Action Codes

Frits Vaandrager ✉ 🏠 🆔
Radboud University, Nijmegen, the Netherlands

Thorsten Wißmann ✉ 🏠 🆔
Radboud University, Nijmegen, the Netherlands
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

──── **Abstract** ────

We provide a new perspective on the problem how high-level state machine models with abstract actions can be related to low-level models in which these actions are refined by sequences of concrete actions. We describe the connection between high-level and low-level actions using *action codes*, a variation of the prefix codes known from coding theory. For each action code $\mathcal{R}$, we introduce a *contraction* operator $\alpha_{\mathcal{R}}$ that turns a low-level model $\mathcal{M}$ into a high-level model, and a *refinement* operator $\varrho_{\mathcal{R}}$ that transforms a high-level model $\mathcal{N}$ into a low-level model. We establish a Galois connection $\varrho_{\mathcal{R}}(\mathcal{N}) \sqsubseteq \mathcal{M} \Leftrightarrow \mathcal{N} \sqsubseteq \alpha_{\mathcal{R}}(\mathcal{M})$, where $\sqsubseteq$ is the well-known simulation preorder. For conformance, we typically want to obtain an overapproximation of model $\mathcal{M}$. To this end, we also introduce a *concretization* operator $\gamma_{\mathcal{R}}$, which behaves like the refinement operator but adds arbitrary behavior at intermediate points, giving us a second Galois connection $\alpha_{\mathcal{R}}(\mathcal{M}) \sqsubseteq \mathcal{N} \Leftrightarrow \mathcal{M} \sqsubseteq \gamma_{\mathcal{R}}(\mathcal{N})$. Action codes may be used to construct adaptors that translate between concrete and abstract actions during learning and testing of Mealy machines. If Mealy machine $\mathcal{M}$ models a black-box system then $\alpha_{\mathcal{R}}(\mathcal{M})$ describes the behavior that can be observed by a learner/tester that interacts with this system via an adaptor derived from code $\mathcal{R}$. Whenever $\alpha_{\mathcal{R}}(\mathcal{M})$ implements (or conforms to) $\mathcal{N}$, we may conclude that $\mathcal{M}$ implements (or conforms to) $\gamma_{\mathcal{R}}(\mathcal{N})$.

1 Introduction

Labeled transition systems (LTSs) constitute one of the most fundamental modeling mechanisms in Computer Science. An LTS is a rooted, directed graph whose nodes represent *states* and whose edges are labeled with *actions* and represent *state transitions*. LTS-based formalisms such as Finite Automata [19], Finite State Machines [23], I/O automata [24], IOTSs [33], and process algebras [4] have been widely used to model and analyze a broad variety of reactive systems, and a rich body of theory has been developed for them.

In order to manage the complexity of computer-based systems, designers structure such systems into hierarchical layers. This allows them to describe and analyze systems at different levels of abstraction. Many LTS-based frameworks have been proposed to formally relate

(a) Original System   **(b)** Existing action refinements   **(c)** Desired action refinement behavior

■ **Figure 1** Example for the (lack of) preservation of determinism in action refinement

models at different hierarchical levels, e.g. [4, 14, 25, 37]. In most of these frameworks, the states of a high-level LTS correspond to sets of states of a low-level LTS via simulation or bisimulation-like relations. However, the actions are fixed and considered to be atomic. Actions used at a lower level of abstraction can be hidden at a higher level, but higher-level actions will always be available at the lower level. For this reason, Rensink & Gorrieri [16, 29] argue that these (bi)simulations relate systems at the same conceptual level of abstraction, and therefore they call them *horizontal* implementation relations. They contrast them with *vertical* implementation relations that compare systems that belong to conceptually different abstraction levels, and have different alphabets of actions.

A prototypical example of a hierarchical design is a computer network. To reduce design complexity, such a network is organized as a stack of layers or levels, each one built upon the one below it [32]. Examples are the transport layer, with protocols such as TCP and UDP, and the physical layer, concerned with transmitting raw bits over a communication channel. Now consider a host that receives an TCP packet in some state s. If P is the set of possible packets then, in an LTS model of the transport layer, state s will contain outgoing transitions labeled with action $receive(p)$, for each $p \in P$. At the physical layer, however, receipt of a packet corresponds to a sequence of $receive(b)$ actions, with b a bit in $\{0, 1\}$. Only after the final bits have arrived, the host knows which packet was actually received. Mechanisms for transforming high-level actions into sequences (or processes) of low-level actions have been addressed extensively in work on action refinements [16]. These approaches, however, are unable to describe the above scenario in a satisfactory manner and somehow assume that a host upfront correctly guesses the packet that it will receive, even before the first bit has arrived. In order to illustrate this problem, we consider the simplified example of an LTS with a distinguished initial state, displayed in Figure 1a, which accepts either input a or input b. At a lower level of abstraction, input a is implemented by three consecutive inputs 1 4 1, whereas input b is implemented by action sequence 1 4 2 (the ASCII encodings of a and b in octal format). An action refinement operator will replace the a-transition in Figure 1a by a sequence of three consecutive transitions with labels 1, 4 and 1, respectively, and will handle the b-transition in an analogous manner. Thus, action refinement introduces a nondeterministic choice (Figure 1b), rather than the deterministic behavior that one would like to see (Figure 1c). As a consequence of this and other limitations, refinement operators have not found much practical use [16].

Based on the observation that any action can be modeled as a state change, some authors (e.g. [2, 10, 22]) prefer modeling formalisms in which the term "action" is only used informally, and Kripke structures rather than LTSs are used to model systems. These state-based approaches have the advantage that a distinction between horizontal and vertical implementation relations is no longer needed, and a single implementation relation suffices. Purely state-based approaches, however, are problematic in cases where we need to interact with a black-box system and (by definition) we have no clue about the state of this system.

Black-box systems prominently occur in the areas of model-based testing [34] and model learning [35]. In these application areas, use of LTSs makes sense and there is a clear practical need for formalisms that allow engineers to relate actions at different levels of abstraction.

Van der Bijl et al [7], for instance, observe that in model-based testing specifications are usually more abstract than the System Under Test (SUT). This means that generated test cases may not have the required level of detail, and often a single abstract action has to be translated (either manually or by an adaptor) to a sequence of concrete actions that are applied to the SUT. Van der Bijl et al [7] study a restricted type of action refinement in which a single input is refined into a sequence of inputs, and implement this in a testing tool.

Also in model learning, typically an adaptor is placed in between the SUT and the learner, to take care of the translation between abstract and concrete actions. For example, in a case study on hand-held smartcard readers for Internet banking, Chalupar et al [9] used abstract inputs that combine several concrete inputs in order to accelerate the learning process and reduce the size of the learned model. In particular, they introduced a single abstract input COMBINED_PIN corresponding to a USB command, followed by a 4-digit PIN code, followed by an OK command. Fiterău-Broştean et al [12] used model learning for a comprehensive analysis of DTLS implementations, and found four serious security vulnerabilities, as well as several functional bugs and non-conformance issues. Handshakes in (D)TLS are defined over flights of messages. Hence, (D)TLS entities are often expected to produce multiple messages before expecting a response. During learning, Fiterău-Broştean et al [12] used an adaptor that contracted multiple messages from the SUT into a single abstract output. Also in other case studies on TLS [30], Wi-Fi [31] and SSH [36, 13], multiple outputs from the SUT were contracted into a single abstract output. Verleg [36] used a single abstract input to execute the entire key re-exchange during learning higher layers of SSH.

In this article, we provide answers to two fundamental questions: (1) How can we formalize the concept of an *adaptor* that translates between abstract and concrete actions?, and (2) Suppose the behavior of an SUT is described by an unknown, concrete model $\mathcal{M}$, and suppose a learner interacts with this SUT through an adaptor and learns an abstract model $\mathcal{N}$. What can we say about the relation between $\mathcal{M}$ and $\mathcal{N}$?

We answer the first question by introducing *action codes*, a variation of the prefix codes known from coding theory [5]. Action codes describe how high-level actions are converted into sequences of low-level actions, and vice versa. This makes them different from action refinements, which specify how high-level actions can be translated into low-level processes, but do not address the reverse translation. Our notion of an action code captures adaptors that are used in practice, and in particular those described in the case studies listed above.

In order to answer the second question we introduce, for each action code $\mathcal{R}$, a *contraction* operator $\alpha_{\mathcal{R}}$ that turns a low-level model $\mathcal{M}$ into a high-level model by contracting concrete action sequences of $\mathcal{M}$ according to $\mathcal{R}$. We also introduce the left adjoint of $\alpha_{\mathcal{R}}$, the *refinement* operator $\varrho_{\mathcal{R}}$ that turns a high-level model $\mathcal{M}$ into a low-level model by refining abstract actions of $\mathcal{N}$ according to $\mathcal{R}$. This refinement operator, for instance, maps the LTS of Figure 1a to the LTS of Figure 1c. We establish a Galois connection $\varrho_{\mathcal{R}}(\mathcal{N}) \sqsubseteq \mathcal{M} \Leftrightarrow \mathcal{N} \sqsubseteq \alpha_{\mathcal{R}}(\mathcal{M})$, where $\sqsubseteq$ denotes the simulation preorder. So if an abstract model $\mathcal{N}$ implements contraction $\alpha_{\mathcal{R}}(\mathcal{M})$, then the refinement $\varrho_{\mathcal{R}}(\mathcal{N})$ implements concrete model $\mathcal{M}$, and vice versa.

In practice, we typically want to obtain an overapproximation of concrete model $\mathcal{M}$. To this end, we introduce the right adjoint of $\alpha_{\mathcal{R}}$, the *concretization* operator $\gamma_{\mathcal{R}}$. This operator behaves like the refinement operator, but adds arbitrary behavior at intermediate points (cf. the demonic completion of [6]). We establish another Galois connection: $\alpha_{\mathcal{R}}(\mathcal{M}) \sqsubseteq \mathcal{N} \Leftrightarrow \mathcal{M} \sqsubseteq \gamma_{\mathcal{R}}(\mathcal{N})$. This connection is useful, because whenever we have established that $\alpha_{\mathcal{R}}(\mathcal{M})$

implements (or conforms to) $\mathcal{N}$, it allows us to conclude that $\mathcal{M}$ implements (or conforms to) $\gamma_\mathcal{R}(\mathcal{N})$.

We show that, in a setting of Mealy machines (Finite State Machines without finiteness requirement), an *adaptor* can be constructed for any action code for which a winning strategy exists in a certain 2-player game. If a learner/tester interacts with an SUT via an adaptor generated from such an action code $\mathcal{R}$, and the SUT is modeled by Mealy machine $\mathcal{M}$, then from the learner/tester perspective, the composition of adaptor and SUT will behave like $\alpha_\mathcal{R}(\mathcal{M})$. Thus, if a learner succeeds to learn an abstract model $\mathcal{N}$ such that $\mathcal{N} \approx \alpha_\mathcal{R}(\mathcal{M})$ then, using the Galois connections, the learner may conclude that $\varrho_\mathcal{R}(\mathcal{N}) \sqsubseteq \mathcal{M} \sqsubseteq \gamma_\mathcal{R}(\mathcal{N})$.

The remainder of this article is structured as follows. We start with a preliminary Section 2 that introduces basic notations and results for LTSs. Next, action codes and the contraction operator are introduced in Section 3. After describing the refinement operator, we establish our first Galois connection in Section 4. Next we define concretization and establish our second Galois connection in Sections 5. Section 6 explains how action codes can be composed, and shows that contraction and refinement commute with action code composition. Section 7 describes how adaptors can be constructed from action codes. Finally, Section 8 contains a discussion of our results and identifies directions for future research. Almost all proofs are formalized in Coq and can be accessed via `https://arxiv.org/src/2301.00199/anc` – we mark those results with a Coq icon 🐓. Appendix A contains comments on the Coq formalization and Appendix B contains full proofs (in natural language) and additional remarks.

2   Preliminaries

If Σ is a set of symbols then Σ^* denotes the set of all finite words over Σ, and Σ^+ the set of all non-empty words. We use ε to denote the empty word, so e.g. $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. Concatenation of words $u, w \in \Sigma^*$ is notated $u \cdot w$ (or simply $u\,w$). We write $u \leq w$ if u is a prefix of w, i.e. if there is $v \in \Sigma^*$ with $u\,v = w$. We write $|w|$ to denote the length of word w.

We use $f \colon X \rightharpoonup Y$ to denote a partial map f from X to Y and write $\mathsf{dom}(f) \subseteq X$ for its domain, i.e. set of $x \in X$ on which f is defined. The *image* $\mathsf{im}(f)$ of a partial map $f \colon X \rightharpoonup Y$ is the set of elements of Y it can reach: $\mathsf{im}(f) := \{f(x) \mid x \in \mathsf{dom}(f)\} \subseteq Y$.

▶ **Definition 2.1** (🐓). *For a set A of action labels, a* labeled transition system (LTS) *is a tuple $\mathcal{M} = \langle Q, q_0, \longrightarrow \rangle$ where Q is a set of states, $q_0 \in Q$ is a starting state, and $\longrightarrow\, \subseteq Q \times A \times Q$ is a transition relation. We write* $\mathsf{LTS}(A)$ *for the class of all LTSs with labels from A. We refer to the three components of an LTS $\mathcal{M}$ as $Q^\mathcal{M}$, $q_0^\mathcal{M}$ and $\longrightarrow_\mathcal{M}$, respectively, and introduce the following notation:*

- $q \xrightarrow{a} q'$ *denotes $(q, a, q') \in\, \longrightarrow$; $q \xrightarrow{a}$ denotes that there is some q' with $q \xrightarrow{a} q'$;*
- $q \xRightarrow{w} q'$ *for $w \in A^*$ denotes that there are finite sequences $a_1, \ldots, a_n \in A$, $r_0, \ldots, r_n \in Q$ such that $w = a_1 \cdots a_n$, and $r_0 = q$, $r_n = q'$ and $r_{i-1} \xrightarrow{a_i} r_i$ for all $1 \leq i \leq n$;*
- $q \xRightarrow{w}$ *denotes that there is q' such that $q \xRightarrow{w} q'$;*
- $q \in Q$ *is* reachable *if there is $w \in A^*$ such that $q_0 \xRightarrow{w} q$.*

A special class of LTSs that is frequently used in conformance testing and model learning are *Mealy machines*. Mealy machines with a finite number of states are commonly referred to as *Finite State Machines*.

▶ **Definition 2.2.** *For non-empty sets of inputs I and outputs O, a (non-deterministic) Mealy machine $\mathcal{M} \in \mathsf{LTS}(I \times O)$ is an LTS where the labels are pairs of an input and an output. We write $q \xrightarrow{i/o} q'$ to denote that $(q, (i, o), q') \in \longrightarrow$. Whenever we omit a symbol in predicate $q \xrightarrow{i/o} q'$ this is quantified existentially. Thus, $\xrightarrow{i/o}$ if there are q and q' s.t. $q \xrightarrow{i/o} q'$, $q \xrightarrow{i/} q'$ if there is an o s.t. $q \xrightarrow{i/o} q'$, and $q \xrightarrow{i/}$ if there is a q' s.t. $q \xrightarrow{i/} q'$.*
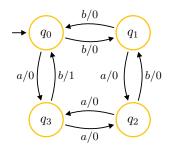

Figure 2 A Mealy machine.

▶ **Example 2.3** (🌵). Figure 2 visualizes a simple Mealy machine with inputs $\{a, b\}$ and outputs $\{0, 1\}$. The machine always outputs 0 in response to an input, except in one specific situation. Output 1 is produced in response to input b if the previous input was a and the number of preceding inputs is odd. The machine has four states q_0, q_1, q_2 and q_3, with starting state q_0 marked by an incoming arrow. In states q_0 and q_2 the number of preceding inputs is always even, whereas in states q_1 and q_3 it is always odd. In states q_2 and q_3 the previous input is always a, whereas in states q_0 and q_1 either the previous input is b, or no input has occurred yet. Thus, only in state q_3 input b triggers output 1.

We introduce some notation and terminology for LTSs.

▶ **Definition 2.4** (🌵). *Let $\mathcal{M} = \langle Q, q_0, \longrightarrow \rangle \in \mathsf{LTS}(A)$ be an LTS. We say that*

- *$\mathcal{M}$ is* deterministic *if, whenever $q \xrightarrow{a}$ for some q and a, there is a unique q' with $q \xrightarrow{a} q'$.*
- *$\mathcal{M}$ is a* tree-shaped *if each state $q \in Q$ can be reached via a unique sequence of transitions from state q_0.*
- *$q \in Q$ is a* leaf, *notated $q \nrightarrow$, if there is no $a \in A$ with $q \xrightarrow{a}$.*
- *$\mathcal{M}$ is* grounded *if every state $q \in Q$ has a path to a leaf.*

We can now define the set of traces of an LTS:

▶ **Definition 2.5** (🌵). *Let $\mathcal{M} = \langle Q, q_0, \longrightarrow \rangle \in \mathsf{LTS}(A)$. A word $w \in A^*$ is a* trace *of state $q \in Q$ if $q \xRightarrow{w}$, and a* trace *of $\mathcal{M}$ is it is a trace of q_0. We write $trace(\mathcal{M})$ for the set $\{w \in A^* \mid q_0 \xRightarrow{w}\}$ of all traces of $\mathcal{M}$.*

▶ **Definition 2.6** (Simulation, 🌵). *For $\mathcal{M}, \mathcal{N} \in \mathsf{LTS}(A)$, a* simulation *from $\mathcal{M}$ to $\mathcal{N}$ is a relation $S \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$ such that*

1. *$q_0^{\mathcal{M}} \, S \, q_0^{\mathcal{N}}$ and*
2. *if $q_1 \, S \, q_2$ and $q_1 \xrightarrow{a}_{\mathcal{M}} q_1'$ then there exists a state q_2' such that $q_2 \xrightarrow{a}_{\mathcal{N}} q_2'$ and $q_1' \, S \, q_2'$.*

We write $\mathcal{M} \sqsubseteq \mathcal{N}$ if there exists a simulation from $\mathcal{M}$ to $\mathcal{N}$.

It is a classical result that trace inclusion coincides with the simulation preorder for deterministic labeled transition systems (see e.g. [26]):

▶ **Lemma 2.7** (🌵). *For all $\mathcal{M}, \mathcal{N} \in \mathsf{LTS}(A)$ where $\mathcal{N}$ is deterministic: $trace(\mathcal{M}) \subseteq trace(\mathcal{N})$ iff $\mathcal{M} \sqsubseteq \mathcal{N}$.*

We will often consider LTSs up to isomorphism of their reachable parts:

▶ **Definition 2.8** (Isomorphism, 🌵). *For $\mathcal{M}, \mathcal{N} \in \mathsf{LTS}(A)$, an* isomorphism *from $\mathcal{M}$ to $\mathcal{N}$ is a bijection $f \colon Q_R^{\mathcal{M}} \to Q_R^{\mathcal{N}}$, where:*

1. $Q_R^{\mathcal{M}} \subseteq Q^{\mathcal{M}}$ and $Q_R^{\mathcal{N}} \subseteq Q^{\mathcal{N}}$ are the subsets of reachable states in $\mathcal{M}$ and $\mathcal{N}$, respectively;
2. $f(q_0^{\mathcal{M}}) = q_0^{\mathcal{N}}$, and
3. $q \xrightarrow{a}_{\mathcal{M}} q'$ iff $f(q) \xrightarrow{a}_{\mathcal{N}} f(q')$, for all $q, q' \in Q_R^{\mathcal{M}}$, $a \in A$.

We write $\mathcal{M} \cong \mathcal{N}$ if there exists an isomorphism from $\mathcal{M}$ to $\mathcal{N}$.

Note that $\cong$ is an equivalence relation on $\mathsf{LTS}(A)$, and that $\mathcal{M} \cong \mathcal{N}$ implies $\mathcal{M} \sqsubseteq \mathcal{N}$, since each isomorphism (when viewed as a relation) is trivially a simulation.

3   Action Codes

Action codes describe how to translate between two action label alphabets, for example from A to B. Intuitively, we understand the first alphabet A as the actions at the lower, concrete level, and the second alphabet B as the actions at the higher, more abstract level. In an action code, a single abstract action $b \in B$ corresponds to a finite, non-empty sequence of concrete actions $a_1 \cdots a_n$ in A. Essentially, action codes are just a special type of *prefix codes* [5]. We provide two equivalent definitions of action codes: one via tree-shaped LTSs and one via partial maps.

▶ **Definition 3.1** (Action code, 🐦). *For sets of action labels A and B, a* (tree-shaped) *action code $\mathcal{R}$ from A to B is a structure $\mathcal{R} = \langle \mathcal{M}, l \rangle$, with $\mathcal{M} = \langle R, r_0, \longrightarrow \rangle \in \mathsf{LTS}(A)$ a deterministic, tree-shaped LTS with L being the set of non-root leaves $L \subseteq R \setminus \{r_0\}$ and an injective function $l \colon L \to B$. We write $\mathsf{Code}(A, B)$ for all action codes from A to B.*

The injectivity of l and the tree-shape ensure that every abstract $b \in B$ is represented by at most one $w \in A^+$.

▶ **Example 3.2.** Figure 3 shows an action code for a fragment of the ASCII encoding in octal format, e.g., 115 encodes the letter M, 145 encodes the letter e, etc.
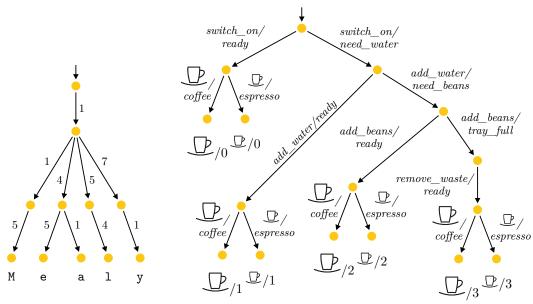

🟧 **Figure 3** Action code for a fragment of ASCII.   🟧 **Figure 4** Action code for a coffee machine.

▶ **Example 3.3.** Figure 4 shows an action code for the activity of getting a cup of coffee or espresso, in the special case of Mealy machines, i.e. where $A = I \times O$ and $B = I' \times O'$ are sets of input/output-pairs. Rather than the full sequence of interventions that is required in order to get a drink, the abstract input/output pair only reports on the type of drink that was ordered and the number of interventions that occurred.

The definition of action codes as LTSs allows an intuitive visualization. For easier mathematical reasoning, we characterize action codes also in terms of maps:

▶ **Definition 3.4** (🌿). *A* (map-based) action code *from A to B is a partial map $f \colon B \rightharpoonup A^+$ which is* prefix-free, *by which we mean that for all $b, b' \in \mathsf{dom}(f)$,*

$$f(b) \le f(b') \qquad implies \qquad b = b'. \tag{1}$$

In the following, we show that these prefix-free partial maps bijectively correspond to the tree-shaped LTSs:

▶ **Lemma 3.5** (🌿). *Every tree-shaped action code $\mathcal{R} \in \mathsf{Code}(A, B)$ induces a unique map-based action code $f \colon B \rightharpoonup A^+$ with the property that for all $b \in B$, $w \in A^+$:*

$$f(b) = w \qquad iff \qquad \exists r \in L \colon r_0 \xRightarrow{w}_{\mathcal{R}} r, \;\; l(r) = b \tag{2}$$

▶ **Lemma 3.6** (🌿). *For each map-based action code $f \colon B \rightharpoonup A^+$, there is (up to isomorphism) a unique tree-shaped action code $\mathcal{R} \in \mathsf{Code}(A, B)$ which is grounded and satisfies (2).*

▶ **Example 3.7** (🌿). For the uniqueness in Lemma 3.6, we use groundedness, because for $A = \{a\}$ and any B, the action codes

$$\mathcal{R} := \left(\rightarrow \bullet \xrightarrow{\;a\;} \bullet \xrightarrow{\;a\;} \bullet \xrightarrow{\;a\;} \cdots \right) \quad \text{and} \quad \mathcal{S} := \left(\rightarrow \bullet \right).$$

both have no non-root leaves, and so they both induce the empty partial map $f \colon B \rightharpoonup A^+$ via Lemma 3.5. This f is undefined for all $b \in B$. And indeed, $\mathcal{R}$ and $\mathcal{S}$ are not isomorphic. The issue is that while the finite $\mathcal{S}$ is grounded, the infinite $\mathcal{R}$ is not grounded. So $\mathcal{R}$ contains subtrees which do not contribute anything to the partial map f but which hinder the existence of an isomorphism.
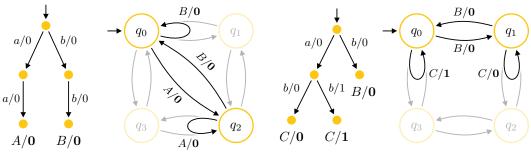
Having shown the correspondence between tree-shaped and map-based action codes $\mathsf{Code}(A, B)$, we can switch between the two views in proofs. Mostly, we use the tree-shaped version for visualization and the map-based version for mathematical reasoning.

Consider a concrete $\mathcal{M} \in \mathsf{LTS}(A)$, together with an action code $\mathcal{R}$ from A to B. We can construct an abstract LTS for the action labels B by walking through $\mathcal{M}$ with seven-league boots, repeatedly choosing input sequences that correspond to runs to some leaf of $\mathcal{R}$, and then contracting this sequence to a single abstract transition.

▶ **Notation 3.8.** In the rest of the paper, we introduce operators $\alpha_{\mathcal{R}}$, $\varrho_{\mathcal{R}}$, $\gamma_{\mathcal{R}}$ on LTSs, involving an action code $\mathcal{R}$. Whenever the action code $\mathcal{R}$ is clear from the context, we omit the index and simply speak of operators α, ϱ, γ for the sake of brevity.

▶ **Definition 3.9** (Contraction, 🌿). *For each action code $\mathcal{R} \in \mathsf{Code}(A, B)$, the* contraction operator $\alpha_{\mathcal{R}} \colon \mathsf{LTS}(A) \to \mathsf{LTS}(B)$ *is defined as follows. For $\mathcal{M} \in \mathsf{LTS}(A)$, the LTS $\alpha_{\mathcal{R}}(\mathcal{M})$ has states $Q^{\alpha(\mathcal{M})} \subseteq Q^{\mathcal{M}}$ and transitions $\longrightarrow_{\alpha(\mathcal{M})}$ defined inductively by the rules (1_α) and (2_α), for all $q, q' \in Q^{\mathcal{M}}$, $b \in B$.*

$$\frac{}{q_0^{\mathcal{M}} \in Q^{\alpha(\mathcal{M})}} \;\; (1_\alpha) \qquad\qquad \frac{q \in Q^{\alpha(\mathcal{M})}, \quad b \in \mathsf{dom}(\mathcal{R}), \quad q \xRightarrow{\mathcal{R}(b)}_{\mathcal{M}} q'}{q \xrightarrow{\;b\;}_{\alpha(\mathcal{M})} q', \qquad q' \in Q^{\alpha(\mathcal{M})}} \;\; (2_\alpha)$$

The initial state $q_0^{\alpha(\mathcal{M})} := q_0^{\mathcal{M}}$ is the same as in $\mathcal{M}$.

(a) An action code $\mathcal{R}$ together with $\alpha_{\mathcal{R}}(\mathcal{M})$

(b) Another code $\mathcal{S}$ together with $\alpha_{\mathcal{S}}(\mathcal{M})$

Figure 5 The resulting contraction of the LTS $\mathcal{M}$ from Figure 2 for different action codes.

▶ **Example 3.10.** Figures 5 shows two examples of action codes and the contractions obtained when we apply them to the Mealy machine of Figure 2 (with the original machine shaded in the background). The examples illustrate that by choosing different codes we may obtain completely different abstractions of the same LTS.

The next proposition asserts that we can view $\alpha_{\mathcal{R}}$ as a monotone function $\alpha_{\mathcal{R}} \colon \mathsf{LTS}(A) \to \mathsf{LTS}(B)$ between preordered classes.

▶ **Proposition 3.11** (Monotonicity, 🐒). *For every action code $\mathcal{R} \in \mathsf{Code}(A, B)$, whenever $\mathcal{M} \sqsubseteq \mathcal{N}$ for $\mathcal{M}, \mathcal{N} \in \mathsf{LTS}(A)$, then $\alpha_{\mathcal{R}}(\mathcal{M}) \sqsubseteq \alpha_{\mathcal{R}}(\mathcal{N})$ in $\mathsf{LTS}(B)$.*
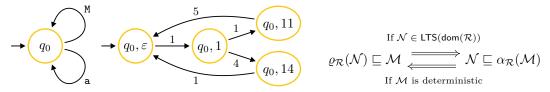
4 Refinements

Now that we have introduced the contraction $\alpha_{\mathcal{R}}$ of an LTS for a code $\mathcal{R}$, it is natural to consider an operation in the other direction, which we call the *refinement* $\varrho_{\mathcal{R}}$. Intuitively, refinement replaces each abstract transition $q \xrightarrow{b} q'$ by a sequence of concrete transitions, as prescribed by $\mathcal{R}$.

▶ **Definition 4.1** (Refinement, 🐒). *For each action code $\mathcal{R} \in \mathsf{Code}(A, B)$, we define the refinement operator $\varrho_{\mathcal{R}} \colon \mathsf{LTS}(B) \to \mathsf{LTS}(A)$ as follows. For $\mathcal{M} \in \mathsf{LTS}(B)$, the LTS $\varrho_{\mathcal{R}}(\mathcal{M}) \in \mathsf{LTS}(A)$ has a set of states*

$$Q^{\varrho(\mathcal{M})} := \{(q, w) \in Q^{\mathcal{M}} \times A^* \mid w = \varepsilon \text{ or (there is } b \text{ with } q \xrightarrow{b}_{\mathcal{M}} \text{ and } w \lneqq \mathcal{R}(b))\}$$

and the initial state $(q_0^{\mathcal{M}}, \varepsilon)$. The transition relation $\longrightarrow_{\varrho(\mathcal{M})}$ is defined by the following rules:

$$\frac{(q, wa) \in Q^{\varrho(\mathcal{M})}}{(q, w) \xrightarrow{a}_{\varrho(\mathcal{M})} (q, wa)} \ (1_{\varrho}) \qquad\qquad \frac{q \xrightarrow{b}_{\mathcal{M}} q' \quad wa = \mathcal{R}(b)}{(q, w) \xrightarrow{a}_{\varrho(\mathcal{M})} (q', \varepsilon)} \ (2_{\varrho})$$

Intuitively, whenever $\varrho(\mathcal{M})$ is in state (q, w), then this corresponds to being in state q in the abstract automaton $\mathcal{M} \in \mathsf{LTS}(B)$ and having observed the actions $w \in A^*$ so far. However, we have insufficiently many actions for finding an abstract transition $q \xrightarrow{b}_{\mathcal{M}} q'$ with $w = \mathcal{R}(b)$ because w is still to short. Nevertheless, whenever $\varrho(\mathcal{M})$ admits a transition to a state (q, w) with $w \neq \varepsilon$, then we know that we can eventually complete w to a sequence corresponding to an abstract transition: there exist at least one $q \xrightarrow{b}_{\mathcal{M}} q'$ for some $b \in \mathsf{dom}(\mathcal{R})$ with $w \leq \mathcal{R}(b)$. If the abstract system $\mathcal{M}$ is non-deterministic, then there may be multiple abstract transitions that match in the final rule (2_{ϱ}), but the transitions produced by rule (1_{ϱ}) are deterministic.

Figure 6 LTS and its refinement w.r.t $\mathcal{R}$ of Figure 3.

Figure 7 Theorem 4.5

▶ **Example 4.2.** Figure 6 shows an example application of a refinement operator that replaces the actions of the LTS $\mathcal{M}$ on the left by their ASCII encoding in octal format, as prescribed by the action code from Figure 3. The initial state is (q_0, ε), corresponding to q_0 in $\mathcal{M}$. Since $\mathcal{M}$ contains abstract labels M and a, with $\mathcal{R}(\mathtt{M}) = 1\,1\,5$ and $\mathcal{R}(\mathtt{a}) = 1\,4\,1$, we need to introduce additional states for having read 1, $1\,1$, and $1\,4$, because those are the sequences of A-actions before we have observed a sequence $\mathcal{R}(b) \in A^+$ for some $b \in B$.

A more visual explanation of $\varrho_{\mathcal{R}}(\mathcal{M})$ is the following: for every state $q \in Q^{\mathcal{M}}$, we consider the outgoing transitions $\{q \xrightarrow{b}_{\mathcal{M}} q' \mid b \in B, q' \in Q^{\mathcal{M}}\}$ and labels $B' \subseteq B$ that appear in it. Then, this outgoing-transition structure is replaced with (a copy of) the minimal subgraph of the tree $\mathcal{R}$ containing all leaves with labels in B'.

Like contraction, the refinement operation also preserves the simulation preorder.

▶ **Proposition 4.3** (Monotonicity, 🌳). *For all action codes $\mathcal{R} \in \mathsf{Code}(A, B)$, if $\mathcal{M} \sqsubseteq \mathcal{N}$ in $\mathsf{LTS}(B)$, then $\varrho_{\mathcal{R}}(\mathcal{M}) \sqsubseteq \varrho_{\mathcal{R}}(\mathcal{N})$ in $\mathsf{LTS}(A)$.*

As $\mathcal{R}$ is deterministic, applying $\varrho_{\mathcal{R}}$ on a deterministic LTS results in a deterministic LTS:

▶ **Proposition 4.4** (Refinement preserves determinism, 🌳). *For every action code $\mathcal{R} \in \mathsf{Code}(A, B)$, if $\mathcal{M} \in \mathsf{LTS}(B)$ is deterministic, then $\varrho_{\mathcal{R}}(\mathcal{M}) \in \mathsf{LTS}(A)$ is deterministic, too.*

▶ **Theorem 4.5** (Galois connection, 🌳). *For $\mathcal{R} \in \mathsf{Code}(A, B)$, $\mathcal{N} \in \mathsf{LTS}(B)$, and $\mathcal{M} \in \mathsf{LTS}(A)$:*

1. *If $\mathcal{N}$ is in the subclass $\mathsf{LTS}(\mathsf{dom}(\mathcal{R})) \subseteq \mathsf{LTS}(B)$, then $\varrho_{\mathcal{R}}(\mathcal{N}) \sqsubseteq \mathcal{M}$ implies $\mathcal{N} \sqsubseteq \alpha_{\mathcal{R}}(\mathcal{M})$.*
2. *If $\mathcal{M}$ is deterministic, then $\mathcal{N} \sqsubseteq \alpha_{\mathcal{R}}(\mathcal{M})$ implies $\varrho_{\mathcal{R}}(\mathcal{N}) \sqsubseteq \mathcal{M}$.*

The condition in the first direction means that $\mathcal{N} \in \mathsf{LTS}(B)$ only makes use of action labels in the subset $\mathsf{dom}(\mathcal{R}) \subseteq B$. Hence, in the proof, we can consider $\mathcal{R}$ to be a total map $\mathsf{dom}(\mathcal{R}) \to A^+$.

▶ Remark 4.6. If we wanted to support non-deterministic $\mathcal{M}$, we can consider a less-pleasant $\varrho'_{\mathcal{R}}$ that replaces every $q \xrightarrow{b} q'$ for $\mathcal{R}(a_1 \cdots a_n) = b$ with literally a sequence $q \xrightarrow{a_1} \cdots \xrightarrow{a_n} q'$. Thus, $\varrho'_{\mathcal{R}}$ would rather create a system as in Figure 1b whereas $\varrho_{\mathcal{R}}$ creates a system as in Figure 1c. However, such an operator $\varrho'_{\mathcal{R}}$ does not preserve determinism.

▶ Remark 4.7. In the proof of the Galois connection, we make use of the fact that our action codes are functional, i.e. that every $b \in B$ is encoded by at most one $w \in A^*$. We would allow multiple, then one can show that α can not have a left-adjoint (details in appendix).

5 Concretizations

In this section, we consider another method of transforming an abstract system into a concrete one: the *concretization* operator. Whereas refinement is the left adjoint of contraction

(Theorem 4.5), this section will establish that concretization is the right adjoint (Theorem 5.5) of contraction. Whereas for refinement we omitted transitions for which the action code $\mathcal{R}$ was not defined, for concretization we add transitions to a new *chaos* state [18] in which any action may occur. Essentially, this is the idea of *demonic completion* of [6]. In order to reduce the number of transitions to the chaos state, the concretization operator is parametric in a reflexive relation $\mathcal{I} \subseteq A \times A$ which describes whether two symbols are sufficiently similar. With this relation, we allow transitions to the chaos state only for those symbols that are not similar to any symbol for which the code is defined:

▶ **Definition 5.1** (Concretization, 🐑). *Let* $\mathcal{M} \in \mathsf{LTS}(B)$ *be an LTS,* $\mathcal{R} \in \mathsf{Code}(A, B)$ *an action code, and* $\mathcal{I} \subseteq A \times A$ *a reflexive relation. The* concretization $\gamma_{\mathcal{R},\mathcal{I}}(\mathcal{M}) \in \mathsf{LTS}(A)$ *consists of:*

- $Q^{\gamma(\mathcal{M})} := Q^{\mathcal{M}} \times N \cup \{\chi\}$ *with* $N := \{w \in A^* \mid w = \varepsilon \text{ or } \exists b \in \mathsf{dom}(\mathcal{R}) : w \lneqq \mathcal{R}(b)\}$.
- $q_0^{\gamma(\mathcal{M})} := (q_0^{\mathcal{M}}, \varepsilon)$
- *Transitions are defined by the following rules:*

$$\frac{wa \in N}{(q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} (q, wa)} \ (1_\gamma) \qquad \frac{q \xrightarrow{b}_{\mathcal{M}} q', \ \mathcal{R}(b) = wa}{(q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} (q', \varepsilon)} \ (2_\gamma)$$

$$\frac{\forall a' \in A, (a, a') \in \mathcal{I} : \ wa' \notin N \wedge wa' \notin \mathsf{im}(\mathcal{R})}{(q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} \chi} \ (3_\gamma) \qquad \frac{}{\chi \xrightarrow{a}_{\gamma(\mathcal{M})} \chi} \ (4_\gamma)$$

Intuitively, N represent the internal nodes of the tree-representation of action code $\mathcal{R}$. The transitions then try to accumulate a word $w \in A^*$ known to the action code (rule (1_γ)). As soon as we reach $w = \mathcal{R}(b)$ for some b, we use a b-transition in the original $\mathcal{M} \in \mathsf{LTS}(B)$ to jump to a new state (rule (2_γ)). The chaos state χ attracts all runs with symbols unknown to the action code. The corresponding rule (3_γ) involves the relation $\mathcal{I} \subseteq A \times A$. The rule only allows a transition to χ for a symbol $a \in A$ if there is no related symbol $a' \in A$, $(a, a') \in A$ for which the code R could make a transition. For general LTSs, we can simply consider $\mathcal{I}$ to be the identity relation on A. Once transitioned to the chaos state χ, we allow transitions for arbitrary action symbols $a \in A$ (rule (4_γ)).
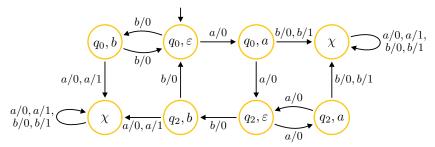
▶ **Example 5.2.** For the special case of Mealy machines $A := I \times O$, we can define $\mathcal{I} \subseteq (I \times O) \times (I \times O)$ to relate (i, o) and (i', o') iff $i = i'$, i.e. two actions are related if they use the same input symbol. Then, we only have transitions to the chaos states if the code can't do any action for the same input symbol $i \in I$. Figure 8 depicts the concretization (for this $\mathcal{I}$) of the Mealy machine of Figure 5a(right) with the action code of Figure 5a(left). To increase readability, we introduced two copies of chaos state χ. Also, multiple labels next to an arrow denote multiple transitions.

Like in the refinement operator, the transition structure of γ is built in such a way that transitions for $b \in B$ in $\mathcal{M}$ correspond to runs of $\mathcal{R}(b)$ in $\gamma(\mathcal{M})$:

$$(q, \varepsilon) \xRightarrow{\mathcal{R}(b)}_{\gamma(\mathcal{M})} \bar{q} \quad \text{iff} \quad \exists q' : q \xrightarrow{b}_{\mathcal{M}} q' \text{ and } \bar{q} = (q', \varepsilon).$$

To make γ right adjoint to α, all runs outside the code $\mathcal{R}$ lead to the chaos state. One may think that the many transitions to the chaos state χ would make the construction $\gamma_{\mathcal{R}}$ trivial. However, only those paths lead to χ for which the action code is not defined.

The following technical condition describes that a code $\mathcal{R}$ contains sufficiently many related symbols compared to a given $\mathcal{M} \in \mathsf{LTS}(A)$:

Figure 8 Concretization of the Mealy machine of Figure 5a.

▶ **Definition 5.3** (🐸). *A code $\mathcal{R} \in \mathsf{Code}(A, B)$ is called $\mathcal{I}$-complete for $\mathcal{M} \in \mathsf{LTS}(A)$, if for all $w \in B^*$, $u \in A^*$, $q \in Q^{\mathcal{M}}$, $a, a' \in A$:*

$$r_0 \xrightarrow{u\,a}_{\mathcal{R}} \quad and \quad (a, a') \in \mathcal{I} \quad and \quad q_0 \xrightarrow{\mathcal{R}^* w\,u}_{\mathcal{M}} q \xrightarrow{a'} \quad implies \quad r_0 \xrightarrow{u\,a'}_{\mathcal{R}}.$$

Intuitively, $\mathcal{I}$-completeness means that if a state $q \in \mathcal{M}$ can do a transition for $a' \in A$ which is related to similar symbol $a \in A$ defined in the action code, then $a' \in A$ itself is also defined in the action code. However, we do not compare arbitrary transitions of q in $\mathcal{M}$ with arbitrary symbols mentioned in $\mathcal{R}$, but only look at the node in $\mathcal{R}$ reached when 'executing $\mathcal{R}$' zero or more times while following the path $q_0 \Longrightarrow q$.

For example, if $\mathcal{I} \subseteq A \times A$ happens to be the identity relation, then $\mathcal{R}$ is $\mathcal{I}$-complete for any $\mathcal{M} \in \mathsf{LTS}(A)$.

▶ **Assumption 5.4.** For the rest of the present Section 5, we fix the sets A, B, an action code $\mathcal{R} \in \mathsf{Code}(A, B)$, and a reflexive relation $\mathcal{I} \subseteq A \times A$.

▶ **Theorem 5.5** (Galois connection, 🐸). *For all $\mathcal{N} \in \mathsf{LTS}(A)$, and $\mathcal{M} \in \mathsf{LTS}(B)$, such that $\mathcal{R}$ is $\mathcal{I}$-complete for $\mathcal{N}$, we have*

$$\alpha_{\mathcal{R}}(\mathcal{N}) \sqsubseteq \mathcal{M} \ (in \ \mathsf{LTS}(B)) \qquad \Longleftrightarrow \qquad \mathcal{N} \sqsubseteq \gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M}) \ (in \ \mathsf{LTS}(A)).$$

▶ **Example 5.6** (🐸). If we instantiate $\mathcal{I}$ to be the identity relation Δ on A, then this means that we simply replace a' with a in rule (3_γ), and then we have above equivalence for all $\mathcal{N} \in \mathsf{LTS}(A)$ and $\mathcal{M} \in \mathsf{LTS}(B)$ (without any side-condition):

$$\alpha_{\mathcal{R}}(\mathcal{N}) \sqsubseteq \mathcal{M} \ (in \ \mathsf{LTS}(B)) \qquad \Longleftrightarrow \qquad \mathcal{N} \sqsubseteq \gamma_{\mathcal{R}, \Delta}(\mathcal{M}) \ (in \ \mathsf{LTS}(A)).$$

▶ **Example 5.7** (🐸). Consider the instantiation of $\mathcal{I}$ for Mealy machines described in Example 5.2. Let $\mathcal{N}$ be our running example of Figure 2, let $\mathcal{R}$ be the action code from Figure 5a(left), and let $\mathcal{M}$ be the abstract Mealy machine from Figure 5a(right), i.e. $\alpha_{\mathcal{R}}(\mathcal{N}) = \mathcal{M}$. One can verify that $\mathcal{R}$ is $\mathcal{I}$-complete for $\mathcal{N}$. Therefore, application of the Galois connection gives that there is a simulation from $\mathcal{N}$ to the Mealy machine $\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M})$ of Figure 8.

It is a standard proof that the operators in a Galois connections are monotone. In that proof, one applies the Galois connection also to $\mathcal{M} := \gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{N})$, so we first need to show that it satisfies the technical completeness condition:

▶ **Lemma 5.8** (🐸). *$\mathcal{R}$ is always $\mathcal{I}$-complete for $\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M})$.*

▶ **Corollary 5.9** (🐸). *$\mathcal{M} \sqsubseteq \mathcal{N}$ in $\mathsf{LTS}(B)$ implies $\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M}) \sqsubseteq \gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{N})$ in $\mathsf{LTS}(A)$.*

▶ Remark 5.10. Monotonicity of concretization also follows by observing that the rules in Definition 5.1 all fit the *tyft* format of [17] if we view $(\cdot, w)$ as a unary operator for each sequence $w \in N$. Monotonicity then follows from the result of [17] that the simulation preorder is a congruence for any operator defined using the *tyft* format. Since contraction also can be defined using the *tyft* format, also monotonicity of contraction (Proposition 3.11) follows from the result of [17].

Like refinement, concretization preserves determinism.

▶ **Proposition 5.11** (🐑). *If $\mathcal{M} \in \mathsf{LTS}(B)$ is a deterministic LTS and Δ the identity relation on A, then $\gamma_{\mathcal{R},\Delta}(\mathcal{M})$ is deterministic, too.*

If the code $\mathcal{R} \in \mathsf{Code}(A, B)$ is defined for all labels mentioned in $\mathcal{M} \in \mathsf{LTS}(B)$, then $\gamma_{\mathcal{R}}$ is even the right inverse of $\alpha_{\mathcal{R}}$, that is, we have a Galois insertion:

▶ **Theorem 5.12** (Galois insertion, 🐑). *If $\mathcal{M} \in \mathsf{LTS}(\mathsf{dom}(\mathcal{R}))$, then $\mathcal{M} \cong \alpha_{\mathcal{R}}(\gamma_{\mathcal{R},\mathcal{I}}(\mathcal{M}))$.*

Note that $\mathsf{dom}(\mathcal{R}) \subseteq B$, and so $\mathsf{LTS}(\mathsf{dom}(\mathcal{R})) \subseteq \mathsf{LTS}(B)$. Since we may reach the chaos state χ in the concretization, it is clear that $\gamma_{\mathcal{R}}$ is not a left inverse of $\alpha_{\mathcal{R}}$ in general.

6   Action Code Composition

Since notions of abstraction can be stacked up, it is natural to consider multiple adaptors for multiple action codes. Assume an action code $\mathcal{R} \in \mathsf{Code}(A, B)$ and an action code $\mathcal{S} \in \mathsf{Code}(B, C)$. Then the composition of $\mathcal{R}$ and $\mathcal{S}$ should be an action code from A to C.

▶ **Definition 6.1** (🐑). *Given two map-based action codes $\mathcal{R} \colon B \rightharpoonup A^+$ and $\mathcal{S} \colon C \rightharpoonup B^+$, we define their* (Kleisli) composition $(\mathcal{R} * \mathcal{S}) \colon C \rightharpoonup A^+$ *by*

$$(\mathcal{R} * \mathcal{S})(c) = \begin{cases} \mathcal{R}(b_1) \cdots \mathcal{R}(b_n) & \textit{if } \mathcal{S}(c) = b_1 \cdots b_n \textit{ with } \forall i \colon b_i \in \mathsf{dom}(\mathcal{R}) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The composed action code $\mathcal{R} * \mathcal{S}$ is only defined for $c \in C$ if $\mathcal{S}$ is defined for c and additionally $\mathcal{R}$ is defined for every letter $b_i \in B$ that appears in the word $\mathcal{S}(c) \in B^+$.

▶ Remark 6.2. The defined composition is an instance of *Kleisli composition* for a monad, which is a standard concept in functional programming and category theory.

▶ **Lemma 6.3** (🐑). *Action codes are closed under composition.*

Concretely, given two map-based action codes $\mathcal{R} \colon B \rightharpoonup A^+$ and $\mathcal{S} \colon C \rightharpoonup B^+$, their Kleisli composition $(\mathcal{R} * \mathcal{S}) \colon C \rightharpoonup A^+$ is again a prefix-free *partial map.*

Now that we can compose action codes, we can now investigate how the previously defined operators on LTSs behave for composed action codes:

▶ **Theorem 6.4** (🐑, 🐑). *Contraction and refinement commute with action code composition: for action codes $\mathcal{R} \in \mathsf{Code}(A, B)$, $\mathcal{S} \in \mathsf{Code}(B, C)$,*

1. $\alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M}) = \alpha_{\mathcal{S}}(\alpha_{\mathcal{R}}(\mathcal{M}))$ *for all $\mathcal{M} \in \mathsf{LTS}(A)$.*
2. $\varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M}) = \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$, *whenever $\mathsf{im}(\mathcal{S}) \subseteq \mathsf{dom}(\mathcal{R})^+$ and for all $\mathcal{M} \in \mathsf{LTS}(C)$.*

For the case of refinement, the additional assumption expresses that every word produced by $\mathcal{S}$ only contains letters $b \in B$ for which $\mathcal{R}$ is defined. The equations of Theorem 6.4 equivalently mean that the following diagrams commute:

$$\begin{array}{ccc} \mathsf{LTS}(A) \xrightarrow{\ \alpha_{\mathcal{R}*\mathcal{S}}\ } \mathsf{LTS}(C) & \qquad & \mathsf{LTS}(A) \xleftarrow{\ \varrho_{\mathcal{R}*\mathcal{S}}\ } \mathsf{LTS}(C) \\ \searrow_{\alpha_{\mathcal{R}}} \quad \nearrow_{\alpha_{\mathcal{S}}} & & \nwarrow_{\varrho_{\mathcal{R}}} \quad \swarrow_{\varrho_{\mathcal{S}}} \\ \mathsf{LTS}(B) & & \mathsf{LTS}(B) \end{array}$$

▶ Remark 6.5 (🦋). Concretization does not commute with action code composition. The reason for that is that the rules (1_γ) and (2_γ) in $\gamma_\mathcal{R}(\gamma_\mathcal{S}(\mathcal{M}))$ would also be applied to transitions for the chaos state in $\gamma_\mathcal{S}(\mathcal{M}) \in \mathsf{LTS}(B)$ (see appendix for details).

7 Adaptors

In this section, we describe how action codes may be used for learning and testing of black-box systems. The general architecture is shown in Figure 9. On the right we see the *system under test (SUT)*, some piece of hardware/software whose behavior can be modeled by a Mealy machine $\mathcal{M}$ with inputs I and outputs O. On the left we see the *learner/tester*, an

Figure 9 Using action codes for learning/testing.

Figure 10 Example 7.4

agent which either tries to construct a model $\mathcal{N}$ of $\mathcal{M}$ by performing experiments, or already has such a model $\mathcal{N}$ and performs experiments (tests) to find a counterexample which shows that $\mathcal{M}$ and $\mathcal{N}$ behave differently. The learner/tester uses abstract inputs X and outputs Y. In between the learner/tester and the SUT we place an *adaptor*, which uses action code $\mathcal{R}$ to translate between the abstract world of the learner/tester and the concrete world of the SUT. In order to enable the adaptor to do its job, we need to make four (reasonable) assumptions.

Our first assumption, common in model-based testing [33], is that the SUT will accept any input from I in any state, that is, we require that $\mathcal{M}$ is *input enabled*: for all $q \in Q^\mathcal{M}$ and $i \in I$, $q \xrightarrow{i/}_\mathcal{M}$. Our second assumption is that code $\mathcal{R}$ is $\mathcal{I}$-complete for $\mathcal{M}$. This ensures that whenever the adaptor sends a concrete symbol $i \in I$ to the SUT, it will accept any output $o \in O$ that the SUT may possibly produce in response. Our third assumption is that whenever the adaptor receives an abstract input $x \in X$ from the learner/tester, it can choose concrete inputs from I that drive $\mathcal{R}$ from its initial state to a leaf with label (x, y), for some $y \in Y$. Output y can then be returned as a response to the learner/tester. Reaching such a leaf is nontrivial since the transitions taken in $\mathcal{R}$ are also determined by the outputs provided by the SUT. We may think of the situation in terms of a 2-player game where the adaptor wins if the game ends in an x-leaf, and the SUT wins otherwise. Formally, we require that $\mathcal{R}$ has finitely many states and a winning strategy for every input $x \in X$, as defined below:

▶ **Definition 7.1** (Winning). *Let* $\mathcal{R} = \langle R, r_0, \longrightarrow, l \rangle \in \mathsf{Code}(I \times O, X \times Y)$ *be an action code with R finite and let $x \in X$. Then*

1. *A leaf $r \in R$ is* winning *for x if $\pi_1(l(r)) = x$.*[1]
2. *An internal state $r \in R$ is* winning *for x with input $i \in I$ if $r \xrightarrow{i/}$ and, for each transition of the form $r \xrightarrow{i/o} r'$, r' is winning for x.*
3. *An internal state $r \in R$ is* winning *for x if it is winning for x with some $i \in I$.*

---

[1] We use projections functions π_1 and π_2 to denote the first and second element of a pair, respectively. So $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$.

4. $\mathcal{R}$ *has a* winning strategy for x *if r_0 is winning for x.*

▶ **Example 7.2.** The action codes for Mealy machines that we have seen thus far (Figures 4, 5a and 5b) are winning for all the inputs that label their leafs. The action code of Figure 4 is not winning for the input ⬕ (latte macchiato), for the simple reason that this input does not label any leaf. If we remove the transition to the leaf ⬔/2 in Figure 4, then the resulting code is no longer winning for ⬔ (espresso), although it is winning for ⬔ (coffee).

Our fourth and final assumption is that action code $\mathcal{R}$ is *determinate*. If an action code is determinate then, for each state r and abstract input x, there is at most one concrete input i such that r is winning for x with i.

▶ **Definition 7.3** (Determinate, 🐑). *An action code $\mathcal{R}$ is* determinate *if, for each state r, whenever $r \xrightarrow{i_1/} r_1$, $r \xrightarrow{i_2/} r_2$ and from both r_1 and r_2 there is a path to a leaf labeled with input x, then $i_1 = i_2$.*

▶ **Example 7.4.** All action codes for Mealy machines that we have seen thus far (Figures 4, 5a and 5b) are determinate. Figure 10 shows an action code that is not determinate: in the root two different concrete inputs a and b are enabled that lead to leafs with the same abstract input **0**. Hence (trivially), this action code does have a winning strategy for input **0**.

Algorithm 1 shows pseudocode for an adaptor that implements action code $\mathcal{R}$. During learning/testing, the adaptor records the current state of the action code in a variable r. When an abstract input x arrives, it first sets r to r_0. As long as current state r is internal, the adaptor chooses an input i that is winning for x, and forwards it to the SUT. When the SUT replies with an output o, the adaptor sets r to a state r' with $r \xrightarrow{i/o} r'$. When the new r is internal the adaptor chooses again a winning input, and updates its current state after interacting with the SUT, etc. When the new r is a leaf with label (x, y) then the adaptor returns symbol y to the learner/tester and waits for the next abstract input to arrive.

🟨 **Algorithm 1** Pseudocode for an adaptor that implements action code $\mathcal{R}$.

```
 1: while true do
 2:     x ← Receive-from-learner()
 3:     r ← r₀
 4:     while r is internal do                    ▷ loop invariant: r is winning for x
 5:         i ← unique input such that r is winning for x with i
 6:         Send-to-SUT(i)
 7:         o ← Receive-from-SUT()
 8:         r ← unique state r' such that r --i/o--> r'    ▷ R is I-complete for M
 9:     end while
10:     Send-to-learner(π₂(l(r)))
11: end while
```

From the perspective of the learner/tester, the combination of the adaptor and SUT behaves the same as the contraction $\alpha_{\mathcal{R}}(\mathcal{M})$. In the appendix, we will formalize this statement by modeling both the combination of adaptor and SUT, as well as contraction $\alpha_{\mathcal{R}}(\mathcal{M})$ as expressions in the process calculus CCS [28], and then establish the existence of *delay simulations* between these expressions. This implies that both expressions have the same traces if we remove all occurrences of the synchronizations between adaptor and SUT, which are invisible from the perspective of the learner.

▶ **Theorem 7.5.** *Let $\mathcal{M} \in$ LTS$(I \times O)$ be an input enabled Mealy machine and let $\mathcal{R} \in$ Code$(I \times O, X \times Y)$ be a finite, determinate action code that has a winning strategy for every input in X and that is output enabled for $\mathcal{M}$. Then the composition of an implementation for $\mathcal{M}$ and an adaptor for $\mathcal{R}$ is delay simulation equivalent to an implementation for $\alpha_\mathcal{R}(\mathcal{M})$.*

Active automata learning algorithms and tools for Mealy machines typically assume that the system under learning is *output deterministic*[2]: the output and target state of a transition are uniquely determined by its source state and input.

▶ **Definition 7.6.** *Mealy machine $\mathcal{M}$ is* output deterministic *if, for each state q and input i,*

$$q \xrightarrow{i/o} r \wedge q \xrightarrow{i/o'} r' \quad \Rightarrow \quad o = o' \wedge r = r'.$$

For action codes that are determinate, contraction preserves output determinism. This property makes it possible to use existing automata learning tools to learn models of an output deterministic SUT composed with a determinate adaptor.

▶ **Proposition 7.7** (🦋). *Suppose $\mathcal{M}$ is a Mealy machine and $\mathcal{R}$ is an action code. If $\mathcal{M}$ is output deterministic and $\mathcal{R}$ is determinate then $\alpha_\mathcal{R}(\mathcal{M})$ is output deterministic.*

8 Discussion and Future Work

Via the notion of action codes, we provided a new perspective on the fundamental question how high-level state machine models with abstract actions can be related to low-level models in which these actions are refined by sequences of concrete actions. This perspective may, for instance, help with the systematic design of adaptors during learning and testing, and the subsequent interpretation of obtained results. Our theory allows for action codes (such as in Figure 4) that are adaptive in the sense that outputs which occur in response to inputs at the concrete level may determine the sequence of concrete inputs that refines an abstract input. We are not aware of case studies in which such adaptive codes are used, but believe they may be of practical interest. One may, for instance, consider a scenario in which an abstract action AUTHENTICATE is refined by a protocol in which a user is either asked to authenticate by entering a PIN code, or by providing a fingerprint.

Close to our work are the results of Rensink and Gorrieri [29], who investigate vertical implementation relations to link models at conceptually different levels of abstraction. These relations are indexed by a refinement function that maps abstract actions into concrete processes. Within a setting of a CCS-like language, Rensink & Gorrieri [29] list a number of proof rules that should hold for any vertical implementation relation, and propose *vertical bisimulation* as a candidate vertical implementation relation for which these proof rules hold. In the setting of our paper, we can define two vertical implementation relations $\sqsubseteq^\mathcal{R}_\gamma$ and $\sqsubseteq^\mathcal{R}_\varrho$, for any action code $\mathcal{R}$, by

$$\mathcal{M} \sqsubseteq^\mathcal{R}_\gamma \mathcal{N} \;\Leftrightarrow\; \mathcal{M} \sqsubseteq \gamma_\mathcal{R}(\mathcal{N}) \quad \text{and} \quad \mathcal{M} \sqsubseteq^\mathcal{R}_\varrho \mathcal{N} \;\Leftrightarrow\; \mathcal{M} \sqsubseteq \varrho_\mathcal{R}(\mathcal{N}).$$

Then $\sqsubseteq^\mathcal{R}_\varrho \subseteq \sqsubseteq^\mathcal{R}_\gamma$ and both relations satisfy all language-independent proof rules of [29]. For instance, we have

$$\frac{\mathcal{M} \sqsubseteq \mathcal{M}' \quad \mathcal{M}' \sqsubseteq^\mathcal{R}_\gamma \mathcal{N}' \quad \mathcal{N}' \sqsubseteq \mathcal{N}}{\mathcal{M} \sqsubseteq^\mathcal{R}_\gamma \mathcal{N}}$$

---

[2] The notion of deterministic that we use in this article is the standard one for LTSs. In the literature on Mealy machines and FSMs, machines that we call output deterministic are called deterministic, and machines that we call deterministic are called observable.

(since $\gamma_{\mathcal{R}}$ is monotone and $\sqsubseteq$ is transitive).With the action code $\mathcal{R}$ of Figure 3, both implementation relations relate the LTSs of Figures 1c and 1a. However, the vertical bisimulation preorder of Rensink and Gorrieri [29] does not relate these LTSs, when using a code that maps a to $1\,4\,1$, and b to $1\,4\,2$. This suggests that bisimulations may not be suitable as vertical implementation relations.

Also close to our work are results of Burton et al [8, 21], who propose a vertical implementation relation in the context of CSP. Instead of action codes, they use *extraction patterns*, a strict monotonic map $extr : Dom \to B^*$, where Dom is the prefix closure of a set $dom \subseteq A^*$ of concrete action sequences that may be regarded as complete. As a mapping from concrete to abstract sequences of actions, extraction patterns are more general than action codes. However, as extraction mappings are not required to have an inverse, establishing interesting Galois connections in this setting may be difficult. With an extraction pattern defined in the obvious way, the LTSs of Figures 1c and 1a are related by the implementation relation of [8]. We are not aware of any other vertical implementation relation proposed in the literature that handles our basic interface refinement example correctly. We find it surprising that the fundamental problem of refining inputs actions has not been properly addressed in the literature, except in some work that apparently has not been picked up outside Newcastle-upon-Tyne and Catania.

Our theory is orthogonal to the one of Aarts et al [1], which explores the use of so-called *mappers* to formalize adaptors that abstract the large action alphabets of realistic applications into small sets of actions that can be handled by a learning tool. Aarts et al [1] also describe the relation between abstract and concrete models using a Galois connection. In practical applications of model learning, it makes sense to construct an adaptor that combines a mapper in the sense of [1] with an action code as introduced in this paper. Fiterău-Broştean et al [11] describe a small domain specific language to specify mapper components, and from which adaptor software can be generated automatically. It would be interesting to extend this domain specific language so that it may also be used to specify action codes.

We developed our theory for LTSs and Mealy machines, using the simulation preorder as the implementation relation. It would be interesting to transfer our results to other modeling frameworks, such as IOTSs [33] timed automata [3] and Markov Decision Processes, and to other preorders and equivalences in the linear-time branching-time spectrum for LTSs [15] and IOTSs [20]. An obvious direction for future work would be to explore how action codes interact with parallel composition. Here the work of [8, 21] may serve as a basis.

Different action codes lead to different contractions, and thereby to different abstract views of a system, see for instance Figures 5a and 5b. We may try to exploit this fact during learning and testing. For instance, if a system $\mathcal{M}$ is too big for state-of-the-art learning algorithms, we may still succeed to learn partial views using cleverly selected action codes. Using our Galois connections we then could obtain various upper and lower bounds for $\mathcal{M}$. Ideally, such an approach may even succeed to uniquely identify $\mathcal{M}$. Maarse [27] quantified the quality of a contraction $\alpha_{\mathcal{R}}(\mathcal{M})$ in terms of the graph-theoretic concept of *eccentricity*. If q and q' are states in an LTS $\mathcal{M}$ then $d(q, q')$ is defined as the number of transitions in the shortest path from q to q' (or ∞ if no such path exists). For any set of states $Q \subseteq Q_{\mathcal{M}}$, the *eccentricity* $\varepsilon(Q)$ is defined as $\max_{q' \in Q_{\mathcal{M}}} \min_{q \in Q} d(q, q')$, that is, the maximal distance one needs to travel to visit a state of $\mathcal{M}$, starting from a state of Q. A good contraction has a small set of states Q and a low eccentricity $\varepsilon(Q)$: it only covers a small subset Q of the states of $\mathcal{M}$, but any state from $\mathcal{M}$ can be reached via a few transitions from a Q-state.

References

1   F. Aarts, B. Jonsson, J. Uijen, and F.W. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *Formal Methods in System Design*, 46(1):1–41, 2015.

2   M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 1(15):73–132, 1993.

3   G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Third International Conference on the Quantitative Evaluation of SysTems (QEST 2006),* 11-14 September 2006, Riverside, CA, USA, pages 125–126. IEEE Computer Society, 2006.

4   J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra.* North-Holland, 2001.

5   J. Berstel and D. Perrin. *Theory of codes.* Academic Press, 1985.

6   M. van der Bijl, A. Rensink, and J. Tretmans. Compositional testing with ioco. In A. Petrenko and A. Ulrich, editors, *Formal Approaches to Software Testing, Third International Workshop on Formal Approaches to Testing of Software, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003*, volume 2931 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2003.

7   M. van der Bijl, A. Rensink, and J. Tretmans. Action refinement in conformance testing. In F. Khendek and R. Dssouli, editors, *Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference, TestCom 2005, Montreal, Canada, May 31 - June 2, 2005, Proceedings*, volume 3502 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2005.

8   J. Burton, M. Koutny, and G. Pappalardo. Implementing communicating processes in the event of interface difference. In *2nd International Conference on Application of Concurrency to System Design (ACSD 2001), 25-30 June 2001, Newcastle upon Tyne, UK*, page 87. IEEE Computer Society, 2001.

9   G. Chalupar, S. Peherstorfer, E. Poll, and J. de Ruiter. Automated reverse engineering using Lego. In *Proceedings 8th USENIX Workshop on Offensive Technologies (WOOT'14),* San Diego, California, Los Alamitos, CA, USA, August 2014. IEEE Computer Society.

10   E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, Cambridge, Massachusetts, 1999.

11   P. Fiterău-Broştean, R. Janssen, and F.W. Vaandrager. Combining model learning and model checking to analyze TCP implementations. In S. Chaudhuri and A. Farzan, editors, *Proceedings 28th International Conference on Computer Aided Verification (CAV'16),* Toronto, Ontario, Canada, volume 9780 of *Lecture Notes in Computer Science*, pages 454–471. Springer, 2016.

12   P. Fiterău-Broştean, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky. Analysis of DTLS implementations using protocol state fuzzing. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2523–2540. USENIX Association, August 2020.

13   P. Fiterău-Broştean, T. Lenaerts, E. Poll, J. de Ruiter, F. Vaandrager, and P. Verleg. Model learning and model checking of SSH implementations. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, SPIN 2017, pages 142–151, New York, NY, USA, 2017. ACM.

14   H. Garavel and F. Lang. Equivalence checking 40 years after: A review of bisimulation tools. In N. Jansen, M. Stoelinga, and P. van den Bos, editors, *A Journey from Process Algebra via Timed Automata to Model Learning*, pages 213–265, Cham, 2022. Springer Nature Switzerland.

15   R.J. van Glabbeek. The linear time — branching time spectrum I. The semantics of concrete, sequential processes. In Bergstra et al. [4], pages 3–99.

16   R. Gorrieri and A. Rensink. Action refinement. In Bergstra et al. [4], pages 1047–1147.

17   J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

18   C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

19   J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

20   R. Janssen, F.W. Vaandrager, and J. Tretmans. Relating alternating relations for conformance and refinement. In W. Ahrendt and S. Lizeth Tapia Tarifa, editors, *Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings*, volume 11918 of *Lecture Notes in Computer Science*, pages 246–264. Springer, 2019.

21   M. Koutny and G. Pappalardo. The ERT model of fault-tolerant computing and its application to a formalisation of coordinated atomic actions. Report 636, Department of Computing Science, University of Newcastle upon Tyne, 1998.

22   L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

23   D. Lee and M. Yannakakis. Principles and methods of testing finite state machines — a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

24   N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1996.

25   N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

26   N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

27   T. Maarse. *Active Mealy Machine Learning Using Action Refinements.* Master's thesis, Radboud University, Institute for Computing and Information Sciences, August 2020.

28   R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

29   A. Rensink and R. Gorrieri. Vertical implementation. *Information and Computation*, 170(1):95–133, 2001.

30   J. de Ruiter and E. Poll. Protocol state fuzzing of TLS implementations. In *24th USENIX Security Symposium*, pages 193–206, Washington, D.C., August 2015. USENIX Association.

31   C. McMahon Stone, T. Chothia, and J. de Ruiter. Extending automated protocol state learning for the 802.11 4-way handshake. In J. López, J. Zhou, and M. Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 325–345. Springer, 2018.

32   A.S. Tanenbaum and D. Wetherall. *Computer networks, 5th Edition.* Pearson, 2011.

33   J. Tretmans. Test generation with inputs, outputs, and repetitive quiescence. *Software–Concepts and Tools*, 17:103–120, 1996.

34   J. Tretmans. Model based testing with labelled transition systems. In R.M. Hierons, J.P. Bowen, and M. Harman, editors, *Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2008.

35   F.W. Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, February 2017.

36   P. Verleg. *Inferring SSH state machines using protocol state fuzzing.* Master thesis, Radboud University Nijmegen, February 2016.

37   M. Yannakakis. Hierarchical state machines. In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, pages 315–330, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

A Formalization in Coq

The full Coq formalization is available in the ancillary files on arxiv:

> `https://arxiv.org/src/2301.00199/anc`

This repository contains both the plain Coq source files and generated HTML documentation for easier reading and navigation (clickable links, etc). We use the Coq version 8.16.1. In order to compile the source files, run

```
coq_makefile -o Makefile *.v
make
```

in the source directory. The HTML documentation can be read *without* compilation.

For each definition or result that was formalized in Coq, the following list provides the identifier of the corresponding object in Coq. In some places the formalization slightly differs from what we use in the paper; if this is the case, we also provide explanations here:

Definition 2.1 $\triangleq$ `LTS` (in `LTS.v`).

Example 2.3 $\triangleq$ `ExampleMealy` (in `MealyExample.v`).

Definition 2.4 $\triangleq$ `deterministic` (in `LTS.v`) – In Coq, we have the notion of determinism parametric in a relation $\mathcal{I} \subseteq A \times A$ such that we can uniformly treat determinism in the sense of LTSs and determinism in the sense of Mealy machines (determinism in the *input*-component of the action labels). This relation will be described in Section 5. The other notions (tree-shaped, leaf, grounded) are formalized in `TreeShapedCode.v`.

Definition 2.5 $\triangleq$ `traces` (in `LTS.v`).

Definition 2.6 $\triangleq$ `Simulation` (in `LTS.v`).

Lemma 2.7 $\triangleq$ `simulation_iff_trace_inclusion_for_deterministic` (in `LTS.v`).

Definition 2.8 $\triangleq$ `Isomorphism` (in `LTS.v`) – In Coq, we implicitly restrict to reachable states by only requiring f to be a left- and right-unique relation $f \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$, but without any requirement that f is total. Together with the other properties of f, this relation restricts to an isomorphism between the reachable states.

Definition 3.1 $\triangleq$ `TreeShapedCode` (in `TreeShapedCode.v`).

Definition 3.4 $\triangleq$ `Code` (in `CodeMap.v`) – In the formlization, the type of a code map is $f \colon B \rightharpoonup A^*$ (instead of $B \rightharpoonup A^+$) and then in addition to prefix-freeness, we require this map never returns the empty word. The reason for this decision is that A^* is just lists over A – a standard data type with good standard library support.

Lemma 3.5 $\triangleq$ `EveryTreeShapedCodeInducesSomeMapBasedCode` (in `TreeShapedCode.v`) – Being in a constructive setting, we have an explicit assumption that decides for each $b \in B$, whether there exists a leaf with label b. Also, the uniqueness of the map-based code is stated in a separate Lemma `TreeDefinesMapUniquely`, to avoid the use of the functional extensionality axiom.

Lemma 3.6 $\triangleq$ `MapToTree` (in `TreeShapedCode.v`) – In the definition of the LTS, we have to be careful such that 1. Coq allows us to extract witnesses and 2. we still don't have duplicate states. The assumption of groundedness is only necessarily in the uniqueness (and not in existence), which is proven as a separate Lemma `MapDefinesTreeUniquely`.

Example 3.7 $\triangleq$ `groundedness_is_needed_for_uniqueness` (in `TreeShapedCode.v`).

Definition 3.9 $\triangleq$ `contraction` (in `Contraction.v`) – In the Coq version, we do not restrict to reachable states, because it makes the reasoning easier to if $\mathcal{M}$ and $\alpha_{\mathcal{R}}(\mathcal{M})$ have the same states. For the visualization of the examples, it is easier to omit unreachable states. In our notion of isomorphism (Definition 2.8), every LTS is isomorphic to its restriction to reachable states.

Proposition 3.11 $\,\hat{=}\,$ `contraction_monotone` (in `LTS.v`).

Definition 4.1 $\,\hat{=}\,$ `refinement` (in `Refinement.v`).

Proposition 4.3 $\,\hat{=}\,$ `refinement_monotone` (in `Refinement.v`).

Proposition 4.4 $\,\hat{=}\,$ `refinement_preserves_determinism` (in `Refinement.v`).

Theorem 4.5 $\,\hat{=}\,$ `refinement_galois_forward`/`refinement_galois_backward` (in `Refinement.v`) – In Coq, the assumption of the first direction is spelled out as: $\mathcal{R}$ is defined for all action codes that appear in $\mathcal{N}$

Definition 5.1 $\,\hat{=}\,$ `concretization` (in `Concretization.v`) – In Coq, we have two equivalent definitions of the transition relation: on the one hand via the rules (1_γ)–(4_γ) and on the other hand as one formula.

Definition 5.3 $\,\hat{=}\,$ `icomplete` (in `Concretization.v`).

Theorem 5.5 $\,\hat{=}\,$ `concretization_galois_connection` (in `Concretization.v`) – In the proof of direction ($\Rightarrow$), we use the law of excluded middle for the property that for a $w \in A^*$ there is some or there is no $b \in B$ with $w \leq \mathcal{R}(b)$; in the constructive setting, this becomes an explicit assumption.

Example 5.6 $\,\hat{=}\,$ `concretization_galois_connection_lts` (in `Concretization.v`).

Example 5.7 $\,\hat{=}\,$ `simulation_from_mealy_to_own_concretization` (in `MealyExample.v`) – We have an auxiliary lemma `mealy_example_code1_is_complete` showing that $\mathcal{R}$ is indeed $\mathcal{I}$-complete for $\mathcal{N}$; there is a non-trivial case, in which we verify inductively that there is no $w \in A^*$ such that $q_0 \xrightarrow{\mathcal{R}^* w} q_3$.

Lemma 5.8 $\,\hat{=}\,$ `icomplete_for_concretization` (in `Concretization.v`).

Corollary 5.9 $\,\hat{=}\,$ `concretization_monotone` (in `Concretization.v`).

Proposition 5.11 $\,\hat{=}\,$ `concretization_preserves_determinism` (in `Concretization.v`).

Theorem 5.12 $\,\hat{=}\,$ `concretization_galois_insertion` (in `Concretizaton.v`).

Definition 6.1 $\,\hat{=}\,$ `compose_codemap` (in `CodeMap.v`).

Lemma 6.3 $\,\hat{=}\,$ `compose_code` (in `CodeMap.v`).

Theorem 6.4 $\,\hat{=}\,$ `contraction_code_composition` (in `Contraction`) – Item for α

Theorem 6.4 $\,\hat{=}\,$ `refinement_preserves_code_composition` (in `RefinementCodeComposition`) – Item for ϱ.

Remark 6.5 $\,\hat{=}\,$ `contraction_does_not_preserve_code_composition` (in `Contraction.v`) – In Appendix B on page 36, we visualize the involved LTSs for the counter example. In the Coq formalization, we directly show that the existence of an isomorphism leads to a contradiction.

Definition 7.3 $\,\hat{=}\,$ `Determinate` (in `Adaptor.v`) – In the formalization, we define *determinate* for general codes $\mathcal{R} \in \mathsf{Code}(A, B)$ and model that two action symbols $a_1, a_2 \in A$ having the same input component by a relation $\mathcal{I}_A \subseteq A \times A$ – in the same style as for the concretization operator. Likewise, we consider a relation $\mathcal{I}_B \subseteq B \times B$ that models that two $b_1, b_2 \in B$ have the same input component.

Proposition 7.7 $\,\hat{=}\,$ `contraction_preserves_determinism` (in `Adaptor.v`) – The formalized property *determinate* is generic in two relations $\mathcal{I}_A \subseteq A \times A$ and $\mathcal{I}_B \subseteq B \times B$, and likewise, the formalization of this proposition uses a notion of determinism generic in such a relation $\mathcal{I}$. Thus, we obtain the proposition in the paper as a special case for the relations $\mathcal{I}_A$ resp. $\mathcal{I}_B$ modeling same input (cf. Example 5.2). As another instance, we obtain that contraction $\alpha_\mathcal{R}$ for any code $\mathcal{R}$ sends deterministic LTSs to deterministic LTSs.

B   Omitted Proofs and Additional Details

Proof of Lemma 3.5

Take equation (2) as the definition of f:

$$f(b) = \begin{cases} w & \text{if there are } r \in L, w \in A^* \text{ with } r_0 \overset{w}{\Longrightarrow}_{\mathcal{R}} r, l(r) = b, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since $r_0 \notin L$, the range of f indeed restricts to $A^+ \subsetneq A^*$. For well-definedness of f, first note that due to the injectivity of the labeling $l\colon L \to B$, there is at most one $r \in L$ with $l(r) = b$, and by $\mathcal{R}$ being tree-shaped, there is precisely one path $r_0 \Longrightarrow_{\mathcal{R}} r$. Let us now prove the required properties of this partial map:

- **Prefix-freeness** (1). Consider $b, b' \in B$ with $f(b) \leq f(b')$. Hence, we have runs

  $$r_0 \overset{w}{\Longrightarrow}_{\mathcal{R}} r \quad r \in L \quad l(r) = b \quad w = f(b)$$

  and

  $$r_0 \overset{w'}{\Longrightarrow}_{\mathcal{R}} r' \quad r' \in L \quad l(r') = b' \quad w' = f(b').$$

  By $f(b) \leq f(b')$, there is some $u \in A^*$ such that $wu = w'$. Thus, the run for $w' = f(b')$ can be decomposed for some $\bar{r} \in Q^{\mathcal{R}}$:

  $$r_0 \overset{w}{\Longrightarrow}_{\mathcal{R}} \bar{r} \overset{u}{\Longrightarrow}_{\mathcal{R}} r'.$$

  The LTS $\mathcal{R}$ is required to be deterministic, so $\bar{r} = r$. Since $r \in L$ is a leaf, i.e. a dead-lock state, we necessarily have $u = \varepsilon$ and $r = r'$. Finally, $b = l(r) = l(r') = b'$.
- **Characterizing property** (2). Verification is immediate because both the property (2) and the definition r use the same witnesses, and every witness $w \in A^*$ must be a non-empty word.
- **Uniqueness.** Consider another prefix-free partial map $g\colon B \rightharpoonup A^+$ satisfying

  $$\text{for all } b \in B, w \in A^+\colon \qquad g(b) = w \quad \text{iff} \quad \exists r \in L\colon r_0 \overset{w}{\Longrightarrow}_{\mathcal{R}} r, \ l(r) = b.$$

  Then, it is immediate that $\mathsf{dom}(g) = \mathsf{im}(L) = \mathsf{dom}(f)$ and that $g(b) = f(b)$ for all $b \in \mathsf{im}(L)$. ◀

Proof of Lemma 3.6

Define action code $\mathcal{R}$ as follows:

- $R := \{w \in A^* \mid w = \varepsilon \text{ or } \exists b \in \mathsf{dom}(f)\colon w \leq f(b)\}$,
- $r_0 := \varepsilon$,
- we put $v \overset{a}{\longrightarrow}_{\mathcal{R}} w$ iff there is $a \in A$ with $va = w$,
- $l(w)$ is the unique $b \in B$ with $f(b) = w$.

Let us first verify that this is a well-defined action code:

- There is nothing to be verified about the state set R (note that we do not require finiteness in the definition of action code).
- By definition of R, the initial state $r_0 := \varepsilon$ is an element of R.
- For the transition structure we need to verify several properties:

- It is grounded because for every $w \in A^*$ with $w \leq f(b)$, $b \in \mathsf{dom}(f)$, the state $f(b)$ is the witnessing leaf (i.e. dead-lock state) below w.
  - It is deterministic: whenever we have $v \xrightarrow{a}_{\mathcal{R}} w$ and $v \xrightarrow{a}_{\mathcal{R}} w'$, we obtain $w = va = w'$.
  - It is tree-shaped: for each $w \in R$, there is a unique run to it, namely $r_0 \xRightarrow{w}_{\mathcal{R}} w$.
- Define set $L \subseteq R$ by $L = \{f(b) \mid b \in B\}$. Then L does not contain the initial state r_0, but every leaf $w \in Q \setminus \{r_0\}$ is in L. Conversely, every $w \in L$ is a leaf because f is prefix-free (1).
  Likewise, $l: L \to B$ is injective because f is prefix-free (1).

For uniqueness, consider another tree-shaped action code $\bar{\mathcal{R}}$ with set of non-root leaves $\bar{L}$ satisfying (2), concretely

$$\text{for all } b \in B, w \in A^+: f(b) = w \quad \text{iff} \quad \exists \bar{r} \in \bar{L}: r_0 \xRightarrow{w}_{\bar{\mathcal{R}}} \bar{r} \quad l^{\bar{\mathcal{R}}}(\bar{r}) = b. \tag{3}$$

We now need to establish an isomorphism $\phi: \bar{\mathcal{R}} \to \mathcal{R}$. Define the map $\phi: R^{\bar{\mathcal{R}}} \to R^{\mathcal{R}}$ by

$$\phi(\bar{r}) = \text{the unique } w \in A^* \text{ with } r_0^{\bar{R}} \xRightarrow{w}_{\bar{\mathcal{R}}} \bar{r}.$$

This map is well defined because $\bar{\mathcal{R}}$ is tree-shaped. To see that $\phi(\bar{r}) \in A^*$ is also in $R^{\mathcal{R}} \subseteq A^*$ for every state $\bar{r}$ of $\bar{\mathcal{R}}$, let $\bar{r}' \in \bar{L}$ be an arbitrary leaf with $\bar{r} \xRightarrow{u}_{\bar{\mathcal{R}}} \bar{r}'$ in $\bar{\mathcal{R}}$ – such a leaf exists because every action code $\bar{\mathcal{R}}$ is required to be grounded. Thus, for $b = l^{\mathcal{R}}(\bar{r}')$ we have $\phi(\bar{r}) \leq wu = f(b)$ by (3), and so $\phi(\bar{r}) \in R^{\mathcal{R}}$ by above definition of $\mathcal{R}$. With our definition of ϕ and $\longrightarrow_{\mathcal{R}}$, it is mechanical to check that $q \xrightarrow{a} q'$ iff $\phi(q) \xrightarrow{a} \phi(q')$. Since $\bar{\mathcal{R}}$ is tree-shaped, ϕ is injective. For surjectivity, consider $w \in R^{\mathcal{R}}$. If $w = \varepsilon$ then we have $\phi(r_0^{\bar{R}}) = \varepsilon$. Otherwise, pick any $b \in \mathsf{dom}(f)$ with $w \leq f(b)$. By (3), we have $\bar{r} \in \bar{L}$ with $q_0^{\bar{\mathcal{R}}} \xRightarrow{f(b)}_{\bar{\mathcal{R}}} r$ in $\bar{\mathcal{R}}$. The intermediate state $\bar{r}'$ with

$$q_0^{\bar{\mathcal{R}}} \xRightarrow{w}_{\bar{\mathcal{R}}} \bar{r}' \xRightarrow{u}_{\bar{\mathcal{R}}} \bar{r} \qquad w\,u = f(b)$$

does satisfy $\phi(\bar{r}') = w$. Thus, ϕ is surjective and bijective in total.

Also, the labelling is preserved by $\phi: \bar{\mathcal{R}} \to \mathcal{R}$ because for every leaf $\bar{r} \in \bar{L}$ with

$$r_0^{\bar{\mathcal{R}}} \xRightarrow{w}_{\bar{\mathcal{R}}} \bar{r}$$

we have $f(l^{\bar{\mathcal{R}}}(\bar{r})) = w$ by (3), and so

$$l(\phi(\bar{r})) = l(w) \overset{\text{def } l}{=} l^{\bar{\mathcal{R}}}(\bar{r}).$$

because $b := l^{\bar{\mathcal{R}}}(\bar{r})$ satisfies $f(b) = w$. ◀

Proof of Proposition 3.11

Given a simulation T from $\mathcal{M}$ to $\mathcal{N}$, we will show that $S := T \cap (Q^{\alpha(\mathcal{M})} \times Q^{\alpha(\mathcal{N})})$ is a simulation relation from $\alpha_{\mathcal{R}}(\mathcal{M})$ to $\alpha_{\mathcal{R}}(\mathcal{N})$; in other words $S \subseteq Q^{\alpha(\mathcal{M})} \times Q^{\alpha(\mathcal{N})}$ is the restriction of $T \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$ to the subsets $Q^{\alpha(\mathcal{M})} \subseteq Q^{\mathcal{M}}$ and $Q^{\alpha(\mathcal{N})} \subseteq Q^{\mathcal{N}}$.

For the initial states, we directly have $(q_0^{\alpha(\mathcal{M})}, q_0^{\alpha(\mathcal{N})}) \in S$ because

$$(q_0^{\alpha(\mathcal{M})}, q_0^{\alpha(\mathcal{N})}) = (q_0^{\mathcal{M}}, q_0^{\mathcal{N}}) \in T.$$

For transitions, consider $(q, p) \in S$ and $q \xrightarrow{b} q'$ in $\alpha(\mathcal{M})$. By definition of $\alpha(\mathcal{M})$, we have

$$b \in \mathsf{dom}(\mathcal{R}) \text{ and } q \xRightarrow{\mathcal{R}(b)} q' \qquad \text{in } \mathcal{M}.$$

In $\mathcal{M}$ and $\mathcal{N}$, the states $(q,p) \in S \subseteq T$ are also related by the simulation, and so we obtain

$$p \xRightarrow{\mathcal{R}(b)} p' \qquad \text{in } \mathcal{N} \text{ with } (q',p') \in T.$$

By the definition of α, we have $p' \in Q^{\alpha(\mathcal{N})}$ and $p \xrightarrow{b} p'$ in $\alpha(\mathcal{N})$. Thus, also $(q',p') \in S$ as desired. ◀

▶ **Example B.1** (Trace semantics). The trace semantics *trace* is closely related to concretiza-tion. Define $\delta \colon \mathsf{LTS}(A) \to \mathsf{LTS}(A+\{\$\})$ by $\delta((Q,q_0,\longrightarrow)) = (Q+\{\$\}, q_0, \longrightarrow \cup \{(q,\$) \mid q \in Q\})$ and define the action code $\mathcal{R} \in \mathsf{Code}(A+\{\$\}, B)$ to $B := A^*$ such that the concrete word $a_1 \cdots a_n \$$ is related to the abstract symbol $a_1 \cdots a_n \in B$, then $\alpha_{\mathcal{R}} \circ \delta \colon \mathsf{LTS}(A) \to \mathsf{LTS}(A^*)$ sends every $\mathcal{M} \in \mathsf{LTS}(A)$ to a system in $\mathsf{LTS}(A^*)$ with states $\{q_0, \$\}$ and transitions

$$q_0 \xrightarrow{w} \$ \quad \Longleftrightarrow \quad w \in trace(\mathcal{M}).$$

Proof of Proposition 4.3

Let $S \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$ be the simulation witnessing $\mathcal{M} \sqsubseteq \mathcal{N}$. Let us verify that

$$T := \{((q,w),(p,w')) \in Q^{\varrho(\mathcal{M})} \times Q^{\varrho(\mathcal{N})} \mid w = w', (q,p) \in S\}$$

is a simulation from $\varrho(\mathcal{M})$ to $\varrho(\mathcal{N})$. Clearly,

$$q_0^{\varrho(\mathcal{M})} = (q_0^{\mathcal{M}}, \varepsilon) \; T \; (q_0^{\mathcal{N}}, \varepsilon) = q_0^{\varrho(\mathcal{N})}.$$

Consider a pair in T, i.e. (q,w), (p,w) with $q \; S \; p$:

1. For transitions produced by rule (2_ϱ),

$$(q,w) \xrightarrow{a}_{\varrho(\mathcal{M})} (q',\varepsilon),$$

we have $q \xrightarrow{b}_{\mathcal{M}} q'$ for some $b \in \mathsf{dom}(\mathcal{R})$ with $\mathcal{R}(b) = wa$ by rule assumption. Since $(q,p) \in S$, the simulation S provides us with a transition

$$p \xrightarrow{b}_{\mathcal{N}} p' \qquad \text{in } \mathcal{N} \qquad \text{and} \qquad (q',p') \in S.$$

Thus, we can apply rule (2_ϱ) in $\varrho(\mathcal{N})$ to obtain

$$(p,w) \xrightarrow{a}_{\varrho(\mathcal{N})} (p',\varepsilon) \qquad \text{in } \varrho(\mathcal{N})$$

which satisfies $(q',\varepsilon) \; T \; (p',\varepsilon)$ by definition of T.

2. For transitions produced by rule (1_ϱ),

$$(q,w) \xrightarrow{a}_{\varrho(\mathcal{M})} (q,wa),$$

we have $(q,wa) \in Q^{\varrho(\mathcal{M})}$. Obviously, $wa \neq \varepsilon$, and so by the definition of $Q^{\varrho(\mathcal{M})}$, there must be at least one transition $q \xrightarrow{b}_{\mathcal{M}} q'$ with $wa \lneqq \mathcal{R}(b)$. By the definition of T, we have $(q,p) \in S$ and so the simulation S provides us with some $p \xrightarrow{b}_{\mathcal{N}} p'$ in $\mathcal{N}$ with $(q',p') \in S$. Hence, $(p,wa) \in Q^{\varrho(\mathcal{N})}$ and so we have $(p,w) \xrightarrow{a}_{\varrho(\mathcal{N})} (p,wa)$ by rule (1_ϱ) and $(q,wa) \; T \; (p,wa)$ by definition of T. ◀

Proof of Proposition 4.4

Consider $(q, w) \in Q^{\varrho(\mathcal{M})}$ and transitions

$$(q, w) \xrightarrow{a} (q_1, w_1) \qquad \text{and} \qquad (q, w) \xrightarrow{a} (q_2, w_2) \qquad \text{in } \varrho(\mathcal{M}).$$

We show $(q_1, w_1) = (q_2, w_2)$ by case distinction:

1. Case: there is $b \in \mathsf{dom}(\mathcal{R})$ with $\mathcal{R}(b) = wa$. Then there is a unique such b because $\mathcal{R}$ is prefix-free. Also because $\mathcal{R}$ is prefix-free, there is no $b' \in \mathsf{dom}(\mathcal{R})$ such that $wa \lneq \mathcal{R}(b')$, and so $(q, wa) \notin Q^{\varrho(\mathcal{M})}$ and so neither of the transitions were produced by rule (1_ϱ). Thus, both transitions come from rule (2_ϱ) and thus we have $w_1 = \varepsilon = w_2$ and transitions:

   $$q \xrightarrow{b}_{\mathcal{M}} q_1 \qquad \text{and} \qquad q \xrightarrow{b}_{\mathcal{M}} q_2.$$

   By assumption, $\mathcal{M}$ is deterministic, so $q_1 = q_2$, and so $(q_1, w_1) = (q_2, w_2)$.
2. Case: there is no $b \in \mathsf{dom}(\mathcal{R})$ with $\mathcal{R}(b) = wa$. Thus, neither of the transitions can be produced by rule (2_ϱ) and so both were produced by rule (1_ϱ). Then, we necessarily have

   $$(q_1, w_1) = (q, wa) = (q_2, w_2). \qquad \blacktriangleleft$$

▶ **Lemma B.2.** *For all $\mathcal{M} \in \mathsf{LTS}(B)$ and $b \in \mathsf{dom}(\mathcal{R})$ of an action code $\mathcal{R} \in \mathsf{Code}(A, B)$, and $q \in Q^{\mathcal{M}}$:*

$$q \xrightarrow{b} q' \text{ in } \mathcal{M} \qquad \text{iff} \qquad (q, \varepsilon) \xRightarrow{\mathcal{R}(b)} (q', \varepsilon) \text{ in } \varrho(\mathcal{M}).$$

Proof of Lemma B.2

For $(\Rightarrow)$, $\mathcal{R}$ is defined for b and we yield $\mathcal{R}(b) = a_1 \cdots a_n$ with $1 \leq n$ and $a_i \in A$ for all $1 \leq i \leq n$. Given $q \xrightarrow{b} q'$ in $\mathcal{M}$, we can construct a path in $\varrho_{\mathcal{R}}(\mathcal{M})$ by $n - 1$ applications of rule (1_ϱ) and one application of rule (2_ϱ):

$$(q, \varepsilon) \xrightarrow{a_1} (q, a_1) \xrightarrow{a_2} (q, a_1 a_2) \cdots \xrightarrow{a_{n-1}} (q, a_1 \cdots a_{n_1}) \xrightarrow{a_n} (q', \varepsilon) \qquad \text{in } \varrho_{\mathcal{R}}(\mathcal{M}),$$

with other notation for $\mathcal{R}(b) = a_1 \cdots a_n$:

$$(q, \varepsilon) \xRightarrow{\mathcal{R}(b)} (q', \varepsilon) \qquad \text{in } \varrho_{\mathcal{R}}(\mathcal{M}),$$

For $(\Leftarrow)$, consider a generalized transition $(q, \varepsilon) \xRightarrow{\mathcal{R}(b)} (q', \varepsilon)$ in $\varrho(\mathcal{M})$. Of course $\mathcal{R}(b) \in A^+$ so we can write it as $\mathcal{R}(b) = wa$ for $w \in A^*$ and $a \in A$. Then, we can consider the intermediate state of the given run for $\mathcal{R}(b)$:

$$(q, \varepsilon) \xRightarrow{w}_{\varrho(\mathcal{M})} (\bar{q}, \bar{w}) \xrightarrow{a}_{\varrho(\mathcal{M})} (q', \varepsilon).$$

All prefixes $v \leq w$ satisfy $v \lneq \mathcal{R}(b)$, so the first transitions for w must have been produced by rule (1_ϱ). Hence, $(\bar{q}, \bar{w}) = (q, w)$. The final a-transition must have been produced by (2_ϱ) because the second component of the tuple is ε. The assumption of this rule contains $q \xrightarrow{b}_{\mathcal{M}} q'$, as desired. $\qquad \blacktriangleleft$

▶ **Lemma B.3.** *For every transition $(q, w) \xrightarrow{a} (q', w')$ in $\varrho(\mathcal{M})$, there is some transition $q \xrightarrow{b} q''$ in $\mathcal{M}$ and $u \in A^*$ such that $wau = \mathcal{R}(b)$ and $(q', w') \xRightarrow{u} (q'', \varepsilon)$ in $\varrho_{\mathcal{R}}(\mathcal{M})$.*

Proof of Lemma B.3

For the transition $(q, w) \xrightarrow{a} (q', w')$, distinguish two cases:

- If $w' = \varepsilon$, then the transition must have been produced by rule (2_ϱ). Thus, the rule assumption contains a transition $q \xrightarrow{b}_{\mathcal{M}} q'$ with $wa = \mathcal{R}(b)$. This is the desired witness $q'' := q'$ and $u = \varepsilon$.
- If $w' \neq \varepsilon$, then the transition must have been produced by rule (1_ϱ) and so $q' = q$ and $w' = wa$. The definition of $Q^{\varrho(\mathcal{M})}$ unfolded for $(q, wa) \in Q^{\varrho(\mathcal{M})}$ yields that there exists a transition $q \xrightarrow{b}_{\mathcal{M}} q''$ with $wa \lneqq \mathcal{R}(b)$. Let $v \in A^*$ and $b \in A$ such that $wavb = \mathcal{R}(b)$. Since $wav \lneqq \mathcal{R}(b)$, we can produce further transitions using rule (1_ϱ) for the letters in $v \in A^*$ and can conclude using rule (2_ϱ) for b:

$$(q', w') = (q, wa) \xLongrightarrow{v}_{\varrho(\mathcal{M})} (q, wau) \xrightarrow{b}_{\varrho(\mathcal{M})} (q'', \varepsilon).$$

This is the desired transition (with $u := vb$). ◀

Proof of Theorem 4.5

Fix systems $\mathcal{N} \in \mathsf{LTS}(B)$, $\mathcal{M} \in \mathsf{LTS}(A)$, and as usual we omit index $\mathcal{R}$ from α and ϱ.

1. For direction ($\Rightarrow$), we assume that $\mathcal{N}$ restricts to the subclass $\mathsf{LTS}(\mathsf{dom}(\mathcal{R})) \subseteq \mathsf{LTS}(B)$. Hence, in the following we simply put $B := \mathsf{dom}(\mathcal{R})$ without loss of generality. Consider $\varrho(\mathcal{N}) \sqsubseteq \mathcal{M}$, witnessed by the simulation $S \subseteq Q^{\varrho(\mathcal{N})} \times Q^{\mathcal{M}}$. We verify that we have a simulation T between $\mathcal{N}$ and $\alpha(\mathcal{M})$ defined by:

   $$T := \{(p, q) \in Q^{\mathcal{N}} \times Q^{\alpha(\mathcal{M})} \mid ((p, \varepsilon), q) \in S\}$$

   The definition of T is well-typed because $Q^{\alpha(\mathcal{M})} \subseteq Q^{\mathcal{M}}$, and moreover,

   $$\{(p, \varepsilon) \mid p \in Q^{\mathcal{N}}\} \subseteq Q^{\varrho(\mathcal{N})}.$$

   The relation T relates the initial states $q_0^{\mathcal{N}} \ T \ q_0^{\alpha(\mathcal{M})}$ because

   $$(q_0^{\mathcal{N}}, \varepsilon) = q_0^{\varrho(\mathcal{N})} \ S \ q_0^{\mathcal{M}} = q_0^{\alpha(\mathcal{M})}.$$

   Now, suppose $p \ T \ q$ and $p \xrightarrow{b}_{\mathcal{N}} p'$. By assumption $\mathsf{dom}(\mathcal{R}) = B$, we can apply Lemma B.2 and obtain

   $$(p, \varepsilon) \xLongrightarrow{\mathcal{R}(b)}_{\varrho(\mathcal{N})} (p', \varepsilon) \qquad \text{in } \varrho(\mathcal{N}).$$

   The simulation S from $\varrho(\mathcal{N})$ to $\mathcal{M}$ transforms this into a path

   $$q \xLongrightarrow{\mathcal{R}(b)}_{\mathcal{M}} q' \text{ in } \mathcal{M} \quad \text{with} \quad (p', \varepsilon) \ S \ q'.$$

   Since $q \in Q^{\alpha(\mathcal{M})}$ also $q' \in Q^{\alpha(\mathcal{M})}$ and thus

   $$q \xrightarrow{b}_{\alpha(\mathcal{M})} q' \text{ in } \alpha(\mathcal{M})$$
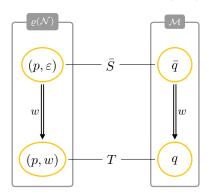
   and moreover $p' \ T \ q'$.
2. For direction ($\Leftarrow$), assume $\mathcal{N} \sqsubseteq \alpha(\mathcal{M})$. Let $S \subseteq Q^{\mathcal{N}} \times Q^{\alpha(\mathcal{M})}$ be a simulation relation from $\mathcal{N}$ to $\alpha(\mathcal{M})$. We define the relations $\bar{S} \subseteq Q^{\varrho(N)} \times Q^{\alpha(\mathcal{M})}$ and $T \subseteq Q^{\varrho(\mathcal{N})} \times Q^{\mathcal{M}}$:

   $$(p, \varepsilon) \ \bar{S} \ q \quad :\Leftrightarrow \quad p \ S \ q$$

$$(p, w)\, T\, q \quad :\Leftrightarrow \quad \exists \bar{q} \in Q^{\alpha(\mathcal{M})} \colon p\, S\, \bar{q} \wedge \bar{q} \overset{w}{\Longrightarrow}_{\mathcal{M}} q$$

So visually, every related pair $(p, w)\, T\, q$ entails states of the following form:

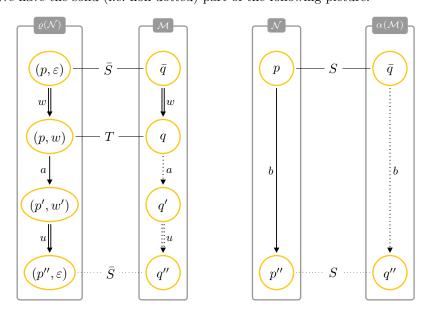We verify that T is a simulation from $\varrho(\mathcal{N})$ to $\mathcal{M}$.

Picking $\bar{q} := q$ and $w := \varepsilon$ shows that the related initial states $q_0^{\mathcal{N}}\, S\, q_0^{\alpha(\mathcal{M})}$ of $\mathcal{N}$ and $\alpha(\mathcal{M})$ imply

$$q_0^{\varrho(\mathcal{N})} = (q_0^{\mathcal{N}}, \varepsilon)\, T\, q_0^{\mathcal{M}}.$$

Suppose $(p, w)\, T\, q$ and $(p, w) \overset{a}{\longrightarrow}_{\varrho(\mathcal{N})} (p', w')$. Lemma B.3 shows that $wa \in A^+$ sits 'below' some $b \in B$ in the action code $\mathcal{R}$: concretely, there are $u \in A^*$, and $b \in B$ such that:

$$(p', w') \overset{u}{\Longrightarrow}_{\varrho(\mathcal{N})} (p'', \varepsilon) \qquad \text{and} \qquad wau = \mathcal{R}(b) \qquad \text{and} \qquad p \overset{b}{\longrightarrow}_{\mathcal{N}} p''.$$

We have the solid (i.e. non-dotted) part of the following picture:

Using the simulation property of $p\, S\, \bar{q}$, we obtain $q'' \in Q^{\alpha(\mathcal{M})}$ with $\bar{q} \overset{b}{\longrightarrow}_{\alpha(\mathcal{M})} q''$ and $p''\, S\, q''$. The transition $\bar{q} \overset{b}{\longrightarrow} q''$ in $\alpha(\mathcal{M})$ must have come from a path $\bar{q} \overset{\mathcal{R}(b)}{\Longrightarrow} q''$ in $\mathcal{M}$. Denote the intermediate states for the decomposition $\mathcal{R}(b) = wau$ by $q_w, q' \in Q^{\mathcal{M}}$:

$$\bar{q} \overset{w}{\Longrightarrow}_{\mathcal{M}} q_w \overset{a}{\longrightarrow}_{\mathcal{M}} q' \overset{u}{\Longrightarrow} q''$$

The assumption that $\mathcal{M}$ is deterministic enforces that $q = q_w$, so $q \xrightarrow{au}_{\mathcal{M}} q''$ as shown in the picture. Also, $p''\, S\, q''$ directly shows $(p'', \varepsilon)\, \bar{S}\, q''$. Now, the picture is complete. For the final proof that T relates (p', w') and q', we distinguish cases:

a. Case $u = \varepsilon$: Then, $p' = p''$, $w' = \varepsilon$ and $q' = q''$. Thus, $\big((p', w'), q'\big) \in \bar{S} \subseteq T$.

b. Case $u \neq \varepsilon$: Then, $wa \lneqq \mathcal{R}(b)$ and so $wa \neq \mathcal{R}(b')$ for all $b' \in B$ by prefix-freeness. So the a-transition can only come from rule (1_ϱ); hence $p = p'$ and $w' = wa$. Finally, $(p', w') = (p, wa)\, T\, q'$ by the definition of T and $\bar{q} \xrightarrow{wa} q'$. ◀

Details for Remark 4.7

Contraction α has no left-adjoint if we allow the action codes to relate multiple concrete words $w_1, w_2 \in A^+$ with the same abstract symbol $b \in B$. We show the special case for both words having length 1, which generalizes to general words. Assume there are two symbols $a_1, a_2 \in A, a_1 \neq a_2$ both related to $b \in B$. Denote a system with two states and one transition directly by $(\bullet \xrightarrow{b} \bullet)$ (the initial state is the left-hand one). Then, we have

$$\alpha(\bullet \xrightarrow{a_1} \bullet) = (\bullet \xrightarrow{b} \bullet) = \alpha(\bullet \xrightarrow{a_2} \bullet).$$

Then, there exists no left adjoint ϱ. Such an adjoint, applied to $(\bullet \xrightarrow{b} \bullet)$ would need to satisfy

$$\varrho(\bullet \xrightarrow{b} \bullet) \sqsubseteq (\bullet \xrightarrow{a_1} \bullet) \quad \text{and} \quad \varrho(\bullet \xrightarrow{b} \bullet) \sqsubseteq (\bullet \xrightarrow{a_2} \bullet)$$

Hence, the initial state in $\varrho(\bullet \xrightarrow{b} \bullet)$ is a leaf state, i.e. $\varrho(\bullet \xrightarrow{b} \bullet)$ is the bottom element for the simulation order $\sqsubseteq$. This easily leads to a contradiction after using the Galois connection in the other direction again. ◀

▶ **Lemma B.4.** *For every $\mathcal{M} \in \mathsf{LTS}(B)$, $q \in Q^\mathcal{M}$, $\mathcal{R} \in \mathsf{Code}(A, B)$, $\mathcal{I}$ being the identity relation on A, and $b \in \mathsf{dom}(\mathcal{R})$:*

1. *Whenever $q \xrightarrow{b}_{\mathcal{M}} q'$ then $(q, \varepsilon) \xRightarrow{\mathcal{R}(b)}_{\gamma(\mathcal{M})} (q', \varepsilon)$.*
2. *Whenever $(q, \varepsilon) \xRightarrow{\mathcal{R}(b)}_{\gamma(\mathcal{M})} \bar{q}$, then $\bar{q} = (q', \varepsilon)$ for some $q' \in Q^\mathcal{M}$ with $q \xrightarrow{b}_{\mathcal{M}} q'$.*

Proof of Lemma B.4

1. Given $q \xrightarrow{b}_{\mathcal{M}} q'$ in $\mathcal{M}$, write $\mathcal{R}(b) \in A^+$ as $\mathcal{R}(b) = wa$ for $w \in A^*$ and $a \in A$. Write $w = w_1 \cdots w_n$, with $n \in \mathbb{N}$ and $w_i \in A$, for $1 \leq i \leq n$. For every $1 \leq i \leq n$, the word $w_1 \cdots w_i \in A^*$ is contained in N because $w_1 \cdots w_i \lneqq \mathcal{R}(b)$. Hence, rule (1_γ) provides us with transitions

$$(q, \varepsilon) \xrightarrow{w_1}_{\gamma(\mathcal{M})} (q, w_1) \xrightarrow{w_2}_{\gamma(\mathcal{M})} (q, w_1 w_2) \cdots \xrightarrow{}_{\gamma(\mathcal{M})} (q, w_1 \cdots w_n) = (q, w)$$

Finally, rule (2_γ) has the assumptions $q \xrightarrow{b}_{\mathcal{M}} q'$ and $wa = \mathcal{R}(b)$ fulfilled, so we have

$$(q, \varepsilon) \xRightarrow{w}_{\gamma(\mathcal{M})} (q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} (q', \varepsilon),$$

i.e. $(q, \varepsilon) \xRightarrow{\mathcal{R}(b)}_{\gamma(\mathcal{M})} (q', \varepsilon)$, as desired.

2. Assume $(q, \varepsilon) \xRightarrow{\mathcal{R}(b)}_{\gamma(\mathcal{M})} \bar{q}$. As before, decompose $\mathcal{R}(b)$ into $\mathcal{R}(b) = wa$ for $w \in A^*$ and $a \in A$. Write $w = w_1 \cdots w_n$ for $w_i \in A$, $n \in \mathbb{N}$. For every i, $1 \leq i < n$, any transition

$$(q, w_1 \cdots w_i) \xrightarrow{w_{i+1}}_{\gamma(\mathcal{M})} p$$

can only be produced by rule (1_γ), because $w_1 \cdots w_i w_{i+1} \in N$. Hence, $p = (q, w_1 \cdots w_{i+})$. Thus, the given run for $\mathcal{R}(b) \in A^+$ is of the form

$$(q, \varepsilon) \xrightarrow{w_1}_{\gamma(\mathcal{M})} (q, w_1) \xrightarrow{w_2}_{\gamma(\mathcal{M})} \cdots \xrightarrow{w_n}_{\gamma(\mathcal{M})} (q, w_1 \cdots w_n) = (q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} \bar{q}.$$

In order to see that the final a-transition is produced by rule (2_γ), note that $wa = \mathcal{R}(b)$ implies that $wa \notin N$ by prefix-freeness (1), and so rule (1_γ) can not have produced this a-transition to $\bar{q}$. Obviously, rule (3_γ) does not match either, and so only rule (2_γ) is left. Thus, there is some $q \xrightarrow{b}_{\mathcal{M}} q'$ and $\bar{q}$ is of the form $\bar{q} = (q', \varepsilon)$, as desired. ◄

Proof of Theorem 5.5

In the present proof, we restrict to the special case where $\mathcal{I} \subseteq A \times A$ is the identity relation on A, which we thus omit from the index in γ. The proof for a general $\mathcal{I}$ is entirely formalized in Coq. Since we have only one action code $\mathcal{R}$ at hand, we omit the index $\mathcal{R}$ for α and γ. We prove both directions seperately:

($\Rightarrow$) Let $\alpha(\mathcal{M}) \sqsubseteq \mathcal{N}$ be witnessed by the simulation $S \sqsubseteq Q^{\alpha(\mathcal{M})} \times Q^{\mathcal{N}}$. We show that $T \sqsubseteq Q^{\mathcal{M}} \times Q^{\gamma(\mathcal{N})}$ defined by

$$\begin{aligned} T := \ & \{(p', (q, w)) \mid p' \in Q^{\mathcal{M}}, (q, w) \in Q^{\gamma(\mathcal{N})}, \exists p \in Q^{\mathcal{M}}, p \xrightarrow{w}_{\mathcal{M}} p', (p, q) \in S\} \\ & \cup \{(p, \chi) \mid p \in Q^{\mathcal{M}}\} \end{aligned}$$

is a simulation. Note that if χ is omitted from $\gamma(\mathcal{N})$ if it is not reachable, and then we also omit it from T. For the initial states, we immediately have

$$(q_0^{\mathcal{M}}, q_0^{\gamma(\mathcal{N})}) = (q_0^{\mathcal{M}}, (q_0^{\mathcal{N}}, \varepsilon)) \in T$$

by the definition of T, because $q_0^{\mathcal{M}} \xrightarrow{\varepsilon} q_0^{\mathcal{M}}$ and $(q_0^{\mathcal{M}}, q_0^{\mathcal{N}}) = (q_0^{\alpha(\mathcal{M})}, q_0^{\mathcal{N}}) \in S$. Since T is defined as a union, we can verify the two parts seperately:

1. Consider $(p', (q, w)) \in T$ and $p' \xrightarrow{a}_{\mathcal{M}} p''$. By definition of T, there is some $p \in Q^{\mathcal{M}}$ with

$$p \xrightarrow{w}_{\mathcal{M}} p' \qquad \text{and} \qquad (p, q) \in S.$$

We distinguish whether $wa \in N$ and $wa \in \mathsf{im}(\mathcal{R})$:
   - If $wa \in N$, then we have $(q, w) \xrightarrow{a}_{\gamma(\mathcal{N})} (q, wa)$ by rule (1_γ) and $(p'', (q, wa)) \in T$ by definition of T.
   - If $wa \notin N$ and $wa \in \mathsf{im}(\mathcal{R})$, then there is some $b \in \mathsf{dom}(\mathcal{R})$ with $\mathcal{R}(b) = wa$. We have $p \xrightarrow{w}_{\mathcal{M}} p' \xrightarrow{a}_{\mathcal{M}} p''$ and so $p \xrightarrow{b}_{\alpha(\mathcal{M})} p''$ by the definition of $\alpha(\mathcal{M})$. Using that S is a simulation, we obtain a transition $q \xrightarrow{b}_{\mathcal{N}} q'$ in $\mathcal{N}$. By rule (2_γ), this translates into a transition $(q, w) \xrightarrow{a}_{\gamma(\mathcal{N})} (q', \varepsilon)$ in $\gamma(\mathcal{N})$. By definition of T, we find $(p'', (q', \varepsilon)) \in T$.
   - If $wa \notin N$ and $wa \notin \mathsf{im}(\mathcal{R})$, then we have $(q, w) \xrightarrow{\chi}_{\gamma(\mathcal{N})}$ by rule (3_γ). In this case, Then, we also have $(p', \chi) \in T$.

2. Consider $(p, \chi) \in T$ and $p \xrightarrow{a}_{\mathcal{M}} p'$ in $\mathcal{M}$. We have $\chi \xrightarrow{a}_{\gamma(\mathcal{N})} \chi$ by rule (4_γ) and $(p', \chi) \in T$, again.

($\Leftarrow$) Assume $M \sqsubseteq \gamma_{\mathcal{R}}(\mathcal{N})$ in $\mathsf{LTS}(A)$, witnessed by a simulation $S \subseteq Q^{\mathcal{M}} \times Q^{\gamma(\mathcal{N})}$. Define $T \subseteq Q^{\alpha(\mathcal{M})} \times Q^{\mathcal{N}}$ by

$$T := \{(p, q) \in Q^{\alpha(\mathcal{M})} \times Q^{\mathcal{N}} \mid (p, (q, \varepsilon)) \in S\}.$$

Here, we use that $Q^{\alpha(\mathcal{M})} \subseteq Q^{\mathcal{M}}$. For the initial states, note that $(q_0^{\alpha(\mathcal{M})}, q_0^{\mathcal{N}}) \in T$ because

$$(q_0^{\mathcal{M}}, (q_0^{\mathcal{N}}, \varepsilon)) = (q_0^{\mathcal{M}}, q_0^{\gamma(\mathcal{N})}) \in S.$$

For the remaining verification, consider $(p, q) \in T$ and a transition $p \xrightarrow{b}_{\alpha(\mathcal{M})} p'$ in $\alpha(\mathcal{M})$. By the definition of α, we have $b \in \mathsf{dom}(\mathcal{R})$ and a run

$$p \xRightarrow{\mathcal{R}(b)}_{\mathcal{M}} p' \qquad \text{in } \mathcal{M}.$$

Using that S is a simulation and that $(p, (q, \varepsilon)) \in S$, this yields a run

$$(q, \varepsilon) \xRightarrow{\mathcal{R}(b)}_{\gamma(\mathcal{N})} q' \qquad \text{in } \gamma(\mathcal{N}) \qquad \text{with} \qquad (p', q') \in S.$$

We do not know yet in which of the two components of $Q^{\gamma(\mathcal{N})} = (Q^{\mathcal{N}} \times N) \cup \{chi\}$ the state q' is. We investigate by decomposing the run for $\mathcal{R}(b) \in A^+$ into $wa = \mathcal{R}(b)$ for $w \in A^*$ and $a \in A$, calling the intermediate state $\bar{q} \in Q^{\gamma(\mathcal{N})}$:

$$(q, \varepsilon) \xRightarrow{w}_{\gamma(\mathcal{N})} \bar{q} \xrightarrow{a}_{\gamma(\mathcal{N})} q' \qquad \text{in } \gamma(\mathcal{N}).$$

Since $w \lneqq \mathcal{R}(b)$, we have $w \in N$. Looking at the rules for the transitions of $\gamma(\mathcal{N})$, we see that the only option for the transitions in $(q, \varepsilon) \xRightarrow{w}_{\gamma(\mathcal{N})} \bar{q}$ with $w \in N$ is via rule (1_γ), so we necessarily obtain $\bar{q} = (q, w)$:

$$(q, \varepsilon) \xRightarrow{w}_{\gamma(\mathcal{N})} (q, w) \xrightarrow{a}_{\gamma(\mathcal{N})} q'$$

Using that $wa = \mathcal{R}(b)$, only rule (2_γ) can have produced the transition $(q, w) \xrightarrow{a} q'$, hence $q' = (q'', \varepsilon)$ for some $q'' \in Q^{\mathcal{N}}$ with $q \xrightarrow{b}_{\mathcal{N}} q''$ in $\mathcal{N}$. In total, we have $(p', (q'', \varepsilon)) = (p', q') \in S$ and by the definition of T:

$$(p', q'') \in T \qquad q \xrightarrow{b}_{\mathcal{N}} q'' \quad \text{in } \mathcal{N}.$$

This shows that T is indeed a simulation. ◀

Proof of Corollary 5.9

Consider $\mathcal{M} \sqsubseteq \mathcal{N}$ in $\mathsf{LTS}(B)$. By the reflexivity, we have

$$\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M}) \sqsubseteq \gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M}) \qquad \text{in } \mathsf{LTS}(A).$$

Applying the Galois connection (Theorem 5.5) from right to left yields

$$\alpha_{\mathcal{R}}(\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M})) \sqsubseteq \mathcal{M} \qquad \text{in } \mathsf{LTS}(B).$$

By transitivity of $\sqsubseteq$ and $\mathcal{M} \sqsubseteq \mathcal{N}$, we obtain

$$\alpha_{\mathcal{R}}(\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M})) \sqsubseteq \mathcal{N} \qquad \text{in } \mathsf{LTS}(B).$$

Applying the Galois connection (Theorem 5.5) conversely from left to right yields the desired

$$\gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{M}) \sqsubseteq \gamma_{\mathcal{R}, \mathcal{I}}(\mathcal{N}) \qquad \text{in } \mathsf{LTS}(A).$$ ◀

Proof of Proposition 5.11

Again, we restrict to the case where $\mathcal{I}$ is the identity relation on A; the general proof is formalized in Coq. We verify the determinacy seperately for the disjoint components of $Q^{\gamma(\mathcal{M})} := (Q^{\mathcal{M}} \times N) \cup \{\chi\}$

1. For $(q, w) \in Q^{\mathcal{M}} \times N$ and two transitions

$$(q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} \bar{q}_1 \quad (q, w) \xrightarrow{a}_{\gamma(\mathcal{M})} \bar{q}_2$$

we distinguish cases like in the assumptions of the rules for $\longrightarrow_{\gamma(\mathcal{M})}$:

- If $wa \in N$, then both transitions have been produced by rule (1_γ) and so $\bar{q}_1 = (q, wa) = \bar{q}_2$.
- If $wa \notin N$ and $wa \in \mathsf{im}(\mathcal{R})$, then there are $b_1, b_2 \in \mathsf{dom}(\mathcal{R})$ with

$$q \xrightarrow{b_1}_{\mathcal{M}} q_1' \qquad \bar{q}_1 = (q_1', \varepsilon) \qquad \mathcal{R}(b_1) = wa$$

$$q \xrightarrow{b_2}_{\mathcal{M}} q_2' \qquad \bar{q}_2 = (q_2', \varepsilon) \qquad \mathcal{R}(b_2) = wa$$

  Since $\mathcal{R}\colon B \rightharpoonup A^+$ is prefix-free (1), it is in particular injective and so $b_1 = b_2$. The LTS $\mathcal{M}$ was assumed to be deterministic, thus $q_1' = q_2'$ and so $\bar{q}_1 = (q_1', \varepsilon) = (q_2', \varepsilon) = \bar{q}_2$.
- If $wa \notin N$ and $wa \notin \mathsf{im}(\mathcal{R})$, then both transitions have been produced by rule (3_γ) and so $\bar{q}_1 = \chi = \bar{q}_2$.

2. Any two outgoing transitions of χ

$$\chi \xrightarrow{a}_{\gamma(\mathcal{M})} \bar{q}_1 \quad \text{and} \quad \chi \xrightarrow{a}_{\gamma(\mathcal{M})} \bar{q}_2$$

have necessarily been created by (4_γ), and so $\bar{q}_1 = \chi = \bar{q}_2$.   ◄

Proof of Theorem 5.12

Again, we restrict to the case where $\mathcal{I}$ is the identity relation on A; the general proof is formalized in Coq. Since we have only one action code $\mathcal{R}$ at hand, we omit the index $\mathcal{R}$ in α and γ in this proof. The LTS $\alpha(\gamma(\mathcal{M}))$ has precisely the states

$$Q^{\alpha(\gamma(\mathcal{M}))} = \{(q, \varepsilon) \mid q \in Q^{\mathcal{M}}\}.$$

In order to see that, we find:

($\subseteq$) If $\bar{q} \in Q^{\alpha(\gamma(\mathcal{M}))}$, then there is a word $b_1 \cdots b_n \in \mathsf{dom}(\mathcal{R})^*$ and are states $\bar{q}_1, \ldots, \bar{q}_n$ with

$$q_0^{\gamma(\mathcal{M})} = (q_0^{\mathcal{M}}, \varepsilon) \xrightarrow{b_1}_{\alpha(\gamma(\mathcal{M}))} \bar{q}_1 \xrightarrow{b_2}_{\alpha(\gamma(\mathcal{M}))} \cdots \xrightarrow{b_n}_{\alpha(\gamma(\mathcal{M}))} \bar{q}_n = \bar{q}. \qquad \text{in } \alpha(\gamma(\mathcal{M})).$$

By definition of α, this corresponds to transitions

$$q_0^{\gamma(\mathcal{M})} = (q_0^{\mathcal{M}}, \varepsilon) \xrightarrow{\mathcal{R}(b_1)}_{\gamma(\mathcal{M})} \bar{q}_1 \xrightarrow{\mathcal{R}(b_2)}_{\gamma(\mathcal{M})} \cdots \xrightarrow{\mathcal{R}(b_n)}_{\gamma(\mathcal{M})} \bar{q}_n = \bar{q}. \qquad \text{in } \gamma(\mathcal{M}).$$

Applying Lemma B.4 to every $\bar{q}_i$, we obtain that $\bar{q} = (q, \varepsilon)$ for some $q \in Q^{\mathcal{M}}$. Note in particular, $\bar{q} \neq \chi$ and so $\chi \notin Q^{\alpha(\gamma(\mathcal{M}))}$.

($\supseteq$) The converse inclusion iterates the other direction of Lemma B.4: we assume $\mathcal{M} \in$ $\mathsf{LTS}(B)$ to be reachable, hence every state $q \in Q^{\mathcal{M}}$ is reachable via some word $b_1 \cdots b_n \in$ B^* by iterating Lemma B.4:

$$q_0^{\mathcal{M}} \xrightarrow{\ b_1 \cdots b_n \ }_{\mathcal{M}} q.$$

The assumption that $\mathcal{M} \in \mathsf{LTS}(\mathsf{dom}(\mathcal{R}))$ implies that $\mathcal{R} \colon B \rightharpoonup A^+$ is defined for every b_i, and thus (q, ε) is reachable in $Q^{\gamma(\mathcal{M})}$:

$$q_0^{\gamma(\mathcal{M})} = (q_0^{\mathcal{M}}, \varepsilon) \xrightarrow{\ \mathcal{R}(b_1) \cdots \mathcal{R}(b_n) \ }_{\gamma(\mathcal{M})} (q, \varepsilon).$$

And hence, the definition of α then sends this run to

$$q_0^{\alpha(\gamma(\mathcal{M}))} = q_0^{\gamma(\mathcal{M})} = (q_0^{\mathcal{M}}, \varepsilon) \xrightarrow{\ b_1 \cdots b_n \ }_{\alpha(\gamma(\mathcal{M}))} (q, \varepsilon).$$

and so $(q, \varepsilon) \in Q^{\alpha(\gamma(\mathcal{M}))}$.

The witnessing bijective bisimulation is

$$\phi \colon Q^{\mathcal{M}} \longrightarrow Q^{\alpha(\gamma(\mathcal{M}))} \qquad \phi(q) = (q, \varepsilon) \quad \in Q^{\alpha(\gamma(\mathcal{M}))} \subseteq Q^{\gamma(\mathcal{M})}.$$

By our above characterization of $Q^{\alpha(\gamma(\mathcal{M}))}$, ϕ is a bijection. It remains to verify that ϕ is a bisimulation:

- For every transition in $\alpha(\gamma(\mathcal{M}))$, concretely $(q, \varepsilon) \xrightarrow{\ b\ } (q', \varepsilon)$, we have $(q, \varepsilon) \xrightarrow{\ \mathcal{R}(b)\ } (q', \varepsilon)$ in $\gamma(\mathcal{M})$ by the definition of α. By Lemma B.4, this implies $q \xrightarrow{\ b\ } q'$ in $\mathcal{M}$; and indeed $\phi(q) = (q, \varepsilon)$ and $\phi(q') = (q', \varepsilon)$.
- Conversely, for every transition $q \xrightarrow{\ b\ } q'$ in $\mathcal{M}$, we have a transition

$$(q, \varepsilon) \xrightarrow{\ \mathcal{R}(b)\ } (q', \varepsilon) \qquad \text{in } \gamma(\mathcal{M})$$

by Lemma B.4 and by $b \in \mathsf{dom}(\mathcal{R})$ provided by the assumption $\mathcal{M} \in \mathsf{LTS}(\mathsf{dom}(\mathcal{R}))$. By the definition of α, we thus have

$$\phi(q) = (q, \varepsilon) \xrightarrow{\ b\ } (q', \varepsilon) = \phi(q') \qquad \text{in } \alpha(\gamma(\mathcal{M})).$$

In total, ϕ is an isomorphism in $\mathsf{LTS}(B)$. ◀

Details of Remark 6.2

Kleisli composition is a recipe to compose maps of the form $C \to T(B)$ and $B \to T(A)$ to a map of type $C \to T(A)$, where T is a monad. In our case, the monad is $T(X) = X^+ + 1$ where 1 is an arbitrary singleton and $+$ denotes disjoint union. This monad T itself is a combination of two monads: $S(X) = X^+$ is the free semigroup-monad. Monads corresponds to algebraic theories and the algebraic theory corresponding to S is that of semigroups. The monad $P(X) = X + 1$ is called the *maybe monad* (or sometimes called *optional* in programming), which allows to model partial maps. The algebraic theory corresponding to P is that of *pointed sets* (the theory consists of one nullary operation). Warning: even though $T(X) = P(S(X)) = X^+ + 1$ and $M(X) = X^*$ are naturally isomorphic functors, they are different monads, because M is the *list monad*, whose corresponding algebraic theory is that of monoids. ◀

Proof of Lemma 6.3

We need to show that $(\mathcal{R} * \mathcal{S})\colon C \rightharpoonup A^+$ is prefix-free. To this end, consider $c, c' \in \mathsf{dom}(\mathcal{R} * \mathcal{S})$ with

$$(\mathcal{R} * \mathcal{S})(c) \le (\mathcal{R} * \mathcal{S})(c').$$

Since $\mathcal{R} * \mathcal{S}$ is defined for both c and c' we can spell out the words as

$$(\mathcal{R} * \mathcal{S})(c) = \mathcal{R}(b_1) \cdots \mathcal{R}(b_n) \qquad \text{for } n \in \mathbb{N} \text{ and } \mathcal{S}(c) = b_1 \cdots b_n$$
$$(\mathcal{R} * \mathcal{S})(c') = \mathcal{R}(b'_1) \cdots \mathcal{R}(b'_m) \qquad \text{for } m \in \mathbb{N} \text{ and } \mathcal{S}(c') = b'_1 \cdots b'_m.$$

Note that we do not know yet whether n or m is bigger! We only know that

$$\mathcal{R}(b_1) \cdots \mathcal{R}(b_n) \le \mathcal{R}(b'_1) \cdots \mathcal{R}(b'_m). \tag{4}$$

We now show by induction that

$$\text{for all } i \text{ with } 0 \le i \le \min(n, m)\colon \quad b_i = b'_i.$$

- In the base case $i = 0$, there is nothing to be shown.
- In the step for i, assume that we have $\forall 0 \le j < i\colon b_i = b'_i$ as the induction hypothesis. Thus, we also have $\mathcal{R}(b_j) = \mathcal{R}(b'_j)$ for all $j < i$ and so the words

  $$\mathcal{R}(b_1) \cdots \mathcal{R}(b_i) \cdots R(b_n) \text{ and } \mathcal{R}(b'_1) \cdots \mathcal{R}(b'_i) \cdots \mathcal{R}(b'_m).$$

  have a common prefix $\mathcal{R}(b_1) \cdots \mathcal{R}(b_{i-1}) = \mathcal{R}(b'_1) \cdots \mathcal{R}(b'_{i-1})$. For general $u, v, w \in C^*$, if $uv \le uw$, then $v \le w$. So after removing the common prefix from both sides of (4), obtain

  $$\mathcal{R}(b_i) \cdots \mathcal{R}(b_n) \le \mathcal{R}(b'_i) \cdots \mathcal{R}(b'_m).$$

  In such a scenario, we either have $\mathcal{R}(b_i) \le \mathcal{R}(b'_i)$ or $\mathcal{R}(b_i) \ge \mathcal{R}(b'_i)$. Since $\mathcal{R}$ is prefix-free (1), we obtain $b_i = b'_i$ in either case.

We can now use the inductively proven statement to show that $b_1 \cdots b_n \le b'_1 \cdots b'_m$ by case distinction:

- If $\min(n, m) = m$, i.e. $n \ge m$, then we can remove the common prefix $\mathcal{R}(b_1) \cdots \mathcal{R}(b_m) = R(b'_1) \cdots \mathcal{R}(b'_m)$ from both sides of (4) in order to obtain

  $$\mathcal{R}(b_{m+1}) \cdots \mathcal{R}(b_n) \le \varepsilon$$

  Since all $\mathcal{R}(b_i) \in A^+$ for all $1 \le i \le n$, we necessarily have $m = n$. This implies $b_1 \cdots b_n \le b'_1 \cdots b'_m$ (both sides are identical).
- If $\min(n, m) = n$, i.e. $n \le m$, then we directly have $b_1 \cdots b_n \le b'_1 \cdots b'_m$.

So in any case, $\mathcal{S}(c) = b_1 \cdots b_n \le b'_1 \cdots b'_m = \mathcal{S}(c')$. Using that $\mathcal{S}$ is prefix-free (1), we conclude $c = c'$, as desired. ◀

Proof of Theorem 6.4 item 1

We show that the systems $\alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$ and $\alpha_{\mathcal{S}}(\alpha_{\mathcal{R}}(\mathcal{M}))$ are even identical. Note that we have state sets:

$$Q^{\alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M})} \subseteq Q^{\mathcal{M}} \text{ and } Q^{\alpha_{\mathcal{R}}(\alpha_{\mathcal{S}}(\mathcal{M}))} \subseteq Q^{\alpha_{\mathcal{S}}(\mathcal{M})} \subseteq Q^{\mathcal{M}}$$

which are both subsets of $Q^{\mathcal{M}}$. Their initial states are identical, because they are both $q_0^{\mathcal{M}}$. We establish the isomorphism by simultaneously showing that the state sets match and that the transitions match:

($\subseteq$) Consider a transition

$$q \xrightarrow{c} q' \qquad \text{in } \alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$$

for which we already assume that $q \in Q^{\alpha_{\mathcal{R}}(\alpha_{\mathcal{S}}(\mathcal{M}))}$. Thus, we have

$$q \xRightarrow{(\mathcal{R}*\mathcal{S})(c)} q' \qquad \text{in } \mathcal{M}.$$

By the definition of $\mathcal{R}*\mathcal{S}$, we have $c \in \mathsf{dom}(\mathcal{S})$ and $b_1, \ldots, b_n \in \mathsf{dom}(\mathcal{R})$ with $\mathcal{S}(c) = b_1 \cdots b_n$, so above sequence can be rewritten as

$$q \xRightarrow{\mathcal{R}(b_1) \cdots \mathcal{R}(b_n)} q' \qquad \text{in } \mathcal{M}$$

or equivalently

$$q \xRightarrow{\mathcal{R}(b_1)} \cdots \xRightarrow{\mathcal{R}(b_n)} q' \qquad \text{in } \mathcal{M}$$

By definition of $\alpha_{\mathcal{R}}$, we have

$$q \xrightarrow{b_1} \cdots \xrightarrow{b_n} q' \qquad \text{in } \alpha_{\mathcal{R}}(\mathcal{M})$$

or equivalently

$$q \xRightarrow{\mathcal{S}(c)} q' \qquad \text{in } \alpha_{\mathcal{R}}(\mathcal{M}).$$

Finally, by the definition of $\alpha_{\mathcal{S}}$, this yields

$$q \xrightarrow{c} q' \qquad \text{in } \alpha_{\mathcal{S}}(\alpha_{\mathcal{R}}(\mathcal{M})).$$

This first shows that all states of $\alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$ are also contained in $\alpha_{\mathcal{R}}(\alpha_{\mathcal{S}}(\mathcal{M}))$ and secondly that the transitions are included, too.

($\supseteq$) The converse direction is analogous, starting with a transition

$$q \xrightarrow{c} q' \qquad \text{in } \alpha_{\mathcal{S}}(\alpha_{\mathcal{R}}(\mathcal{M}))$$

for which we know $q \in Q^{\alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M})}$ already. Thus, $c \in \mathsf{dom}(\mathcal{S})$ and we obtain

$$q \xRightarrow{\mathcal{S}(c)} q' \qquad \text{in } \alpha_{\mathcal{R}}(\mathcal{M}).$$

With $b_1 \cdots b_n = \mathcal{S}(c)$, we have

$$q \xrightarrow{b_1} \cdots \xrightarrow{b_n} q' \qquad \text{in } \alpha_{\mathcal{R}}(\mathcal{M}).$$

This implies that $b_i \in \mathsf{dom}(\mathcal{R})$ for every $1 \leq i \leq n$ and moreover

$$\left(q \xrightarrow{\mathcal{R}(b_1)} \cdots \xrightarrow{\mathcal{R}(b_n)} q'\right) = \left(q \xrightarrow{\mathcal{R}(b_1)\cdots\mathcal{R}(b_n)} q'\right) \qquad \text{in } \mathcal{M}.$$

Since all $b_i \in \mathsf{dom}(\mathcal{R})$ and $c \in \mathsf{dom}(\mathcal{S})$, we find that $c \in \mathsf{dom}(\mathcal{R} * \mathcal{S})$ and so $(\mathcal{R} * \mathcal{S})(c) = \mathcal{R}(b_1)\cdots\mathcal{R}(b_n)$ and

$$q \xrightarrow{(\mathcal{R} * \mathcal{S})(c)} q' \qquad \text{in } \mathcal{M}.$$

Finally, by the definition of $\alpha_{\mathcal{R}*\mathcal{S}}$, we conclude

$$q \xrightarrow{c} q' \qquad \text{in } \alpha_{\mathcal{R}*\mathcal{S}}(\mathcal{M}). \hfill \blacktriangleleft$$

Proof of Theorem 6.4 item 2

For the map-based action code $\mathcal{R} \colon B \rightharpoonup A^+$, define the partial map

$$\mathcal{R}^* \colon B^* \rightharpoonup A^* \qquad \mathcal{R}^*(\varepsilon) = \varepsilon \qquad \mathcal{R}^*(b\,w) = \mathcal{R}(b)\,\mathcal{R}^*(w) \quad \text{(if both defined)}.$$

By this, we mean that the inductive case $\mathcal{R}^*(b\,w)$ is only defined if both $\mathcal{R}(b)$ and $\mathcal{R}^*(w)$ are defined.[3] With this definition, we have that

$$(\mathcal{R} * \mathcal{S})(c) = \mathcal{R}^*(\mathcal{S}(c)) \text{ for all } c \in C.$$

For the isomorphism $h \colon \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})) \to \varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$, the involved state sets are by Definition 4.1 of the form:

$$Q^{\varrho_{\mathcal{S}}(\mathcal{M})} \subseteq Q^{\mathcal{M}} \times B^* \qquad Q^{\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))} \subseteq (Q^{\mathcal{M}} \times B^*) \times A^* \qquad Q^{\varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})} \subseteq Q^{\mathcal{M}} \times A^*$$

Define a partial map $h \colon Q^{\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))} \to Q^{\varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})}$ by

$$h((q, u), v) = \begin{cases} (q, \mathcal{R}^*(u)\,v) & \text{if } \mathcal{R}^*(u) \text{ is defined} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Such a partial map is sufficient to establish an isomorphism between the reachable parts (cf. Definition 2.8) of $\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$ and $\varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$, because we can show that if $((q, u), v)$ is reachable, then $\mathcal{R}^*(u)$ is defined: if $((q, u), v)$ is reachable, then the *shortest* path from the initial state must end with a path of the form:

$$((q, \varepsilon), \varepsilon) \xRightarrow{w} ((q, u), \varepsilon) \xRightarrow{v'} ((q, u), v) \qquad \text{in } \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})).$$

If we require that v' is the shortest path from $((q, u), \varepsilon)$ to $((q, u), v)$, then all transitions of v' must come from rule (1_ϱ), and so $v = v'$. If we require w to be the shortest path, then by an iterated application of Lemma B.3, we find that $w = \mathcal{R}^*(u)$.

In order to show that h is a simulation, consider a *reachable* transition

$$((q, u), v) \xrightarrow{a} ((q', u'), v') \qquad \text{in } \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})).$$

The transition being reachable implies that $\mathcal{R}^*(u)$ is defined. By Lemma B.3, there exists a transition

$$(q, u) \xrightarrow{b} (q'', u'') \qquad \text{in } \varrho_{\mathcal{S}}(\mathcal{M})$$

---

[3] $\mathcal{R}^*$ is also called the Kleisli extension of $\mathcal{R}$ for the monad $(-)^*$ on partial maps

with $\mathcal{R}(b) = var$ and some $r \in A^*$ such that

$$((q, u), \varepsilon) \overset{v}{\Longrightarrow} ((q, u), v) \overset{a}{\longrightarrow} ((q', u'), v') \overset{r}{\Longrightarrow} ((q'', u''), \varepsilon) \qquad \text{in } \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})).$$

Applying Lemma B.3 to the above b-transition in $\varrho_{\mathcal{S}}(\mathcal{M})$ provides us with some $c \in C$, $q''' \in Q^{\mathcal{M}}$, and $s \in B^*$ with $\mathcal{S}(c) = ubs$ such that

$$q \overset{c}{\longrightarrow} q''' \qquad \text{in } \mathcal{M}$$

and

$$(q, \varepsilon) \overset{u}{\Longrightarrow} (q, u) \overset{b}{\longrightarrow} (q'', u'') \overset{s}{\Longrightarrow} (q''', \varepsilon) \qquad \text{in } \varrho_{\mathcal{S}}(\mathcal{M}).$$

Since all involved states are reachable, $\mathcal{R}^*(u)$ and $\mathcal{R}^*(s)$ are defined. In total, we have that

$$(\mathcal{R} * \mathcal{S})(c) = \mathcal{R}^*(ubs) = \mathcal{R}^*(u)\,\mathcal{R}(b)\,\mathcal{R}^*(s) = \mathcal{R}^*(u)\,var\,\mathcal{R}^*(s)$$

and in particular

$$\mathcal{R}^*(u)\,va \leq (\mathcal{R} * \mathcal{S})(c).$$

Thus, the state

$$h((q, u), v) = (q, \mathcal{R}^*(u)\,v) \text{ in } \varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$$

has an a-transition to

$$h((q', u'), v') = (q', \mathcal{R}^*(u')\,v'),$$

as desired.

For the verification that h is a simulation in the converse direction, i.e. from $\varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$ to $\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$, consider a transition

$$(q, u) \overset{a}{\longrightarrow} (q', u') \qquad \text{in } \varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M}).$$

Again, using Lemma B.3, we obtain $c \in C$ with

$$(\mathcal{R} * \mathcal{S})(c) = uav \text{ and } q \overset{c}{\longrightarrow} q'' \text{ and } (q', u') \overset{v}{\Longrightarrow} (q'', \varepsilon).$$

By the definition of $\mathcal{R} * \mathcal{S}$, we thus obtain that $u \in A^*$ must be of the shape

$$\mathcal{R}^*(w)\,r = u \qquad \text{for some } w \in \mathsf{dom}(\mathcal{R})^*$$

By the definition of $\mathcal{R} * \mathcal{S}$, we have

$$w\,b \leq \mathcal{S}(c) \text{ with } u = \mathcal{R}^*(w)\,r \text{ and } ra \leq \mathcal{R}(b).$$

Then, $h((q, w), r) = (q, u)$ and we distinguish:

- If $ua = (\mathcal{R} * \mathcal{S})(c)$, then the above a-transition in $\varrho_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$ is produced by rule (2_ϱ), and we can use the same rule in $\varrho_{\mathcal{S}}(\mathcal{M})$ and $\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$ to establish the desired transition a-transition to

  $$((q, w), r) \overset{a}{\longrightarrow} ((q, \varepsilon), \varepsilon) \quad \text{in } \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})).$$

- If $ua \lneqq (\mathcal{R} * \mathcal{S})(c)$ but $ra = \mathcal{R}(b)$, then we use the rule (1_ϱ) in $\varrho_{\mathcal{S}}(\mathcal{M})$ and but rule (2_ϱ) in $\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$:

  $$((q, w), r) \overset{a}{\longrightarrow} ((q, w\,b), \varepsilon) \quad \text{in } \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})).$$

- If $ua \lneqq (\mathcal{R} * \mathcal{S})(c)$ and $ra \lneqq \mathcal{R}(b)$, then we use rules (1_ϱ) in both $\varrho_{\mathcal{S}}(\mathcal{M})$ and $\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$:

  $$((q, w), r) \overset{a}{\longrightarrow} ((q, w), r\,a) \quad \text{in } \varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M})).$$

Hence, in any of the above cases, we have a corresponding a-transition in $\varrho_{\mathcal{R}}(\varrho_{\mathcal{S}}(\mathcal{M}))$. ◀

Details for Remark 6.5

We fix $\mathcal{I}$ to be the identity relation for both codes, and hence omit $\mathcal{I}$ in the following. It is a standard result about Galois connections (and adjunctions in general) that they are compatible with composition: the right-adjoint of the composition of two functions is equal to the composition of the respective right-adjoints. One only needs to be warned that 'equal' here refers to the equality induced by the order $\sqsubseteq$, which means mutual simulation: For all action codes $\mathcal{R} \in \mathsf{Code}(A, B)$, $\mathcal{S} \in \mathsf{Code}(B, C)$ and $\mathcal{M} \in \mathsf{LTS}(C)$:

$$\gamma_{\mathcal{R}*\mathcal{S}}(\mathcal{M}) \sqsubseteq \gamma_{\mathcal{R}}(\gamma_{\mathcal{S}}(\mathcal{M})) \qquad \text{and} \qquad \gamma_{\mathcal{R}*\mathcal{S}}(\mathcal{M}) \sqsupseteq \gamma_{\mathcal{R}}(\gamma_{\mathcal{S}}(\mathcal{M})).$$

This is however weaker than the notion of isomorphism we consider (Definition 2.8). Concretely, we even have the following counterexample with $\gamma_{\mathcal{R}*\mathcal{S}}(\mathcal{M}) \not\cong \gamma_{\mathcal{R}}(\gamma_{\mathcal{S}}(\mathcal{M}))$. Consider sets $A = \{a\}$, $B = \{b\}$, $C = \{c\}$ and the action codes

$$\mathcal{R} \in \mathsf{Code}(A, B) \quad \mathcal{R}\colon B {\rightharpoonup} A^+ \quad b \mapsto a\,a$$
$$\mathcal{S} \in \mathsf{Code}(B, C) \quad \mathcal{S}\colon C {\rightharpoonup} B^+ \quad \text{undefined everywhere}$$

Start with a singleton system that has no transitions:

$$\mathcal{M} := \boxed{\rightarrow \enspace q_0} \qquad \text{in } \mathsf{LTS}(C)$$

For the empty $\mathcal{S}$, the concretization $\gamma_{\mathcal{S}}\colon \mathsf{LTS}(C) \to \mathsf{LTS}(B)$ sends this into the system

$$\gamma_{\mathcal{S}}(\mathcal{M}) = \boxed{\rightarrow \enspace q_0, \varepsilon \xrightarrow{\ b\ } \chi \circlearrowright b} \qquad \text{in } \mathsf{LTS}(B)$$

We have a b-transition from q_0 to χ because $\mathcal{S}(b)$ is undefined. For the next action code $\mathcal{R}$, the concretization $\gamma_{\mathcal{R}}\colon \mathsf{LTS}(B) \to \mathsf{LTS}(A)$ treats χ as an ordinary state, so it produces the following LTS, in which we have omitted the unreachable sink state introduced by $\gamma_{\mathcal{R}}$:

$$\gamma_{\mathcal{R}}(\gamma_{\mathcal{S}}(\mathcal{M})) \cong \boxed{\rightarrow \enspace (q_0, \varepsilon), \varepsilon \xrightarrow{\ a\ } (q_0, \varepsilon), a \xrightarrow{\ a\ } \chi, \varepsilon \underset{a}{\overset{a}{\rightleftarrows}} \chi, a} \quad \text{in } \mathsf{LTS}(A)$$

Here, we omitted the unreachable chaos state χ introduced by $\gamma_{\mathcal{R}}$, because the unreachable parts are not relevant for our notion of isomorphism $\cong$.

On the other hand, the composed action code $\mathcal{R}*\mathcal{S} \in \mathsf{Code}(A, C)$ is undefined everywhere, so analogously to $\gamma_{\mathcal{S}}$, concretization for $\mathcal{R} * \mathcal{S}$ sends above $\mathcal{M} \in \mathsf{LTS}(C)$ to

$$\gamma_{\mathcal{R}*\mathcal{S}}(\mathcal{M}) = \boxed{\rightarrow \enspace q_0, \varepsilon \xrightarrow{\ a\ } \chi \circlearrowright a} \qquad \text{in } \mathsf{LTS}(A).$$

Obviously, $\gamma_{\mathcal{R}*\mathcal{S}}(\mathcal{M})$ is not isomorphic to $\gamma_{\mathcal{R}}(\gamma_{\mathcal{S}}(\mathcal{M}))$, but there are canonical simulations in either direction, induced by the Galois connection between α and γ, using that α commutes with action code composition. ◀

Proof of Theorem 7.5

The notion of delay simulation equivalence that is used in the statement of the theorem can be formally defined as follows:

▶ **Definition B.5.** *Let* $\mathcal{M} = \langle Q, q_0, \longrightarrow \rangle \in \mathsf{LTS}(A)$ *be an LTS, where* A *is a set of labels that contains the* hidden action τ. *Let* $q \Longrightarrow q'$ *denote that there is finite sequence of states* $r_0, \ldots, r_n \in Q$ *such that* $r_0 = q$, $r_n = q'$ *and* $r_{i-1} \xrightarrow{\tau} r_i$ *for all* $1 \leq i \leq n$. *A relation* $S \subseteq Q \times Q$ *is called a* delay simulation *if it satisfies the following transfer property:*

- *If* $(q, r) \in S$ *and* $q \xrightarrow{a} q'$ *then either* $a = \tau$ *and* $q = q'$, *or* $\exists r', r''$ *such that* $r \Longrightarrow r' \xrightarrow{a} r''$ *and* $(q', r'') \in S$.

We write $q \sqsubseteq_d r$ *if there exists a weak delay simulation that relates* q *and* r. *We say that* q *and* r *are* delay simulation equivalent, *notation* $q \equiv_d r$, *if both* $q \sqsubseteq_d r$ *and* $r \sqsubseteq_d q$.

We describe the behavior of an implementation for $\mathcal{M}$ and an adaptor for $\mathcal{R}$ formally as expressions in Milner's Calculus of Communicating Systems (CCS) [28]. The semantics of CCS is defined in terms of an infinite LTS in which the states are CCS expressions, and the transitions between states are defined by structural operational semantics rules given in [28]. In the rest of this proof, we will assume that the reader is familiar with the CCS calculus. In our CCS expressions we use action names taken from I, O, X and Y, and without loss of generality we assume these four sets to be disjoint. Process $\mathsf{Impl}(\mathcal{M})$ describes the behavior of an implementation for $\mathcal{M}$ in which inputs and outputs are separated and occur sequentially. We define $\mathsf{Impl}(\mathcal{M})$ as the CCS expression $M(q_0^{\mathcal{M}})$, where for $q \in Q^{\mathcal{M}}$ and $i \in I$,

$$
M(q) = \sum_{i \in I} i \cdot M(q, i)
$$

$$
M(q, i) = \sum_{o \in O, q' \in Q^{\mathcal{M}} \mid q \xrightarrow{i/o} q'} \bar{o} \cdot M(q')
$$

Similarly, we introduce a process $\mathsf{Adaptor}(\mathcal{R})$ that describes the behavior of an adaptor for action code $\mathcal{R}$. Following the pseudocode of Algorithm 1, we define $\mathsf{Adaptor}(\mathcal{R})$ as the CCS expression $P(r_0)$, where for $r \in R$ and $x \in X$,

$$
P(r) = \sum_{x \in X} x \cdot Q(r, x)
$$

$$
Q(r, x) = \bar{i} \cdot R(r, x) \qquad \text{if } r \text{ is internal and } i \text{ winning for } x \text{ in } r
$$

$$
R(r, x) = \sum_{o \in O, r' \in R \mid \xrightarrow{i/o}_{\mathcal{M}} \wedge r \xrightarrow{i/o} r' \wedge i \text{ winning for } x \text{ in } r} o \cdot Q(r', x)
$$

$$
Q(r, x) = \overline{\pi_2(l(r))} \cdot P(r_0) \qquad \text{if } r \text{ is a leaf}
$$

Processes $\mathsf{Adaptor}(\mathcal{R})$ and $\mathsf{Impl}(\mathcal{M})$ may synchronize via actions taken from $I \cup O$. If we compose these processes using the CCS composition operator $|$, and apply the CCS restriction operator $\backslash$ to hide all communications, we obtain a CCS expression that describes the behavior of the parallel composition of the adaptor and the SUT. We claim that this composition is delay simulation equivalent to the expression $\mathsf{Impl}(\alpha_{\mathcal{R}}(\mathcal{M}))$ that describes the behavior of an implementation of $\alpha_{\mathcal{R}}(\mathcal{M})$:

$$
(\mathsf{Adaptor}(\mathcal{R}) \mid \mathsf{Impl}(\mathcal{M})) \backslash (I \cup O) \quad \equiv_d \quad \mathsf{Impl}(\alpha_{\mathcal{R}}(\mathcal{M})) \tag{5}
$$

Here we define $\mathsf{Impl}(\alpha_\mathcal{R}(\mathcal{M}))$ as the CCS expression $N(q_0^{\alpha(\mathcal{M})})$, where for $q \in Q^{\alpha(\mathcal{M})}$ and $x \in X$,

$$N(q) \;\; = \;\; \sum_{x \in X} x \cdot N(q,x)$$

$$N(q,x) \;\; = \;\; \sum_{y \in Y, q' \in Q^{\alpha(\mathcal{M})} \;\mid\; q \xrightarrow{\; x/y \;}_{\alpha(M)} q'} \bar{y} \cdot N(q')$$

Consider the following relation S between CCS expressions:

$$
\begin{aligned}
S \;\; = \;\; & \{((P(r_0) \mid M(q)) \setminus (I \cup O), \; N(q)) \;\mid\; q \in Q^{\alpha(\mathcal{M})}\} \\
& \cup \; \{((Q(r,x) \mid M(q')) \setminus (I \cup O), N(q,x)) \;\mid\; q \in Q^{\alpha(\mathcal{M})}, \; q' \in Q^\mathcal{M}, \\
& \qquad r \in R \text{ winning for } x \in X \;\wedge\; \exists \sigma \in (I \times O)^*: r_0 \xRightarrow{\sigma}_\mathcal{R} r \wedge q \xRightarrow{\sigma}_\mathcal{M} q'\} \\
& \cup \; \{((R(r,x) \mid M(q',i)) \setminus (I \cup O), \; N(q,x)) \;\mid\; q \in Q^{\alpha(M)}, \; q' \in Q^\mathcal{M}, \\
& \qquad r \in R \text{ winning for } x \in X \text{ with } i \in I \;\wedge\; \exists \sigma \in (I \times O)^*: r_0 \xRightarrow{\sigma}_\mathcal{R} r \wedge q \xRightarrow{\sigma}_\mathcal{M} q'\}
\end{aligned}
$$

We claim that S is a delay simulation relation. In order to prove this, we check that the transfer property holds for all pairs of related states and enabled transitions:

1. Assume $((P(r_0) \mid M(q)) \setminus (I \cup O), \; N(q)) \in S$ and $(P(r_0) \mid M(q)) \setminus (I \cup O) \xrightarrow{x} (Q(r_0,x) \mid M(q)) \setminus (I \cup O)$, for some $x \in X$. We observe that $N(q) \xrightarrow{x} N(q,x)$ and note that $((Q(r_0,x) \mid M(q)) \setminus (I \cup O), \; N(q,x)) \in S$ since r_0 is winning for x, $r_0 \xRightarrow{\;}_\mathcal{R} r_0$ and $q \xRightarrow{\sigma}_\mathcal{M} q$.

2. Assume $((Q(r,x) \mid M(q')) \setminus (I \cup O), \; N(q,x)) \in S$ and $(Q(r,x) \mid M(q')) \setminus (I \cup O) \xrightarrow{\tau} (R(r,x) \mid M(q',i)) \setminus (I \cup O)$, for r internal and i the unique input that is winning for x in r. By the assumption, $q \in Q^{\alpha(M)}$, $q' \in Q^\mathcal{M}$, and there exists $\sigma \in (I \times O)^*$ such that $r_0 \xRightarrow{\sigma}_\mathcal{R} r$ and $q \xRightarrow{\sigma}_\mathcal{M} q'$. Then $((R(r,x) \mid M(q',i)) \setminus (I \cup O), \; N(q,x)) \in S$, as required.

3. Assume $((R(r,x) \mid M(q',i)) \setminus (I \cup O), \; N(q,x)) \in S$ and $(R(r,x) \mid M(q',i)) \setminus (I \cup O) \xrightarrow{\tau} (Q(r',x) \mid M(q'')) \setminus (I \cup O)$, with $r \xrightarrow{i/o} r'$ and $q' \xrightarrow{i/o} q''$. By the assumption, $q \in Q^{\alpha(M)}$, $q' \in Q^\mathcal{M}$, r is winning for x with i, and there exists $\sigma \in (I \times O)^*$ such that $r_0 \xRightarrow{\sigma}_\mathcal{R} r$ and $q \xRightarrow{\sigma}_\mathcal{M} q'$. Then $q'' \in Q^\mathcal{M}$ and, by definition of winning, r' is winning for x. Moreover, if we take $\sigma' = \sigma \cdot (i,o)$, then $r_0 \xRightarrow{\sigma'}_\mathcal{R} r'$ and $q \xRightarrow{\sigma'}_\mathcal{M} q''$. This implies that $((Q(r',x) \mid M(q'')) \setminus (I \cup O), \; N(q,x)) \in S$, as required.

4. Assume $((Q(r,x) \mid M(q')) \setminus (I \cup O), \; N(q,x)) \in S$ and $(Q(r,x) \mid M(q')) \setminus (I \cup O) \xrightarrow{\overline{\pi_2(l(r))}} (P(r_0) \mid M(q')) \setminus (I \cup O)$, for r a leaf. By the assumption, $q \in Q^{\alpha(\mathcal{M})}$, $q' \in Q^\mathcal{M}$, r is winning for x, and there exists $\sigma \in (I \times O)^*$ such that $r_0 \xRightarrow{\sigma}_\mathcal{R} r$ and $q \xRightarrow{\sigma}_\mathcal{M} q'$. By definition of the contraction operator, $q \xrightarrow{l(r)}_{\alpha(\mathcal{M})} q'$ and $q' \in Q^{\alpha(\mathcal{M})}$. But this means $N(q,x) \xrightarrow{\overline{\pi_2(l(r))}} N(q')$. Now observe that $((P(r_0) \mid M(q')) \setminus (I \cup O), \; N(q')) \in S$, as required.

Next consider the following relation T between CCS expressions:

$$
\begin{aligned}
T \;\; = \;\; & \{(N(q), \; (P(r_0) \mid M(q)) \setminus (I \cup O)) \;\mid\; q \in Q^{\alpha(\mathcal{M})}\} \\
& \cup \; \{(N(q,x), \; (Q(r_0,x) \mid M(q)) \setminus (I \cup O)) \;\mid\; q \in Q^{\alpha(\mathcal{M})}\}
\end{aligned}
$$

We claim that T is a delay simulation relation, and check that the transfer property holds for all pairs of related states and enabled transitions:

1. Assume $(N(q), (P(r_0) \mid M(q)) \setminus (I \cup O)) \in T$ and $N(q) \xrightarrow{x} N(q, x)$, for some $x \in X$. We observe that $(P(r_0) \mid M(q)) \setminus (I \cup O) \xrightarrow{x} (Q(r_0, x) \mid M(q)) \setminus (I \cup O)$ and note that $(N(q, x), (Q(r_0, x) \mid M(q)) \setminus (I \cup O)) \in T$.

2. Assume $(N(q, x), (Q(r_0, x) \mid M(q)) \setminus (I \cup O)) \in T$ and $N(q, x) \xrightarrow{\bar{y}} N(q')$. Then $\alpha(\mathcal{M})$ has a transition $q \xrightarrow{x/y} q'$ and $q' \in Q^{\alpha(\mathcal{M})}$. By definition of $\alpha(\mathcal{M})$, $\mathcal{R}$ has a leaf r with $l(r) = (x, y)$ and there exists a sequence σ such that $r_0 \xRightarrow{\sigma}_{\mathcal{R}} r$ and $q \xRightarrow{\sigma}_{\mathcal{M}} q'$. Let

$$\sigma \;=\; (i_1, o_1)(i_2, o_2) \cdots (i_n, o_n)$$

Then $\mathcal{R}$ has states $r_1, \dots, r_n$ and $\mathcal{M}$ has states $s_0, \dots, s_n$ such that:

$$r_0 \xrightarrow{i_1/o_1}_{\mathcal{R}} r_1 \xrightarrow{i_2/o_2}_{\mathcal{R}} r_2 \cdots \xrightarrow{i_n/o_n}_{\mathcal{R}} r_n$$

$$q = s_0 \xrightarrow{i_1/o_1}_{\mathcal{M}} s_1 \xrightarrow{i_2/o_2}_{\mathcal{M}} s_2 \cdots \xrightarrow{i_n/o_n}_{\mathcal{M}} s_n = q'$$

From these runs in $\mathcal{R}$ and $\mathcal{M}$ we may construct a sequence of τ-transitions:

$$
\begin{aligned}
(Q(r_0, x) \mid M(s_0)) \setminus (I \cup O) \;\;& \xrightarrow{\tau} \;\; (R(r_0, x) \mid M(s_0, i_1)) \setminus (I \cup O) \\
& \xrightarrow{\tau} \;\; (Q(r_1, x) \mid M(s_1)) \setminus (I \cup O) \\
& \;\;\vdots \\
& \xrightarrow{\tau} \;\; (R(r_{n-1}, x) \mid M(s_{n-1}, i_n)) \setminus (I \cup O) \\
& \xrightarrow{\tau} \;\; (Q(r_n, x) \mid M(s_n)) \setminus (I \cup O)
\end{aligned}
$$

Note that our assumptions that $\mathcal{R}$ is determinate and has a winning strategy for every input $x \in X$ impy that the inputs that occur in σ are always winning. From the above sequence of τ-transitions we may conclude

$$(Q(r_0, x) \mid M(q)) \setminus (I \cup O) \;\; \Longrightarrow \;\; (Q(r, x) \mid M(q')) \setminus (I \cup O)$$

Since $(Q(r, x) \mid M(q')) \setminus (I \cup O) \xrightarrow{\bar{y}} (P(r_0) \mid M(q')) \setminus (I \cup O)$ and $(N(q'), (P(r_0) \mid M(q')) \setminus (I \cup O)) \in T$, the transfer property follows.

Because S is a delay simulation from $(\mathsf{Adaptor}(\mathcal{R}) \mid \mathsf{Impl}(\mathcal{M})) \setminus (I \cup O)$ to $\mathsf{Impl}(\alpha_{\mathcal{R}}(\mathcal{M}))$, and T is a delay simulation from $\mathsf{Impl}(\alpha_{\mathcal{R}}(\mathcal{M}))$ to $(\mathsf{Adaptor}(\mathcal{R}) \mid \mathsf{Impl}(\mathcal{M})) \setminus (I \cup O)$, identity (5) follows, and thereby the theorem. ◀

Proof of Proposition 7.7

Assume $\mathcal{M}$ is output deterministic and $\mathcal{R}$ is determinate. Let $\mathcal{N} = \alpha_{\mathcal{R}}(\mathcal{M})$. Suppose that $\mathcal{N}$ has transitions $q \xrightarrow{x/y'}_{\mathcal{N}} q'$ and $q \xrightarrow{x/y''}_{\mathcal{N}} q''$. We need to show $y' = y''$ and $q' = q''$. The transitions have been derived using rule (2_α), and formulated for tree-shaped action codes $\mathcal{R}$, we know there are generalized transitions

$$
\begin{aligned}
q \xRightarrow{u'/s'}_{\mathcal{M}} q' \quad & r_0 \xRightarrow{u'/s'}_{\mathcal{R}} r' \quad r' \in L \;\; l(r') = (x, y') \\
q \xRightarrow{u''/s''}_{\mathcal{M}} q'' \quad & r_0 \xRightarrow{u''/s''}_{\mathcal{R}} r'' \quad r'' \in L \;\; l(r'') = (x, y'')
\end{aligned}
$$

Now since $\mathcal{R}$ is determinate, the first inputs in u' and u'' must be identical. But since $\mathcal{M}$ is output deterministic, this implies that the first outputs in s' and s'' must also be identical. Moreover, the paths from q to q' and q'' in $\mathcal{M}$ share the same initial transition. Since action codes are deterministic, the paths from r_0 to r' and r'' in $\mathcal{R}$ also share the same initial transition. By repeating this line of reasoning, we can "zip together" the paths from q to q' and q'' in $\mathcal{M}$, and the paths from r_0 to r' and r'' in $\mathcal{R}$, and obtain $u' = u''$, $s' = s''$, $q' = q''$, $r' = r''$ and $y' = y''$, as required. ◀