitions to a state where p holds true."
- $a_0 \models R$, where $R = p \wedge \Box R$: "starting from state a_0, property p holds true now and at every state reachable from a_0." The recursion is a "maximal"



$a_i \models \Diamond\Diamond p, i \in 0..1$

$a_2 \not\models \Diamond\Diamond p$

$a_j \not\models \nu R.p \wedge \Box R, j \in 0..2$

$a_j \models \mu R.p \vee \Diamond S, j \in 0..2$

Fig. 16. Examples of modal mu-calculus properties

$s \in \Sigma_C \qquad \phi \in \mathcal{L}_{Atom} \qquad q \in Atom \qquad Z \in \mathit{Identifier}$

$$\phi \; ::= \; q \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Box\phi \mid \Diamond\phi \mid \mu Z.\phi \mid \nu Z.\phi \mid Z$$

$s \models q$ iff $q \in \mathcal{I}_C(s)$

$s \models \neg\phi$ iff $s \not\models \phi$

$s \models \phi_1 \wedge \phi_2$ iff $s \models \phi_1$ and $s \models \phi_2$

$s \models \phi_1 \vee \phi_2$ iff $s \models \phi_1$ or $s \models \phi_2$

$s \models \Box\phi$ iff for all s' such that $s \longrightarrow s', s' \models \phi$

$s \models \Diamond\phi$ iff there exists s' such that $s \longrightarrow s'$ and $s' \models \phi$

$s \models \mu Z.\phi$ iff there exists $i \geq 0$ such that $s \models \phi_i$, where $\begin{cases} \phi_0 = \mathit{false} \\ \phi_{i+1} = [\phi_i/Z]\phi \end{cases}$

$s \models \nu Z.\phi$ iff for all $i \geq 0, s \models \phi_i$, where $\begin{cases} \phi_0 = \mathit{true} \\ \phi_{i+1} = [\phi_i/Z]\phi \end{cases}$

Fig. 17. Modal mu-calculus for finite-state Kripke structures

one (it holds true even for infinite paths of transitions), and it is written more tersely as $\nu R.p \wedge \Box R$.

- $a_0 \models S$, where $S = p \vee \Diamond S$: "starting from state a_0, there is a finite sequence of transitions that leads to a state where property p holds true." The recursion is a "minimal" one (it holds true for a path from a_0 if and only if it holds for a finite prefix of the path), and it is written more tersely as $\mu S.p \vee \Diamond S$.

Figure 16 presents the example structure and notes which of the above properties hold true for states within the structure.

The precise syntax and semantics of modal mu-calculus appears in Figure 17. The semantics of negation makes it easy to prove that $\neg\Diamond\phi$ is equivalent to $\Box\neg\phi$ and $\neg\Box\phi$ is equivalent to $\Diamond\neg\phi$.

$$s \in \Sigma_C \qquad \phi \in \mathcal{L}_{Atom} \qquad q \in Atom \qquad Z \in Identifier$$
$$\rho \in Env = Identifier \to \mathcal{P}(\Sigma_C)$$

$s \models \phi$ iff $s \in [\![\phi]\!]\rho$

$[\![\cdot]\!] \in \mathcal{L}_{Atom} \to Env \to \mathcal{P}(\Sigma_C)$

$$[\![q]\!]\rho = \{s \mid q \in \mathcal{I}_C(s)\}$$
$$[\![\neg\phi]\!]\rho = \Sigma_C - [\![\phi]\!]\rho$$
$$[\![\phi_1 \wedge \phi_2]\!]\rho = [\![\phi_1]\!]\rho \cap [\![\phi_2]\!]\rho$$
$$[\![\phi_1 \vee \phi_2]\!]\rho = [\![\phi_1]\!]\rho \cup [\![\phi_2]\!]\rho$$
$$[\![\Box\phi]\!]\rho = \{s \mid \text{ for all } s' \text{ such that } s \longrightarrow s', s' \in [\![\phi]\!]\rho\}$$
$$[\![\Diamond\phi]\!]\rho = \{s' \mid \text{ there exists } s' \text{ such that } s \longrightarrow s' \text{ and } s' \in]\!]\phi]\!]\rho\}$$
$$[\![\mu Z.\phi]\!]\rho = \bigcap\{S \subseteq \Sigma_C \mid F(S) \subseteq S\}$$
$$[\![\nu Z.\phi]\!]\rho = \bigcup\{S \subseteq \Sigma_C \mid S \subseteq F(S)\}, \quad \text{where } F(S) = [\![\phi]\!](\rho + [Z \mapsto S])$$

Fig. 18. Modal mu-calculus semantics for infinite-state structures

The semantics of the recursion operators, μZ and νZ, depend on the fact that Σ_C, the state set of structure $\mathcal{C}$, has finite cardinality. In such a case, $\mu Z.\phi$ unfolds into an "infinite disjunction," $\phi_0 \vee \phi_1 \vee \cdots \vee \phi_j \vee \cdots$, where $\phi_0 = false$ and $\phi_{i+1} = [\phi_i/Z]\phi$. (Note: $[\phi_i/Z]\phi$ represents the syntactic substitution of ϕ_i for all free occurrences of Z in ϕ.) Similarly, $\nu Z.\psi$ unfolds into an "infinite conjunction," $\psi_0 \wedge \psi_1 \wedge \cdots \wedge \psi_j \wedge \cdots$, where $\psi_0 = true$ and $\psi_{i+1} = [\psi_i/Z]\psi$. (Note: Assume that no state possesses the atomic property, *false* and every state possesses property, *true*.)

If Σ_C has infinite cardinality, the simple semantics of the recursion operators must be replaced by the machinery of Tarski's fixed-point theorems; see Figure 18 for the resulting semantics. Note: $\rho + [Z \mapsto S]$ defines the function that maps Z to S and all other arguments, I, to $\rho(I)$.

In both Figures, it is assumed that all occurrences of identifiers, Z, occur bound and only in *positive* positions in propositions, ϕ, that is, there are exactly an even quantity of negations, $\neg$, that enclose Z. (For example, Z is enclosed by two negations in $\mu Z.\neg(p \wedge \neg\Box Z)$.) This restriction ensures that the semantics of μZ and νZ are well defined.

If a Kripke structure with labelled transitions is used, as in Figure 1, one can define the modalities so that the labels are taken into account:

$$s \models [\ell]\phi \text{ iff } \text{ for all } s' \text{ such that } s \xrightarrow{\ell} s', s' \models \phi$$
$$s \models \langle\ell\rangle\phi \text{ iff } \text{ there exists } s' \text{ such that } s \xrightarrow{\ell} s' \text{ and } s' \models \phi$$

Here, $[\ell]$ represents $\Box$ restricted to just ℓ-labelled transitions and $\langle\ell\rangle$ represents $\Diamond$ restricted to just ℓ-labelled transitions [21]. The results described in the sections that follow apply to the modal mu-calculus with the "labelled modalities" as well.

Because of the simple semantics of the negation operator, it is easy to validate that the modal mu-calculus is *consistent*: It is impossible to prove both $s \models \phi$ and $s \models \neg\phi$, for all states s and propositions ϕ. Also, the calculus is *complete* because one can always prove either $s \models \phi$ or $s \models \neg\phi$.

Unlike the semantics of LTL and CTL, the semantics of modal mu-calculus states nothing about paths, and there is no restriction that all execution paths be infinite.

5.4 Translating CTL into modal mu-calculus

We can readily translate the six modalities of CTL into modal mu-calculus:

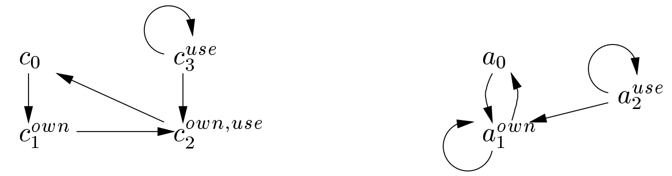
$$\begin{aligned}
\mathsf{EX}\phi &= \Diamond\phi \\
\mathsf{AX}\phi &= \Box\phi \wedge \Diamond true \\
\mathsf{AG}\phi &= \nu Z.\phi \wedge \mathsf{AX}Z \\
\mathsf{EG}\phi &= \nu Z.\phi \wedge \mathsf{EX}Z \\
\mathsf{A}(\phi_1 \mathsf{U} \phi_2) &= \mu Z.\phi_2 \vee (\phi_1 \wedge \mathsf{AX}Z) \\
\mathsf{E}(\phi_1 \mathsf{U} \phi_2) &= \mu Z.\phi_2 \vee (\phi_1 \wedge \mathsf{EX}Z)
\end{aligned}$$

In the translation of $\mathsf{AX}\phi$, we must include the phrase, $\Diamond true$, to ensure that a next move does indeed exist. If we are certain that the Kripke structure we analyze contains no deadlocked states, that is, all execution paths must be infinite, then we have

$$\mathsf{AX}\phi = \Box\phi$$

Because of the above translations, the remainder of the paper examines propositions one writes in the modal mu-calculus.

Define $\mathcal{R} = \{(c_0, a_0), (c_1, a_1), (c_2, a_1), (c_3, a_2)\}$:



		$a_i \models (1),\ \text{for } i \in 0..1$
(1)	$\nu Z.(use \supset own) \wedge \Box Z$	and also $c_i \models (1),\ \text{for } i \in 0..2$
(2)	$\mu Z.\neg own \vee (\Box Z \wedge \Diamond true)$	$a_1 \not\models (2)$
		but $c_i \models (2),\ \text{for } i \in 1..2$

Fig. 19. Analysis of abstract structure

6 What propositions do simulations respect?

For Kripke structures $\mathcal{C}$ and $\mathcal{A}$ and property reflecting simulation, $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$, say that $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$ holds. In this classic situation, we plan to analyze $\mathcal{A}$ to determine which propositions, ϕ, of the modal mu-calculus hold true for states in $\mathcal{A}$. We want reflection, so that the knowledge we gain about $\mathcal{A}$ can apply to $\mathcal{C}$ as well:

$$c\, \mathcal{R}\, a \text{ and } a \models \phi \text{ imply } c \models \phi$$

Are we guaranteed this result from the existence of the property-reflecting simulation, $\mathcal{R}$? We study some examples to learn more.

Figure 19 gives a simple example where a resource-using process is abstracted by a property-reflecting simulation and the abstract structure is analyzed for two correctness properties.

Using CTL notation, we can restate Proposition (1) in the Figure as $\mathsf{AG}(use \supset own)$ and Proposition (2) as $\mathsf{A}(true \mathsf{U} \neg own)$ (or, $\mathsf{AF}\neg own$), but we use the modal mu-calculus formulas for reasons of exposition.

In the Figure, Proposition (1) states that, for all reachable states, the resource must be owned before it is used. (Of course, $use \supset own$ abbreviates $\neg use \vee own$.) This proposition is proved true for states a_0 and a_1 of the abstract structure, and indeed, the proposition holds true for the corresponding states of the concrete structure. These results are consistent with the reflection that we desire.

Define $\mathcal{R} = \{(start, start_a), (lose, lose_a), (win, win_a)\}$:

$start_a \models \Diamond won$ but $start \not\models \Diamond won$

Fig. 20. A refinement (left) of a specification (right)

Proposition (2) asserts that the resourse will not be owned forever. (Note that $\Diamond true$ indicates that there must be a next state.) Proposition (2) holds for states c_1 and c_2 (and c_0) of the concrete model, but the loss of precision in the abstract model, due to state merging, makes it impossible to verify (2) on state a_1, which represents c_1 and c_2. Failure to validate a correctness property on an abstract model might not be a disaster—perhaps a more detailed abstract model can be formulated on which the proposition can be validated, e.g., [4,23,41]. In any case, the result of this experiment is consistent with reflection.

Next, consider Figure 20, which shows, on the right, a specification of a slot machine ("one-armed bandit") and on the left, the refinement of the specification. The refinement seems somewhat "unfair" to the machine's user, but nonetheless, there is a property-reflecting simulation between the refinement and the specification, as indicated in the Figure.

The "unfairness" of the refinement is reflected in the fact that the proposition, $\Diamond won$, holds true of the start state of the specification but *fails* to hold of the start state of the refinement—reflection is violated!

An even simpler example of violation of reflection is seen in this trivial example of a property-reflecting simulation:

Let $\mathcal{C} = \langle \{c_0\}, \{\}, \mathcal{I}_C \rangle$, where $\mathcal{I}_C(c_0) = \{p\}$.

Let $\mathcal{A} = \langle \{a_0\}, \{\}, \mathcal{I}_A \rangle$, where $\mathcal{I}_A(a_0) = \{\}$.

Define $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$ as $c_0 \,\mathcal{R}\, a_0$. Then, $a_0 \models \neg p$ but $c_0 \not\models \neg p$.

There are two possible causes of violation of reflection:

(1) The diamond modality, $\Diamond$, and the negation operator, $\neg$, are unreliable for analysis of abstract structures.
(2) The notion of property-reflecting simulation is too weak to enforce reflection.

Arguments can be made for both claims, and we investigate both in the sequel.

6.1 Propositions that necessarily hold

When a simulation reflects atomic properties, we are ensured that $c\,\mathcal{R}\,a$ and $q \in \mathcal{I}_A(a)$ imply $q \in \mathcal{I}_C(c)$. Say that we want this property to ensure reflection for formulas in temporal logic:

$$c\,\mathcal{R}\,a \text{ and } a \models \phi \text{ imply } c \models \phi$$

In this case, we say that the validation of ϕ for a makes ϕ *necessarily hold true for c*. In the previous section, we saw propositions in the full modal mu-calculus that hold true for an abstract structure do not necessarily hold true for the corresponding concrete structure. Fortunately, there is an easily defined sublogic of the modal mu-calculus whose formulas do necessarily hold true of concrete structures when they hold true for abstract ones:

Definition 24 *Given $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$,*

(1) define $N(\mathcal{R}) = \{q \in Atom \mid \text{ for all } c \in \Sigma_C, a \in \Sigma_A, c\,\mathcal{R}\,a \text{ and } q \in \mathcal{I}_A(a) \text{ imply } q \in \mathcal{I}_C(c)\}$.
(2) define $Necessarily_{N(\mathcal{R})}$ as the subset of modal mu-calculus generated by this syntax rule:

$$\phi \;::=\; q \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Box\phi \mid \mu Z.\phi \mid \nu Z.\phi \mid Z$$

for $q \in N(\mathcal{R})$ and $Z \in Identifier$.
(3) for $\phi \in Necessarily_{N(\mathcal{R})}$, define $a \models_{nec} \phi$ iff $a \models \phi$.

Regarding Clause (i), when $\mathcal{R}$ is property reflecting, $N(\mathcal{R}) = Atom$. As for Clause (ii), the sublogic, $Necessarily_{N(\mathcal{R})}$, is the modal mu-calculus less the diamond and negation operators. Clause (iii) notes that the semantics of this sublogic is just the one from Figure 17. The sublogic is useful for expressing safety and invariance properties, that is, "good behavior" that must hold for all paths in a structure; we study examples momentarily.

Given the newly defined sublogic, we have this theorem, whose proof can be found in Loiseaux, et al. [31], among others:

Theorem 25 *For $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$ and for all $c \in \Sigma_C$, $a \in \Sigma_A$, and $\phi \in Necessarily_{N(\mathcal{R})}$:*

$$c\,\mathcal{R}\,a \textit{ and } a \models_{nec} \phi \textit{ imply } c \models \phi$$

That is, properties stated in $Necessarily_{N(\mathcal{R})}$ that are validated on an abstract structure are necessarily true of the corresponding concrete structure.

Here are two examples of the forms of properties that one can validate as necessarily true:

(1) $\nu Z.p \wedge \Box Z$: an invariant property, p, holds true at every reachable state.
(2) $\mu Z.p \vee \Box Z$: a state having property p or a terminal state must be reached in a finite number of transitions

When we reconsider Propositions (1) and (2) within Figure 19, we find that neither belong to $\mathit{Necessarily}_{N(\mathcal{R})}$, which is unfortunate because both are useful.

We might repair Proposition (1) by using the atomic property set $\mathit{Atom} = \{\mathit{own}, \mathit{notown}, \mathit{use}, \mathit{notuse}\}$ and rewrite the proposition to read

$$\nu Z.(\mathit{notuse} \vee \mathit{own}) \wedge \Box Z$$

Of course, we would require the consistency condition that for all $c \in \Sigma_C$, $\{\mathit{notown}, \mathit{own}\} \not\subseteq \mathcal{I}_C(c)$ and $\{\mathit{notuse}, \mathit{use}\} \not\subseteq \mathcal{I}_C(c)$.

A second possible solution is to ensure that simulation $\mathcal{R}$ both reflects *and* preserves atomic properties; then the syntax of $\mathit{Necessarily}_{N(\mathcal{R})}$ can be extended to

$$\phi \ ::= \ q \ \mid \ \neg q \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \phi_1 \vee \phi_2 \ \mid \ \Box\phi \ \mid \ \mu Z.\phi \ \mid \ \nu Z.\phi \ \mid \ Z$$

for $q \in \mathit{Atom}$. (More precisely, we need only demand that $\mathcal{R}$ reflect unnegated atomic properties, q, and preserve the atomic properties that appear within $\neg q$.) Then, Proposition (1) from Figure 19 can be used as it is.

When we consider Proposition (2) from Figure 19, we note that $\Diamond \mathit{true}$ has no natural representation in the $\mathit{Necessarily}_{N(\mathcal{R})}$ logic. We sidestep this problem by demanding that the concrete structure have no deadlocked states, that is, states from which no transitions are possible. Now, any abstract structure that is a simulation of the concrete structure will have no deadlocked states, hence all its execution paths are infinite and there is no need for the phrase, $\Diamond \mathit{true}$, because it necessarily holds. This simplifies Proposition (2) to

$$\mu Z.\neg \mathit{own} \vee \Box Z$$

Theorem 25 forms the foundation for the commonly used temporal logic ACTL [6], which is CTL less E and $\neg$ and where $\mathsf{AX}\phi$ is understood as $\Box\phi$.

6.2 Propositions that possibly hold

The development in the previous subsection can be profitably dualized: When a simulation preserves atomic properties, we have that $c \mathcal{R} a$ and $q \in \mathcal{I}_C(c)$ imply $q \in \mathcal{I}_A(a)$. Say that we want this property to ensure preservation for formulas in a temporal logic:

$$c \, \mathcal{R} \, a \text{ and } c \models \phi \text{ imply } a \models \phi$$

Since an analysis is normally undertaken on the abstract structure, preservation appears of little use. But consider its definition in the contrapositive:

$$c \, \mathcal{R} \, a \text{ and } a \not\models \phi \text{ imply } c \not\models \phi$$

This makes clear that preservation proves effective for *refuting* properties of a concrete structure. For this reason, we say that the refutation of ϕ for a makes ϕ *not possibly hold true for c*. Similarly, we say that the validation of ϕ for a makes ϕ *possibly hold true for c.*

Now, we follow a parallel development to the previous subsection:

Definition 26 *Given $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$,*

(1) define $P(\mathcal{R}) = \{q \in Atom \mid$ for all $c \in \Sigma_C, a \in \Sigma_A, c \; \mathcal{R} \; a$ and $q \in \mathcal{I}_C(c)$ imply $q \in \mathcal{I}_A(a)\}$.
(2) define $Possibly_{P(\mathcal{R})}$ as the subset of modal mu-calculus generated by this syntax rule:

$$\phi \;::=\; q \;\mid\; \phi_1 \wedge \phi_2 \;\mid\; \phi_1 \vee \phi_2 \;\mid\; \Diamond\phi \;\mid\; \mu Z.\phi \;\mid\; \nu Z.\phi \;\mid\; Z$$

for $q \in P(\mathcal{R})$ and $Z \in Identifier$.
(3) for $\phi \in Possibly_{P(\mathcal{R})}$, define $a \models_{pos} \phi$ iff $a \models \phi$.

For Clause (i), when $\mathcal{R}$ is property preserving, $P(\mathcal{R}) = Atom$; for Clause (ii), $Possibly_{P(\mathcal{R})}$ is the modal mu-calculus less the box and negation operators; Clause (iii) notes that the semantics of the sublogic is the one from Figure 17. The sublogic is useful for expressing instances of "bad behavior" that must be refuted for all paths in a structure.

The proof of the following theorem can be found in Kelb [25] and Levi [30], among others:

Theorem 27 *For $\mathcal{C} \lhd_{\mathcal{R}} \mathcal{A}$ and for all $c \in \Sigma_C$, $a \in \Sigma_A$, and $\phi \in Possibly_{P(\mathcal{R})}$:*

$$c \, \mathcal{R} \, a \text{ and } a \not\models_{pos} \phi \text{ imply } c \not\models \phi$$

That is, properties stated in $Possibly_{P(\mathcal{R})}$ that are refuted on an abstract structure are not possibly true of the corresponding concrete structure.

If a simulation, $\mathcal{R}$, both reflects and preserves atomic properties, we can employ this slightly more useful variant of $Possibly_{P(\mathcal{R})}$:

$$\phi \ ::= \ q \ | \ \neg q \ | \ \phi_1 \wedge \phi_2 \ | \ \phi_1 \vee \phi_2 \ | \ \Diamond\phi \ | \ \mu Z.\phi \ | \ \nu Z.\phi \ | \ Z$$

for $q \in Atom$. (Again, it suffices that $\mathcal{R}$ merely preserve unnegated atomic properties, q, and reflect the atomic properties that appear within $\neg q$.)

Examples of propositions in $Possibly_{P(\mathcal{R})}$ that are good candidates for refutation are

(1) $\nu Z.p \wedge \Diamond Z$: there exists an infinite path, all of whose states possess an erroneous property, p.
(2) $\mu Z.p \vee \Diamond Z$: there exists a finite path to an undesirable property, p.

A novel application of $Possibly_{P(\mathcal{R})}$ was found by Dwyer and Pasareanu [14], who write correctness properties in precondition-postcondition format: $\phi_1 \Rightarrow \phi_2$. The precondition, ϕ_1, is called a *filter* [15] and states basic well-formed properties of the structure (e.g., procedure call-return paths are well formed, or interleavings of steps of synchronized processes are well formed).

Dwyer and Pasareanu then generate an inexpensive, simplistic abstract structure and "filter" it by checking $a_0 \models_{pos} \phi_1$, for its start state, a_0. If ϕ_1 is refuted, then the abstract structure is not possibly good enough, and the derivation of the refutation is used to refine the abstract structure. The structure is checked again, and only when ϕ_1 possibly holds true for a_0, does the second check, $a_0 \models_{nec} \phi_2$, proceed.

6.3 Connecting necessarily true and possibly true via negation

The example that ended the previous subsection suggests that the negation operator can be added to both $Necessarily_{N(\mathcal{R})}$ and $Possibly_{P(\mathcal{R})}$ if one is willing to "jump" from one logic to the other. The following semantics of negation for necessarily and possibly propositions was proposed by Kelb [25]:

$$(1) \qquad a \models_{nec} \neg\phi \text{ iff } a \not\models_{pos} \phi$$

$$(2) \qquad a \models_{pos} \neg\phi \text{ iff } a \not\models_{nec} \phi$$

Clause (1) should be read as "necessarily $\neg\phi$ is not possibly ϕ"; Clause (2) states, "possibly $\neg\phi$ is not necessarily ϕ." To employ Clauses (1) and (2), we must verify that $\mathcal{R}$ reflect and preserve the atomic properties contained in ϕ. Also, proposition ϕ in (1) must be written in the sublogic $Possibly_{P(\mathcal{R})}$; similarly, ϕ in (2) must be written in $Necessarily_{N(\mathcal{R})}$—at this point, we cannot interpret $a \models_{nec} \neg\Box\phi'$, for example.

Theorems 25 and 27 can be readily extended to include the above semantics (subject to the restrictions just mentioned).

7 Mixed-transition systems

Adding the negation operator to both $Necessarily_{N(\mathcal{R})}$ and $Possibly_{P(\mathcal{R})}$ places us tantalizingly close to using full modal mu-calculus for checking propositions that necessarily and possibly hold true. But, to define semantics for $a \models_{nec} \Diamond\phi$ and $a \models_{pos} \Box\phi$, we are forced to return to the basic question of the suitability of simulations for relating concrete to abstract structures.

Recall the annoying situation from Figure 20: a specification of a slot machine contained the transition, $start_a \longrightarrow win_a$, but the specification's refinement did not; therefore $start_a \models \Diamond won$ but $start \not\models \Diamond won$. One's intuition demands that the transition, $start_a \longrightarrow win_a$ should be—indeed, must be—preserved in the refinement, but it was not. In this regard, the concrete structure in Figure 20 is not well matched to the abstract structure. This generated the problem with the assertion, $\Diamond won$.

Throughout this paper, we have tacitly read a transition, $a \longrightarrow a' \in\rightarrow_A$, as an abstract-structure transition step that *may* occur in the corresponding concrete structure. This reading makes perfect sense when a concrete structure is simplified into an abstract structure—say, by merging states—because precision-of-transitions is lost. But the anomalous situation described in the previous paragraph cries out for a form of abstract-structure transition that *must* appear in the corresponding concrete structure. Such a form of transition will be crucial for practical refinement activities.

The existing abstract-structure transitions are termed "may-transitions," e.g., $start_a \stackrel{may}{\longrightarrow} lose_a$ from Figure 20, and the collection of may-transitions is named $\stackrel{may}{\longrightarrow}_A$. A simulation of a concrete structure by an abstract structure relies on the may-transitions:

$$\langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle \lhd_{\mathcal{R}} \langle \Sigma_A, \stackrel{may}{\longrightarrow}_A, \mathcal{I}_A \rangle$$

That is, transitions in $\rightarrow_C$ are mimicked by those in $\stackrel{may}{\longrightarrow}_A$.

Define $\mathcal{R} = \{(start, start_a), (lose, lose_a), (win_0, win_a), (win_1, win_a)\}$:

Fig. 21. Simulation by a mixed-transition system

Now, here is the novelty: We allow an abstract structure to possess "must-transitions," with the understanding that a must-transition in an abstract structure must have a counterpart in the concrete structure. In particular, we might add $start_a \stackrel{must}{\longrightarrow} win_a$ to Figure 20 and force the concrete structure to respect the must-transitions. For this, we require a simulation in the *dual* direction:

$$\langle \Sigma_A, \stackrel{must}{\longrightarrow}_A, \mathcal{I}_A \rangle \lhd_{\mathcal{R}^{-1}} \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle$$

That is, each must-transition in $\stackrel{must}{\longrightarrow}_A$ is mimicked by a transition in $\rightarrow_C$.

The addition of must-transitions and the requirement of dual simulation form the basis for *mixed-transition systems* [12,13]:

Definition 28 *A* Kripke mixed-transition system *is a tuple,* $\mathcal{K} = \langle \Sigma_K, \stackrel{must}{\longrightarrow}_K, \stackrel{may}{\longrightarrow}_K, \mathcal{I}_K \rangle$, *where*

- Σ_K *is a set of states.*
- $\stackrel{must}{\longrightarrow}_K, \stackrel{may}{\longrightarrow}_K \subseteq \Sigma_K \times \Sigma_K$ *are transition relations with finite image.*
- $\mathcal{I}_K : \Sigma_K \rightarrow \mathcal{P}(Atom)$ *associates atomic properties,* $\mathcal{I}_K(s) \subseteq Atom$*, to each* $s \in \Sigma_K$*.*

For a Kripke structure, $\mathcal{C} = \langle \Sigma_C, \rightarrow_C, \mathcal{I}_C \rangle$*, and a Kripke mixed-transition system* $\mathcal{A} = \langle \Sigma_A, \stackrel{must}{\longrightarrow}_A, \stackrel{may}{\longrightarrow}_A, \mathcal{I}_A \rangle$*, a relation,* $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$*, is a* simulation of $\mathcal{C}$ by $\mathcal{A}$*, written* $\mathcal{C} \unlhd_{\mathcal{R}} \mathcal{A}$*, iff* $\mathcal{C} \lhd_{\mathcal{R}} \langle \Sigma_A, \stackrel{may}{\longrightarrow}_A, \mathcal{I}_A \rangle$ *and* $\langle \Sigma_A, \stackrel{must}{\longrightarrow}_A, \mathcal{I}_A \rangle \lhd_{\mathcal{R}^{-1}} \mathcal{C}$*.*

In the literature, may-transitions are sometimes titled "liberal transitions" [8] or "free transitions" [12,13]. Similarly, must-transitions are sometimes called "conservative transitions" [8] or "constrained transitions" [12,13].

Figure 21 shows a variation of the slot-machine example in Figure 20, where a must-transition is included the specification structure, on the right, to ensure that the implementation, on the left, gives its user a chance to win.

The specification in the Figure is a mixed-transition system, and relation $\mathcal{R}$, defined in the Figure, is a simulation. We expect the simulation to be strong enough that it ensures that $start_a \models_{nec} \Diamond won$ means that $start \models \Diamond won$, and indeed, this is the case. Note that $\Diamond\Diamond won$ appears to hold true for state win_a, but this property does not necessarily hold for either of win_1 and win_2—of course, this is because the transition, $win_a \stackrel{may}{\longrightarrow} win_a$ is a may-transition (and not a must-transition), meaning that two moves to a winning state do not necessarily occur in the concrete structure.

The distinction between may- and must-transitions leads to these semantics for the two remaining missing modalities:

$$
\begin{array}{ll}
(1) & a \models_{nec} \Diamond\phi \text{ iff there exists } a' \in \Sigma_A \text{ such that } a \stackrel{must}{\longrightarrow} a' \text{ and } a' \models_{nec} \phi \\
(2) & a \models_{pos} \Box\phi \text{ iff for all } a' \in \Sigma_A, a \stackrel{must}{\longrightarrow} a' \text{ implies } a' \models_{pos} \phi
\end{array}
$$

Clause (1) says, for $\Diamond\phi$ to necessarily hold true, there must be a must-transition that leads to a state where ϕ necessarily holds. This notion is the one illustrated in Figure 21.

Clause (2) says, for $\Box\phi$ to possibly hold true, all must-transitions lead to states where ϕ possibly holds true. Perhaps the contrapositive of (2) is clearer: To refute $\Box\phi$, locate a must-transition that leads to a state that refutes ϕ.

Theorems 25 and 27 are easily extended with the semantics clauses (1) and (2). Figure 22 summarizes the necessarily and possibly interpretations of the full modal mu-calculus for a mixed-transition system.

Not surprisingly, there is a straightforward way to define an abstract structure's may- and must-transitions, given the concrete structure, $\mathcal{C}$, the abstract structure's state set, Σ_A, and a binary relation, $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$ [8]:

$$
\begin{array}{l}
a_1 \stackrel{may}{\longrightarrow} a_2 \text{ iff there exist } c_1, c_2 \in \Sigma_C \\
\qquad \text{such that } c_1 \;\mathcal{R}\; a_1, c_2 \;\mathcal{R}\; a_2, \text{ and } c_1 \longrightarrow c_2 \\
a_1 \stackrel{must}{\longrightarrow} a_2 \text{ iff for all } c_1 \in \Sigma_C, \text{ if } c_1 \;\mathcal{R}\; a_1, \\
\qquad \text{then there exists } c_2 \in \Sigma_C \text{ such that } c_1 \longrightarrow c_2 \text{ and } c_2 \;\mathcal{R}\; a_2
\end{array}
$$

As proved by Cleaveland, Iyer, and Yankovich [8], this definition gives the most precise mixed-transition system with respect to Σ_A and $\mathcal{R}$.

Mixed-transition systems are general—perhaps, too much so. Consider this mixed-transition system, $\mathcal{A} = \langle\{a_0, a_1\}, \{a_0 \stackrel{must}{\longrightarrow} a_1\}, \{\}, \mathcal{I}_A\rangle$, where $\mathcal{I}_A(a_0) = \{p\}$ and $\mathcal{I}_A(a_1) = \{\}$; it is drawn just this simply:

$$a_0^p \xrightarrow{must} a_1$$

Assume $\mathcal{C} \trianglelefteq_{\mathcal{R}} \mathcal{A}$:

$$a \in \Sigma_a \qquad \phi \in \mathcal{L}_{Atom} \qquad p \in N(\mathcal{R}) \qquad q \in P(\mathcal{R}) \qquad Z \in \mathit{Identifier}$$

Necessarily interpretation:

$a \models_{nec} p$ iff $p \in \mathcal{I}_A(a)$

$a \models_{nec} \neg\phi$ iff $a \not\models_{pos} \phi$

$a \models_{nec} \phi_1 \wedge \phi_2$ iff $a \models_{nec} \phi_1$ and $a \models_{nec} \phi_2$

$a \models_{nec} \phi_1 \vee \phi_2$ iff $a \models_{nec} \phi_1$ or $a \models_{nec} \phi_2$

$a \models_{nec} \Box\phi$ iff for all a' such that $a \xrightarrow{may} a', a' \models_{nec} \phi$

$a \models_{nec} \Diamond\phi$ iff there exists a' such that $a \xrightarrow{must} a'$ and $a' \models_{nec} \phi$

Possibly interpretation:

$a \models_{pos} q$ iff $q \in \mathcal{I}_A(a)$

$a \models_{pos} \neg\phi$ iff $a \not\models_{nec} \phi$

$a \models_{pos} \phi_1 \wedge \phi_2$ iff $a \models_{pos} \phi_1$ and $a \models_{pos} \phi_2$

$a \models_{pos} \phi_1 \vee \phi_2$ iff $a \models_{pos} \phi_1$ or $a \models_{pos} \phi_2$

$a \models_{pos} \Box\phi$ iff for all a' such that $a \xrightarrow{must} a', a' \models_{pos} \phi$

$a \models_{pos} \Diamond\phi$ iff there exists a' such that $a \xrightarrow{may} a'$ and $a' \models_{pos} \phi$

The semantics of $\mu Z.\phi$ and $\nu Z.\phi$ remain the same as in Figures 17 and 18.

Fig. 22. Necessarily and possibly interpretations for modal mu-calculus

Without knowing the concrete structure, $\mathcal{C}$, that the above structure simulates, we will assume that $p \in N(\mathcal{R})$ and $p \in P(\mathcal{R})$, for some relation, $\mathcal{R}$. (That is, $\mathcal{R}$ reflects and preserves atomic property p.) In this case, we easily verify that

$$a_0 \models_{nec} \Box p$$

even though there is the must-transition, $a_0 \xrightarrow{must} a_1$, and $a_1 \not\models_{nec} p$! Next, we use the must-transition to deduce that $a_0 \not\models_{pos} \Box p$, and by the semantics of negation, we conclude that

$$a_0 \models_{nec} \neg\Box p$$

producing a logical contradiction! $\mathcal{A}$ is an "inconsistent specification" in the sense that there is no Kripke structure, $\mathcal{C}$, and property-reflecting-and-preserving simulation, $\mathcal{R}$, such that $\mathcal{C} \trianglelefteq_{\mathcal{R}} \mathcal{A}$.

The example raises a reasonable question: When there exists a transition, $a \xrightarrow{must} a'$, why not require $a \xrightarrow{may} a'$ also? After all, if a structure *must* make a transition from a to a', is it not reasonable to state that it *may* make the very same transition?

If we return to the example structure, $\mathcal{A}$, and add the transition, $a_0 \xrightarrow{may} a_1$, to it, we find that the problematic proposition disappears:

$$a_0 \not\models_{nec} \Box p$$

This pleasant situation is developed in the next section.

8 Modal-transition systems

Simply stated, a *modal-transition system* is a mixed-transition system where every must-transition is also a may-transition [28,29]:

Definition 29 *A* Kripke modal-transition system *is a tuple,* $\mathcal{K} = \langle \Sigma_K, \xrightarrow{must}_K, \xrightarrow{may}_K, \mathcal{I}_K \rangle$*, where*

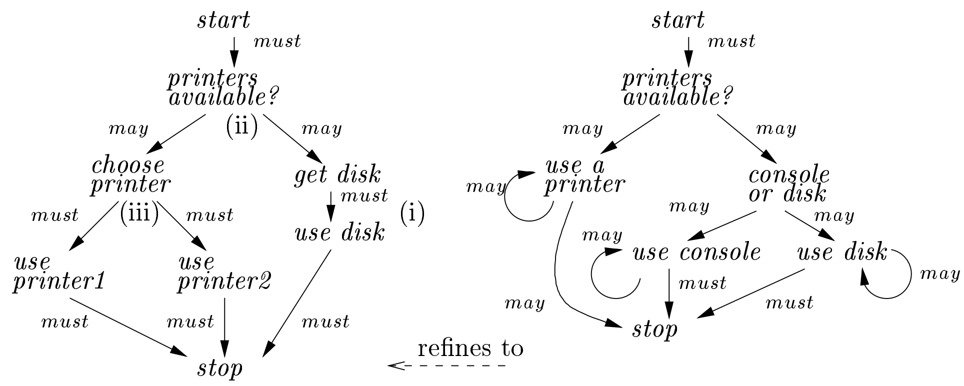
- Σ_K *is a set of states.*
- $\xrightarrow{must}_K, \xrightarrow{may}_K \subseteq \Sigma_K \times \Sigma_K$ *are transition relations with finite image such that* $\xrightarrow{must}_K \subseteq \xrightarrow{may}_K$*.*
- $\mathcal{I}_K : \Sigma_K \to \mathcal{P}(Atom)$ *associates atomic properties,* $\mathcal{I}_K(s) \subseteq Atom$*, to each* $s \in \Sigma_K$*.*

A modal-transition system where $\xrightarrow{must}_K = \xrightarrow{may}_K$ *is* concrete*.*

Larsen and Thomsen [29] proposed modal-transition systems as a framework for stepwise refinement, where a specification (structure) containing must- and may-transitions is stepwise refined into a sequence of structures that preserve the must-transitions and convert some of the may-transitions into must-transitions until finally a *concrete* modal-transition system results. Figure 23 shows an example of a refinement step, where a modal-transition system specifying a print method is refined into a system that implements part of the specification and leaves part for additional refinement.

Modal-transition systems are equally useful for stepwise abstraction, and Figure 24 shows how a sequential program's state space is abstracted to a control-flow graph and then a procedure "block" for flow-insensitive interprocedural analysis.

Note: a *must*-transition implies a *may*-transition:



Notes:
(i) The refinement converted one may-transition into a must-transition and removed several transitions.
(ii) This choice is decided at a later stage, e.g., load-time.
(iii) This nondeterminism must be available for run-time.

Fig. 23. Refinement with modal-transition systems

Note: a *must*-transition implies a *may*-transition:

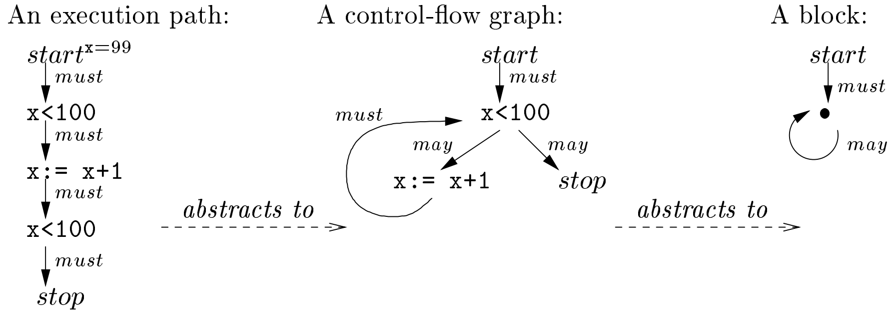


Fig. 24. Abstraction with modal-transition systems

Modal-transition systems possess several pleasant properties not held by mixed-transition systems in general. First,

$$s \models_{nec} \phi \text{ implies } s \models_{pos} \phi$$

holds for all $s \in \Sigma_K$ and $\phi \in \mathcal{L}_{Atom}$, assuming that the atomic properties in ϕ are both reflected and preserved. The proof is an easy induction on the structure of ϕ.

From this result, it immediately follows that the necessarily interpretation is consistent: It is impossible to prove both $s \models_{nec} \phi$ and also $s \models_{nec} \neg\phi$. Similarly, the possibly interpretation is complete: One can always prove $s \models_{pos}$

ϕ or else $s \models_{pos} \neg\phi$.

We can retain the definition of simulation from mixed-transition systems; Larsen prefers the term, *refinement*:

Definition 30 *For Kripke modal-transition systems, $\mathcal{C}$ and $\mathcal{A}$, a relation, $\mathcal{R} \subseteq \Sigma_C \times \Sigma_A$, is a* refinement*, written $\mathcal{C} \trianglelefteq_{\mathcal{R}} \mathcal{A}$, iff*

- $\langle \Sigma_C, \stackrel{may}{\longrightarrow}_C, \mathcal{I}_C \rangle \lhd_{\mathcal{R}} \langle \Sigma_A, \stackrel{may}{\longrightarrow}_A, \mathcal{I}_A \rangle$
- $\langle \Sigma_A, \stackrel{must}{\longrightarrow}_A, \mathcal{I}_A \rangle \lhd_{\mathcal{R}^{-1}} \langle \Sigma_C, \stackrel{must}{\longrightarrow}_C, \mathcal{I}_C \rangle$.

If $\mathcal{C} \trianglelefteq_{\mathcal{R}} \mathcal{A}$, we say that $\mathcal{C}$ *refines* $\mathcal{A}$.

Every modal-transition system is a "consistent specification" in the sense that there is always a concrete structure that refines it—merely convert all the may-transitions into must-transitions and use the identity relation as the refinement relation.

When a refinement preserves a modal-transition system's states, as does the one mentioned in the previous sentence, it is easy to understand the refinement process as a monotonic operation: For a modal-transition system, $\mathcal{K}$, we can represent its $\stackrel{must}{\longrightarrow}_K$ and $\stackrel{may}{\longrightarrow}_K$ components by a function of the form:

$$trans_{\mathcal{K}} : (\Sigma_K \times \Sigma_K) \rightarrow Boolean_\perp$$

where $Boolean_\perp$ is this partially ordered set:

$$Boolean_\perp = \begin{array}{ccc} true & & false \\ & \perp & \end{array}$$

If $(s, s') \in \stackrel{must}{\longrightarrow}_K$, then $trans_{\mathcal{K}}(s, s') = true$; else if $(s, s') \notin \stackrel{may}{\longrightarrow}_K$, then $trans_{\mathcal{K}}(s, s') = false$; else $trans_{\mathcal{K}}(s, s') = \perp$. That is, $s \stackrel{may}{\longrightarrow} s'$ asserts uncertainty about the presence of a transition, which will be resolved by refinement to a concrete structure. If $\Sigma_K = \Sigma_{K'}$, then $\mathcal{K}'$ refines $\mathcal{K}$ iff $trans_{\mathcal{K}} \sqsubseteq_{\Sigma_K \times \Sigma_K \rightarrow Boolean_\perp} trans_{\mathcal{K}'}$.

Of course, a refinement can add or remove states. This was seen in Figure 23, where at Note (iii) in the Figure, states were added and at Note (i), states were removed.

We retain the reflection property for modal-transition systems: For $\mathcal{C} \trianglelefteq_{\mathcal{R}} \mathcal{A}$, if $\mathcal{R}$ reflects atomic properties, then

$$c \, \mathcal{R} \, a \text{ and } a \models_{nec} \phi \text{ imply } c \models_{nec} \phi$$

for all $c \in \Sigma_C$ and $a \in \Sigma_A$. Dually, if $\mathcal{R}$ preserves atomic properties, then

$$c\,\mathcal{R}\,a \text{ and } c \models_{pos} \phi \text{ imply } a \models_{pos} \phi$$

(Of course, if negation is included, then $\mathcal{R}$ must reflect and preserve appropriate atomic properties.)

But there is a stronger result, due to Larsen and Boudol [28], that strengthens the above implications into equivalences. The characterization requires a few additional definitions, which appear in the next subsection.

8.1 Characterizing a modal-transition system by its properties

At the beginning of this paper, we used $\mathcal{I}_K$ to define the atomic properties that held true of the states of a Kripke transition system, $\mathcal{K}$. Once we introduced temporal logic as a language for stating properties of paths, we wedded, step by step, a Kripke transition system, $\mathcal{K}$, to those properties that were necessarily and possibly true of it. In this subsection, we finish the work by characterizing a modal-transition system in terms of the properties it satisfies.

We introduce several new concepts so that we can discuss the properties that hold true of a modal-transition system independently of the system's refinements.

Definition 31 *For a* Kripke modal-transition system $\mathcal{K} = \langle \Sigma_K, \xrightarrow{must}_K, \xrightarrow{may}_K, \mathcal{I}_K \rangle$ *and* $s \in \Sigma_K$*,*

- *We write $\mathcal{K}_s$ to denote that the current or starting state of $\mathcal{K}$ is s. We call $\mathcal{K}_s$ a* pointed modal-transition system. *Let* $\mathit{Derivatives}(\mathcal{K}) = \{\mathcal{K}_s \mid s \in \Sigma_K\}$.
- *When $\mathcal{K}_s$ moves, by means of $s \xrightarrow{may} s'$ (or $s \xrightarrow{must} s'$), we write $\mathcal{K}_s \xrightarrow{may} \mathcal{K}_{s'}$ (or $\mathcal{K}_s \xrightarrow{must} \mathcal{K}_{s'}$, respectively) and say that $\mathcal{K}_{s'}$ is a* derivative *of $\mathcal{K}_s$.*

Remember that a pointed modal-transition system, $\mathcal{K}_s$, is itself a modal transition system. Hence, $\mathit{Derivatives}(\mathcal{K})$ is a *set* of modal-transition systems. The above Definition suggests that, when a pointed modal-transition system makes a transition, it "becomes" a different pointed modal-transition system.

Up to now, we have defined simulation and refinement relations on pairs of state sets, but the definitions readily adapt to pairs of sets of modal-transition systems:

Definition 32 *For modal-transition systems, $\mathcal{C}$ and $\mathcal{A}$, a relation $\mathcal{R} \subseteq \mathit{Derivatives}(\mathcal{C}) \times \mathit{Derivatives}(\mathcal{A})$ is a* refinement, *if whenever $\mathcal{C}_c\ \mathcal{R}\ \mathcal{A}_a$, for*

$c \in \Sigma_C$, $a \in \Sigma_A$, *then*

(1) $\mathcal{C}_c \overset{may}{\longrightarrow} \mathcal{C}_{c'}$ *implies there exists* $\mathcal{A}_{a'}$ *such that* $\mathcal{A}_a \overset{may}{\longrightarrow} \mathcal{A}_{a'}$ *and* $\mathcal{C}_{c'}\ \mathcal{R}\ \mathcal{A}_{a'}$*;*
(2) $\mathcal{A}_a \overset{must}{\longrightarrow} \mathcal{A}_{a'}$ *implies there exists* $\mathcal{C}_{c'}$ *such that* $\mathcal{C}_c \overset{must}{\longrightarrow} \mathcal{C}_{c'}$ *and* $\mathcal{C}_{c'}\ \mathcal{R}\ \mathcal{A}_{a'}$*;*

When $(\mathcal{C}_c, \mathcal{A}_a) \in \mathcal{R}$, we write $\mathcal{C}_c \trianglelefteq_{\mathcal{R}} \mathcal{A}_a$, because $\mathcal{R}$ justifies why pointed modal-transition system $\mathcal{C}_c$ refines pointed modal-transition system $\mathcal{A}_a$. As before, we write $\mathcal{C}_c \trianglelefteq \mathcal{A}_a$ to assert existence of some $\mathcal{R}$ such that $\mathcal{C}_c \trianglelefteq_{\mathcal{R}} \mathcal{A}_a$. Also, we can augment the Definition by the clause,

$$\textit{(3)a} \quad \mathcal{I}_C(c) \supseteq \mathcal{I}_A(a)$$

to define a property-reflecting refinement, and we use

$$\textit{(3)b} \quad \mathcal{I}_C(c) \subseteq \mathcal{I}_A(a)$$

with the Definition to define a property-preserving one. We can include both *3a* and *3b* to define a property-reflecting-and-preserving refinement.

Now, we use the above definitions to do away with specific instances of $\mathcal{R}$: Recall from Section 3.2 that there is a technique for synthesizing the largest possible simulation that exists between two structures. We employ this technique to generate the largest possible refinement for two sets of pointed modal-transition systems:

Definition 33 *For modal-transition systems* $\mathcal{C}$ *and* $\mathcal{A}$*, define* $\mathcal{R}_\infty \subseteq \mathit{Derivatives}(\mathcal{C}) \times \mathit{Derivatives}(\mathcal{A})$ *as* $\mathcal{R}_\infty = \bigcap_{i \geq 0} \mathcal{R}_i$ *, where*

$$\mathcal{R}_0 = \mathit{Derivatives}(\mathcal{C}) \times \mathit{Derivatives}(\mathcal{A})$$

$$\begin{aligned}\mathcal{R}_{i+1} = \{(\mathcal{C}_c, \mathcal{A}_a) \mid\ & c \in \Sigma_C, a \in \Sigma_A, \\ & \textit{(i)}\ \mathcal{C}_c \overset{may}{\longrightarrow} \mathcal{C}_{c'} \textit{ implies there exists } \mathcal{A}_{a'} \textit{ such that} \\ & \quad \mathcal{A}_a \overset{may}{\longrightarrow} \mathcal{A}_{a'} \textit{ and } \mathcal{C}_{c'}\ \mathcal{R}_i\ \mathcal{A}_{a'}; \\ & \textit{(ii)}\ \mathcal{A}_a \overset{must}{\longrightarrow} \mathcal{A}_{a'} \textit{ implies there exists } \mathcal{C}_{c'} \textit{ such that} \\ & \quad \mathcal{C}_c \overset{must}{\longrightarrow} \mathcal{C}_{c'} \textit{ and } \mathcal{C}_{c'}\ \mathcal{R}_i\ \mathcal{A}_{a'}\}\end{aligned}$$

By the usual reasoning techniques [21,28], $\mathcal{R}_\infty$ is proved to define the largest refinement relation on the derivatives of $\mathcal{C}$ and $\mathcal{A}$. In particular, if there is another refinement, $\mathcal{R} \subseteq \mathit{Derivatives}(\mathcal{C}) \times \mathit{Derivatives}(\mathcal{A})$, then $\mathcal{R} \subseteq \mathcal{R}_\infty$. Hence,

$$\mathcal{C}_c \trianglelefteq \mathcal{A}_a \text{ iff } (\mathcal{C}_c, \mathcal{A}_a) \in \mathcal{R}_\infty$$

for all $c \in \Sigma_C$ and $a \in \Sigma_A$. This result lets us ignore specific refinement relations in future calculations.

As noted earlier, we can define a largest property-reflecting refinement by adding Clause *3a* to the definition of $\mathcal{R}_{i+1}$, and we can define a largest property-preserving refinement by adding Clause *3b* to $\mathcal{R}_{i+1}$. From hereon, we write

$$\mathcal{C}_c \trianglelefteq_{refl} \mathcal{A}_a$$

to denote there is a property-reflecting refinement between $\mathcal{C}_c$ and $\mathcal{A}_a$, and

$$\mathcal{C}_c \trianglelefteq_{pres} \mathcal{A}_a$$

to denote existence of a property-preserving one.

Now, we can see how a pointed modal-transition system is characterized by the set of propositions that necessarily (or, possibly) hold true for it. First, define the syntax of $\mathcal{HM}$ as follows:

$$\phi \ ::= \ q \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \phi_1 \vee \phi_2 \ \mid \ \Box\phi \ \mid \ \Diamond\phi$$

for $q \in Atom$. $\mathcal{HM}$ is of course the modal mu-calculus less negation and recursion. Although this logic is less expressive than the full modal mu-calculus, it is expressive enough to state the main results of this section:

Definition 34 *For pointed modal-transition system $\mathcal{K}_s$, define*

(1) $\mathcal{M}_{nec}(\mathcal{K}_s) = \{\phi \in \mathcal{HM} \mid s \models_{nec} \phi\}$
(2) $\mathcal{M}_{pos}(\mathcal{K}_s) = \{\phi \in \mathcal{HM} \mid s \models_{pos} \phi\}$

Of course, $\mathcal{M}_{nec}(\mathcal{K}_s)$ names the propositions in $\mathcal{HM}$ that necessarily hold true for $\mathcal{K}_s$, and $\mathcal{M}_{pos}(\mathcal{K}_s)$ name those that possibly hold true (assuming property reflection and preservation, respectively).

Theorem 35 *[28]: For pointed modal-transition systems $\mathcal{C}_c$ and $\mathcal{A}_a$,*

(1) $\mathcal{C}_c \trianglelefteq_{refl} \mathcal{A}_a$ *iff* $\mathcal{M}_{nec}(\mathcal{C}_c) \supseteq \mathcal{M}_{nec}(\mathcal{A}_a)$
(2) $\mathcal{C}_c \trianglelefteq_{pres} \mathcal{A}_a$ *iff* $\mathcal{M}_{pos}(\mathcal{C}_c) \subseteq \mathcal{M}_{nec}(\mathcal{A}_a)$

We outline part of the proof for the first claim. (The basic proof idea comes from Hennessey and Milner [21]; see Larsen [28] also.) The "only if" part follows from Theorem 25. The "if" part is proved with $\mathcal{R}_\infty$ and the contrapositive: If $\mathcal{C}_c \not\trianglelefteq_{\mathcal{R}_\infty} \mathcal{A}_a$, then there is some $\phi \in \mathcal{HM}$ such that $a \models_{nec} \phi$ but $c \not\models_{nec} \phi$.

We prove this result for all derivatives of $\mathcal{C}$ and $\mathcal{A}$: For each such pair of derivatives, $\mathcal{C}_s$, and $\mathcal{A}_t$, if $\mathcal{C}_s \not\trianglelefteq_{\mathcal{R}_\infty} \mathcal{A}_t$, then there is some minimal $i \geq 0$ such that $\mathcal{C}_s \not\trianglelefteq_{\mathcal{R}_i} \mathcal{A}_t$. Keeping this in mind, we systematically derive counterexamples, $\phi_{s,t}$, for all such pairs, s, t, by induction on the value of i.

For $i = 0$, the situation is vacuous. For $i = n+1$, there are three possibilities:

(1) $\mathcal{C}_c \xrightarrow{may} \mathcal{C}_{c'}$ but for all $\mathcal{A}_a \xrightarrow{may} \mathcal{A}_{a_j}$, $j \in 1..m$, not $\mathcal{C}_{c'}\ \mathcal{R}_n\ \mathcal{A}_{a'_j}$: By the induction hypothesis, for each pair, c', a_j, there is a proposition, ϕ_{c',a_j}, such that $a_j \models_{nec} \phi_{c',a_j}$ but $c' \not\models_{nec} \phi_{c',a_j}$. Therefore, the proposition, $\phi_{c,a} = \Box(\phi_{c',a_1} \vee ... \vee \phi_{c',a_m})$ is the desired counterexample.

(2) $\mathcal{A}_a \xrightarrow{must} \mathcal{A}_{a'}$ but for all $\mathcal{C}_c \xrightarrow{must} \mathcal{C}_{c_j}$, $j \in 1..m$, not $\mathcal{C}_{c_j}\ \mathcal{R}_n\ \mathcal{A}_{a'}$: This case proceeds like the previous one, and the counterexample is $\phi_{c,a} = \Diamond(\phi_{c',a_1} \wedge ... \wedge \phi_{c',a_m})$.

(3) $\mathcal{I}_C(c) \not\supseteq \mathcal{I}_A(a)$, that is, there is some $q \in Atom$ such that $a \models_{nec} q$ and $c \not\models_{nec} q$. Choose $\phi_{c,a}$ to be q.

This completes the proof.

The Theorem is of course preserved when we add the recursion operators to the logic, $\mathcal{HM}$. If we study the collection of property-reflecting-and-preserving refinements, we can safely add negation to the logic as well.

The Theorem is of value because it identifies a pointed modal-transition system by the properties that it satisfies, up to equivalence-by-mutual-refinement. In practice, the Theorem can be used to quickly refute when one structure is not a refinement or abstraction of another by finding a property that one structure possesses that the other does not.

With similar machinery, Theorem 35 can also be proved true for mixed-transition systems [22].

Not surprisingly, Theorem 35 can be adapted into a result that characterizes a pointed modal-transition system, $\mathcal{K}_s$, by just *one* proposition, $\phi_{\mathcal{K}_s}$, such that $\mathcal{K}'_{s'} \trianglelefteq_{refl} \mathcal{K}_s$ iff $s' \models_{nec} \phi_{\mathcal{K}_s}$. Proposition $\phi_{\mathcal{K}_s}$ is constructed with the assistance of Theorem 35 and the recursion operator, ν; see Larsen [28] for details. Conversely, Boudol and Larsen characterize which sets of formulas in $\mathcal{HM}$ can be implemented by modal transition systems [2].

Acknowledgements

Yoshiki Kinoshita and the other organizers of the Workshop on Refinement and Abstraction are thanked for encouraging me to write this paper; Anindya Banerjee, Michael Huth, Radha Jagadeesan, Markus Müller-Olm, and the anonymous referee

supplied useful comments and corrections. Hiroshi Watanabe spotted an error in Section 3.3, which led to a significant correction.

References

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison Wesley, 1986.

[2] G. Boudol and K. Larsen. Graphical vs. logical specifications. *Theoretical Computer Science*, 106:3–20, 1992.

[3] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[4] 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.

[5] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[6] 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.

[7] 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.

[8] R. Cleaveland, P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In *SAS'95: Proc. 2d. Static Analysis Symposium*, Lecture Notes in Computer Science 983, pages 51–63. Springer, 1995.

[9] 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.

[10] 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.

[11] P. Cousot and R. Cousot. Higher-order abstract interpretation. In *Proc. IEEE Int'l. Conf. Programming Languages.* IEEE Press, 1994.

[12] D. Dams. *Abstract interpretation and partition refinement for model checking.* PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1996.

[13] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM TOPLAS*, 19:253–291, 1997.

[14] M. Dwyer and C. Pasareanu. Filter-based model checking of partial systems. In *Proc. ACM SIGSOFT Symp. on Foundations of Software Engineering (FSE)*, 1998.

[15] M. Dwyer and D. Schmidt. Limiting state explosion with filter-based refinement. In Annalisa Bossi, editor, *International Workshop on Verification, Model Checking and Abstract Interpretation, Port Jefferson, Long Island, N.Y.*, `http://www.cis.ksu.edu/~schmidt/papers/filter.ps.Z`, 1997.

[16] A. Emerson. Temporal and modal logic. In J. vanLeeuwen, editor, *Handbook of Theoretical Computer Science: Volume B*, pages 995–1072. Elsevier, 1990.

[17] M. Fowler. *UML Distilled*. Addison-Wesley, 1997.

[18] C. Gunter. *Semantics of Programming Languages*. MIT Press, Cambridge, MA, 1992.

[19] D. Harel. Statecharts: a visual formalization for complex systems. *Science of Computer Programming*, 8, 1987.

[20] M. Hecht. *Flow Analysis of Computer Programs*. Elsevier, 1977.

[21] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[22] M. Huth, R. Jagadeesan, and D. Schmidt. A domain equation for refinement in mixed transition systems. Technical report, Computer Science Department, Kansas State University, 2000.

[23] B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic partitioning in analyses of numerical properties. In *Static Analysis Symposium '99*, volume 1694 of *Lecture Notes in Computer Science*, pages 39–50. Springer-Verlag, 1999.

[24] 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.

[25] P. Kelb. Model checking and abstraction: a framework preserving both truth and failure information. Technical Report Technical report, OFFIS, University of Oldenburg, Germany, 1994.

[26] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[27] K. Larsen. Proof systems for Hennessy-Milner logic with recursion. In M. Duachet and M. Nivat, editors, *CAAP88*, number 299 in Lecture Notes in Computer Science. Springer-Verlag, 1988.

[28] K. Larsen. Modal specifications. In J. Sifakis, editor, *CAV'89*, number 407 in Lecture Notes in Computer Science, pages 232–246. Springer-Verlag, 1989.

[29] K. Larsen and B. Thomsen. A modal process logic. In *Third IEEE Symp. Logic in Computer Science*, pages 203–210. IEEE Press, 1988.

[30] F. Levi. A symbolic semantics for abstract model checking. In *Static Analysis Symposium: SAS'98*, volume 1503 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[31] 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.

[32] C. McGowan. An inductive proof technique for interpreter equivalence. In R. Rustin, editor, *Formal Semantics of Programming Languages*, pages 139–148. Prentice-Hall, 1972.

[33] 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.

[34] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Lecture Notes in Computer Science 92, 1980.

[35] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[36] 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.

[37] 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.

[38] F. Nielson. Two-level semantics and abstract interpretation. *Theoretical Computer Science*, 69(2):117–242, 1989.

[39] D. Park. Concurrency and automata in infinite strings. Lecture Notes in Computer Science 104, pages 167–183. Springer, 1981.

[40] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[41] H. Saidi. Model checking guided abstraction and analysis. In J. Palsberg, editor, *Static Analysis 2000*, number 1824 in Lecture Notes in Computer Science, pages 377–396. Springer-Verlag, 2000.

[42] 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.

[43] 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.

[44] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.