Minimization of Streaming Transducers

Christian Bianchini
Università degli Studi di Udine, Italy

Gabriele Puppis
Università degli Studi di Udine, Italy

Abstract

We provide general criteria for the existence of minimal models of streaming transducers, namely devices that read an input word and produce an output value by iteratively updating an internal memory. This abstract model subsumes classical (sub)sequential transducers (Schützenberger), streaming string-to-string transducers (Alur–Černý), polynomial automata (Benedikt et al.), and variants of streaming string-to-tree transducers (Alur–D'Antoni). We then instantiate these criteria to obtain effective minimization results for variants of the latter model, where outputs are terms constructed incrementally by extending (tuples of) terms either at the leaves or at the roots.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases Streaming transducers, Minimization

Digital Object Identifier 10.4230/LIPIcs.LICS.2026.27

Related Version *Full Version*: `http://arxiv.org/abs/2605.11190`

Funding This work was partially supported by INdAM - GNCS Project (CUP: E53C25002010001).

1 Introduction

We study the existence of minimal models of streaming transducers. By streaming transducer we generically mean a device that consumes an input word and deterministically produces a corresponding output of a certain type by updating data in its internal memory. Examples of such models are the sequential transducers [61], the streaming string-to-string transducers [2], the polynomial automata [16], and the streaming string-to-tree transducers [3, 18].

A classical notion of memory that is often used, especially in the previously mentioned models, consists of a finite number of registers that can be assigned numbers, strings, terms, etc. Registers can be manipulated using specific operations, that we call updates (e.g. sums and products of numbers, concatenations of strings, substitutions of terms). An important aspect is that *the memory of our transducer model is "write-only"*: it serves exclusively as a mechanism for generating outputs and is separated from the machine's control flow.

We will assume that the set of updates available for manipulating the internal memory of a transducer is closed under composition; in particular, updates will naturally form a monoid structure. In this sense, our model of transducer is also similar to the *monoidal transducer* of [8]. The main difference is that in a monoidal transducer the data domain itself is required to be a monoid, and the outputs are produced by aggregating, via the monoid product, the values emitted by the transitions, whereas in our model only the updates carry a monoid structure. Another minor difference is that our data is typed. More generally, one can see monoidal transducers as a syntactic fragment of streaming transducers.

The main goal of the paper is to establish general criteria for the existence of minimal streaming transducers, in the same spirit of the celebrated Myhill-Nerode theorem [43], and then apply these criteria to obtain new minimization results for concrete examples of transducer classes. Drawing inspiration from a categorical approach to automata minimization [41], and from more recent work that views automata as functors [26, 27], we adopt an *algebraic notion of minimality* for transducers. Informally, a transducer A is algebraically


© Gabriele Puppis and Christian Bianchini;
licensed under Creative Commons License CC-BY 4.0

minimal if, for every equivalent transducer B, A can be obtained as a quotient of a subobject of B. The "quotient-of-a-subobject" relation (subquotient for short) is formalized in terms of suitable pairs of functions that transform the configuration space of a transducer and that generalize surjective functions (for inducing quotients) and injective functions (for inducing subobjects). In category theory, such functions are called epis and monos, and we shall adopt the same terminology here. However, we will keep category-theoretic jargon to a minimum and use it only when it provides a clear and convenient shorthand.

The mere existence of algebraically minimal transducers is quite significant: a minimal transducer, being a subquotient of all equivalent transducers, inherits from those objects any property that is closed under taking quotients and subobjects (cf. Eilenberg's theory of varieties/pseudovarieties [34, 35]). For example, it often happens that classes of transductions definable by logical formalisms (notably, the class of first-order definable transductions) are characterized in terms of forbidden patterns at the structural level of transducers (e.g., aperiodicity of the transformation semigroup). In these cases, the properties that characterize definability in logical formalisms are naturally closed under quotients and subobjects. In particular, if some equivalent transducer avoids a forbidden pattern (e.g. a non-trivial permutation of the configuration space), then so does a minimal transducer; conversely, any obstruction present in a minimal transducer also appears in every equivalent transducer. When algebraically minimal transducers can be computed, these characterizations become effective: membership in a subquotient-closed fragment can be decided by inspecting the structure of a minimal transducer. We refer the reader to [60, 38, 25, 37, 59, 22, 51, 28] for examples of applications of these principles in the theory of automata and transducers.

Another important application of minimization is automata learning: in active learning settings, the availability of canonical/minimal representatives is used to maintain compact hypotheses and justify correctness of the learned model up to equivalence. See, for instance, [6, 56, 64, 63, 33, 19, 46, 23, 49, 27, 8].

The results in the first part of the paper give sufficient and necessary conditions for the existence of algebraically minimal transducers, and can be summarized as follows:

▶ **Theorem 1 (rephrasing of Theorems 22–24).** *A class of transducers admits algebraically minimal models if and only if*[1] *the underlying data structure admits constrained domains and greatest common divisors for every non-empty set of updates. In addition, if every constrained domain is also finitary, then algebraically minimal models also exist within the subclass of finite transducers.*

In the second part of the paper, we give examples of applications of the above theorem for minimizing streaming transducers that produce terms as outputs. More precisely, the outputs of these transducers are constructed incrementally while reading the input, by extending (tuples of) terms either at the leaves (called downward string-to-term transducers) or at the roots (called upward string-to-term transducers). For both models, minimization is effective, meaning that, given a finite transducer, one can compute a minimal one equivalent to it. To the best of our knowledge, these minimization results are new, and can be seen as non-trivial generalizations of Choffrut's minimization of sequential transducers [24], one of the few core minimization theorems for transducers. Even if some proofs are technically difficult, the most interesting part of this contribution lies in a connection between minimization

[1] The only-if direction actually relies on mild assumptions on the sets of data and updates available to the transducers (cf. the definition of standard data structure preceding Theorem 24). These assumptions are satisfied by essentially all data structures underlying the models considered in the literature.

of string-to-term transducers, the theory of unifiers of Robinson, Martelli and Montanari [57, 48], and the theory of anti-unifiers of Plotkin and Reynolds [53, 54]. Specifically, the former theory is used to compute a subobject of an upward string-to-term transducer, while the latter theory is used to compute a quotient of a downward string-to-term transducer.

Related work. There are several ways to define and construct a minimal transducer. The two most prominent approaches in the literature either minimize concrete resources, such as the number of control states and/or registers (cf. [5, 32, 10, 14, 15]), or even the amount of non-determinism (cf. [12, 13]), or adopt an algebraic notion of minimality, such as the one discussed above, expressed through a universal subquotient property relative to all equivalent objects (cf. [41, 40, 26, 27, 8]). Unlike resource minimization, whose goal is to optimize a chosen concrete measure of an implementation, algebraic minimality aims at identifying a canonical representative of an equivalence class.

In some settings, however, the two approaches coincide: an object that is minimal in the algebraic sense is also one that uses the fewest concrete resources. The paradigmatic example is finite-state automata minimization, but similar phenomena also occur for automata enhanced with registers (cf. [17, 20, 49, 9]). In these cases, the automaton with the fewest states is unique and can be obtained by first pruning unreachable states, thereby passing to a subobject, and then merging states according to the coarsest equivalence compatible with the recognized language. The resulting automaton is therefore a quotient of every other equivalent pruned automaton, and satisfies precisely the universal subquotient property.

For transducers, the relationship between algebraic minimality and resource minimization is more subtle. On the one hand, algebraically minimal transducers do have a minimum number of control states among all equivalent transducers (this follows from standard arguments). On the other hand, they are not necessarily optimal with respect to the number of registers used. In fact, it is even difficult to define a uniform notion of register that applies to all generic models of transducers. When such a notion exists, the best one can generally expect is to minimize locally, subject to the already minimized control structure, the number of registers available at each control state that are not fixed by a single assignment. We will see in Section 6.2 an example of this phenomenon for the class of upward string-to-term transducers.

The algebraic minimization approach is also closely related to the classical linear-algebraic theory of weighted automata, going back at least to the realization results of Carlyle and Paz [21] and to the minimization theorem of Fliess [39]. Indeed, a weighted automaton can be transformed into an equivalent cost register automaton (CRA) with a single control state and linear register updates, where the states of the weighted automaton become the registers of the one-state CRA. Thus, minimizing states in the former amounts to minimizing registers in the latter. From our perspective, Fliess' minimization can be seen as a concrete instance, in a linear-algebraic setting, of the same "subobject followed by quotient" approach underlying algebraic minimization: one first restricts to the smallest subspace generated by reachable values, corresponding to a subobject, and then quotients by the equivalence induced by the remaining linear observations.

Organization of the paper. Section 2 fixes basic notation for functions and terms. Section 3 introduces the category-theoretic concepts underlying algebraic minimality and the existence of minimal objects. Section 4 defines the abstract model of streaming transducer, while Section 5 proves minimization results under simple assumptions. Section 6 gives effective minimization results for two concrete classes of transducers, namely downward string-to-term transducers and upward string-to-term transducers. Finally, Section 7 concludes and discusses a few research directions.

Due to space constraints, most proofs are only sketched or omitted; full details are available in the extended version at `http://arxiv.org/abs/2605.11190`. For convenience, technical terms and notations in the electronic version of this manuscript are hyper-linked to their definitions (cf. `https://ctan.org/pkg/knowledge`).

2 Preliminaries

Functions. Given a function f, we denote by $\mathrm{Dom}(f)$, $\mathrm{Cod}(f)$, and $\mathrm{Rng}(f)$, respectively, its *domain*, *codomain*, and *range*[2]. In particular, we have $\mathrm{Rng}(f) \subseteq \mathrm{Cod}(f)$ and this inclusion might be strict. *We assume that the domain A and codomain B are always part of the definition of a function, and they are usually specified within the notation $f : A \to B$.* We use the same notation for a partial function $f : A \to B$, with a bit of care for its domain: we distinguish the *source set* A from the *domain of definition* of f, defined as the subset $\mathrm{Dom}(f)$ of A that contains the elements x where $f(x)$ is defined. When equating two partial functions, like in $f = g$, we require that (1) f and g have the same source, domain of definition, and codomain, and (2) for all $x \in \mathrm{Dom}(f)$, $f(x) = g(x)$.

Given two (partial) functions $f : A \to B$ and $g : B \to C$, we denote by $f;g$ their *composition*, interpreted according to the *diagrammatic order*[3], namely, $f;g : A \to C$ is defined on those x's such that $x \in \mathrm{Dom}(f)$ and $f(x) \in \mathrm{Dom}(g)$, and maps any such x to $g(f(x)) \in C$. When we write a composition $f;g$ we always tacitly assume that the codomain of f is the same as the source of g; however, $\mathrm{Rng}(f)$ may be strictly contained in the source of g, and $\mathrm{Dom}(g)$ may be strictly contained in $\mathrm{Cod}(f)$. A *factorization* of a (partial) function $h : A \to B$ is a pair of (partial) functions $f : A \to C$ and $g : C \to B$ such that $h = f;g$.

Terms. A *term* is a non-empty, finite tree, labelled over a ranked alphabet Γ, where the number of children of each node coincides with the rank of its label. An example is the term $a(b(c), c)$, provided that $a, b, c \in \Gamma$ have ranks $2, 1, 0$, respectively. We shall also work with terms over an alphabet expanded with variables, say $\Gamma \uplus X$, with $X = \{x_1, x_2, \ldots, y, \ldots\}$, where the variables are assumed to have rank 0, and thus necessarily occur at the leaves.

We mostly manipulate terms using (*first-order*) *substitutions*: given a term t over an n-tuple of variables $\bar{x} = (x_1, \ldots, x_n)$ and given an n-tuple of terms $\bar{u} = (u_1, \ldots, u_n)$ (possibly over a different tuple of variables), we denote by $t[\bar{x}/\bar{u}]$ the result of substituting in t every occurrence of variable x_i, simultaneously for all $i = 1, \ldots, n$, with the corresponding term u_i. For example, if $t = a(x_1, x_2, x_1)$, $\bar{u} = (b, c(x_3))$, then $t[\bar{x}/\bar{u}] = a(b, c(x_3), b)$.

3 Minimality via category theory

We focus on an algebraic notion of minimality for transducers, characterized by a universal property formalized in terms of quotient and subobject relations. Although these relations can be tailored to specific models of transducers, such definitions tend to obscure the key properties behind model-dependent details. Since we aim to establish minimization results for several classes of transducers, we work at a higher level of abstraction, where quotients and subobjects can be defined once, uniformly and independently of the model. For this, we

[2] We prefer to not use the term 'image' since this is reserved for another concept, to be introduced later.
[3] We are aware that functional composition is often written with $\circ$ and in the opposite order; we believe that the adopted notation eases readability, especially here since we often deal with long sequences of compositions.

use a simplified category-theoretic framework. We call *arrow* any member of a restricted class of functions or partial functions, assumed to be closed under composition and to contain an *identity* on each set. The sets forming the domains and codomains of these (partial) functions may carry additional structure, and are generically called *objects*; accordingly, an arrow can be seen as a structure-preserving (partial) function. Examples of arrows are the transformations within a transducer's configuration space, representing transition functions, and the transformations between transducers themselves, representing transducer morphisms. A *category* is precisely a class of objects connected by arrows.

All definitions in this section are understood relative to a fixed category, which we keep implicit whenever no ambiguity arises.

Epis and monos. We begin by discussing special types of arrows, called epis and monos, which capture quotients and subobjects, respectively, and generalize surjective and injective maps between sets. Such generalizations are common in category theory and come in several strengths. For instance, plain epis lie at the bottom of the standard hierarchy

$$\text{epis} \supseteq \text{extremal epis} \supseteq \text{strong epis} \supseteq \text{strict epis} \supseteq \text{regular epis}.$$

In this work, we mostly use epis and strong monos, a choice tailored to our minimization results. In the category of unrestricted functions between sets, the hierarchies of epis and monos collapse to the usual surjective and injective functions. In more structured categories, where arrows are restricted functions —as happens for the transition functions of our transducers— these notions may diverge. For example, in the category of polynomial maps over algebraic varieties, the function $f : \mathbb{R}^2 \to \mathbb{R}^2$ defined by $f(x, y) = (x, xy)$ is epi but not surjective, since $(0, 1) \notin \mathrm{Rng}(f)$.

Below are the definitions of the important types of arrows that we are going to use:

- An arrow $f : A \to B$ is an *epi*, denoted $A \overset{f}{\twoheadrightarrow} B$, if for all arrows $h, h' : B \to C$, $f;h = f;h'$ implies $h = h'$.
- An arrow $g : B \to C$ is a *mono*, denoted $B \overset{g}{\rightarrowtail} C$, if for all arrows $h, h' : A \to B$, $h;g = h';g$ implies $h = h'$.
- An arrow $i : A \to B$ is an *isomorphism*, denoted $A \overset{i}{\simeq} B$, if there is an arrow $i^{-1} : B \to A$, called *inverse* of i, such that $i;i^{-1}$ and $i^{-1};i$ are identities on A and B, respectively. We write $A \simeq B$, without the annotation i, to state the existence of such an isomorphism between A and B. In a similar way, by thinking of arrows as objects in a new category (the arrow category), we define an *isomorphism* between two arrows $f : A \to B$ and $f' : A' \to B'$ as a pair of object isomorphisms $i : A \to A'$ and $j : B \to B'$ such that $f;j = i;f'$. We denote this by $f \overset{i,j}{\simeq} f'$, or simply by $f \simeq f'$.
- An arrow $g : C \to D$ is a *strong mono*, denoted $C \overset{g}{\rightarrowtail\!\!\!\!\!\!\rightarrow} D$, if for all epi $f : A \twoheadrightarrow B$ and arrows $h : A \to C$ and $h' : B \to D$, if $f;h' = h;g$, then there is a unique arrow $d : B \to C$ such that $h = f;d$ and $h' = d;g$. In other words, if the left diagram commutes, then also the right diagram commutes for some unique d:

The above condition is called *diagonal fill-in property*, and is crucial for reasoning with diagrams involving epis and strong monos. Two standard consequences will be used

repeatedly. First, every strong mono is a mono: indeed, by taking f to be an identity and by considering two arrows $h, h'' : A \to C$ such that $h;g = h'';g'$, we get $h = d = h'$. Second, every arrow that is both epi and strong mono is an object isomorphism, obtained by applying the same diagonal fill-in property to the square where h and h' are identities. It is easy to see that the classes of epis, monos, isomorphisms, and strong monos are all closed under composition.

Factorization systems. A factorization $h = f;g$ is called *strong* if f is an epi and g is a strong mono, and in this case the intermediate object $C = \mathrm{Cod}(f)$ is called the *image* of h. The uniqueness of the image of h follows directly from the diagonal fill-in property:

▶ **Lemma 2 ([1, Propositions 14.4]).** *The image of an arrow, if a strong factorization exists, is unique up to isomorphism.*

An object I is *initial* if for every object A, there is a unique arrow $!^A : I \to A$, called the *initial arrow* for A. Symmetrically, an object F is *final* if for every object A, there is a unique arrow $!_A : A \to F$, called the *final arrow* for A.

▶ **Lemma 3 ([1, Propositions 7.3, 7.6]).** *Initial (resp. final) objects, if exist in the considered category, are unique up to isomorphism.*

Below, we define algebraic minimality and we outline some general conditions for the existence of algebraically minimal objects, based on a reworking of [26].

- A is a *quotient* of B if $B \twoheadrightarrow A$, i.e. there is an epi from B to A,
- A is a *subobject* of B if $A \rightarrowtail B$, i.e. there is a strong mono from A to B
- A is a *subquotient* of B if $A \twoheadleftarrow C \rightarrowtail B$ for some C, i.e. A is a quotient of some subobject C of B,
- A is *algebraically minimal* if it is a subquotient of every object.

Next, assume that the category under consideration admits:

1. a strong factorization of every arrow, together with the corresponding image,
2. an initial object I,
3. an object M that is final in the *sub-category* consisting of the images of the initial arrows (*initial images* for short).

Recall that, by Lemmas 2 and 3, the above objects are unique up to isomorphism. We also remark that the above assumptions differ slightly w.r.t. the presentation of [26], specifically because we do not require the existence of a final object in the full category, but rather in the sub-category of initial images. The reason for this difference is that it simplifies the construction of the minimal object, especially for the case of transducers.

For every object A, we can then define:

- $\mathrm{Reach}(A)$ as the image of the initial arrow $I \to A$,
- $\mathrm{Obs}(\mathrm{Reach}(A))$ as the image of the final arrow $\mathrm{Reach}(A) \to M$ in the sub-category of initial images.

Note that the initial object I is isomorphic to its image $\mathrm{Reach}(I)$, since, by definition, there is a unique arrow from I to itself, and this is both the identity and the composition $I \twoheadrightarrow \mathrm{Reach}(I) \rightarrowtail I$.

The notations $\mathrm{Reach}(A)$ and $\mathrm{Obs}(\mathrm{Reach}(A))$ are inspired by the operations performed to minimize deterministic finite automata. Specifically, $\mathrm{Reach}(A)$ reflects the idea that constructing a strong factorization of the initial arrow $I \to A$ amounts to pruning A by retaining only its reachable states. Likewise, $\mathrm{Obs}(\mathrm{Reach}(A))$ is suggestive of the fact that the final arrow $\mathrm{Reach}(A) \to M$ induces an observational equivalence that groups reachable states inducing the same residual languages.

▶ **Proposition 4 (variation of [26, Lemma 3.5]).** *Under the previous assumptions that enable the constructions* Reach(−) *and* Obs(Reach(−))*, we have that*

1. $I \simeq \mathrm{Reach}(I)$,
2. $M \simeq \mathrm{Obs}(I)$,
3. $M \simeq \mathrm{Obs}(\mathrm{Reach}(A))$ *for every object* A.

In particular, $\mathrm{Obs}(\mathrm{Reach}(A))$ *is a subquotient of* A*, and because* $M \simeq \mathrm{Obs}(\mathrm{Reach}(A))$*,* M *is a subquotient of every* A*, hence an algebraically minimal object.*

We conclude by observing that, in general, unless the category is particularly well-behaved, the subquotient relation does not need to be transitive or anti-symmetric. In particular, there may exist non-isomorphic, minimal objects that are one a subquotient of another. However, if the objects are of the form $\mathrm{Obs}(\mathrm{Reach}(A))$ and $\mathrm{Obs}(\mathrm{Reach}(A'))$, then they are necessarily also isomorphic.

4 The transducer model

Our transducers can store data from an arbitrary, fixed universe (e.g. real numbers, words, or trees), and can manipulate it via a basic set of operations, called updates (e.g. sums and products of numbers, concatenations of strings, term substitutions).

4.1 Data structures

We define a *data structure* as an algebra $\mathbb{D}$ of typed data. In most cases, the reader can assume *types* to be natural numbers, although later we will introduce more abstract types — even with this extension, we shall always assume that the set of types is countable. The *domain* of a type τ, denoted $\mathbb{D}_\tau$, is the subset of $\mathbb{D}$ consisting of all data of type τ. For example, if τ is a number, then its domain may include τ-tuples of terms. The data structure is also equipped with a set of operations, called *updates*, and this set is assumed to be closed under composition and to contain identities on each domain. Each update is associated with an input type α and an output type β, and accordingly maps any data in $\mathbb{D}_\alpha$ to a data in $\mathbb{D}_\beta$. In particular, the domain of definition of an update coincides with the domain of its input type. We denote by $\mathbb{D}_{\alpha\to\beta}$ the set of updates with input type α and output type β.

The following examples of data structures with numeric types will be used later:

- ***Free term algebra.*** The domain associated with each type τ contains τ-tuples of *ground terms*, namely, terms without variables. Note that there is only one data of type $\tau = 0$, namely, the empty tuple $()$. An update in $\mathbb{D}_{\alpha\to\beta}$ is specified by a β-tuple of terms $\bar{c} = (c_1, \dots, c_\beta)$ over an α-tuple of variables $\bar{x} = (x_1, \dots, x_\alpha)$, and maps any α-tuple of ground terms $\bar{t} = (t_1, \dots, t_\alpha)$ to the β-tuple $\big(c_1[\bar{x}/\bar{t}], \dots, c_\beta[\bar{x}/\bar{t}]\big)$. Intuitively, such an update extends the input terms $t_1, \dots, t_\alpha$ upward, at their roots. One can further restrict the types of updates in $\mathbb{D}_{\alpha\to\beta}$ by requiring that the specifications $\bar{c}$ satisfy any combination of the following conditions: (1) every variable x_i occurs *at most once* in $\bar{c}$ (*copyless updates*, as opposed to *copyful updates*); (2) every variable x_i occurs *at least once* in $\bar{c}$ (*non-erasing updates*); (3) every variable x_i occurs *exactly once* in $\bar{c}$ and when the variables of $\bar{c}$ are listed from left to right they appear in the precise order $x_1, \dots, x_\alpha$ (*linear updates*).
- ***Leaf substitution algebra.*** The domain of each type τ contains terms with exactly τ variable occurrences, without repetitions and in a fixed left-to-right order $x_1, \dots, x_\tau$. Since variable names are immaterial in this setting, we replace them by placeholders and write the substitution simply as $t[\bar{u}]$. We call these objects *linear terms*. An update in $\mathbb{D}_{\alpha\to\beta}$ is

specified by an α-tuple $\bar{u} = (u_1, \ldots, u_\alpha)$ of linear terms with precisely β placeholders, and maps every linear term t of type α to $t[\bar{u}]$ of type β. Intuitively, such an update extends t downward, at the leaves.

- ***Polynomial register algebra.*** The domain of each type τ consists of τ-tuples of elements from a given field, such as $\mathbb{Q}$, or more generally from a polynomial ring, such as $\mathbb{Q}[z]$. The updates are *polynomial maps* of the form $(x_1, \ldots, x_\alpha) \mapsto (p_1(\bar{x}), \ldots, p_\beta(\bar{x}))$, where each p_i is a polynomial in $\bar{x} = (x_1, \ldots, x_\alpha)$. As before, one may restrict updates, for example, to affine or linear maps.
- ***String register algebra.*** Here the data consists of tuples of strings, whose types are again their arities. The updates are generated by atomic operations that prepend or append a letter to a component, concatenate two components, insert a new empty component, duplicate, delete, or swap components. As with terms, one can restrict these updates, for instance by forbidding duplication (copyless updates) and/or deletion (non-erasing updates) [2, 50].

The free term algebra is particularly important for us, because many data structures can be presented as its quotients. For example, strings over Γ can be represented by terms over the ranked alphabet $\Gamma \uplus \{\varepsilon, \odot\}$, where symbols in Γ and ε have rank 0, while $\odot$ has rank 2 and represents concatenation. To enforce associativity and identity laws for ε, one quotients terms by the finest congruence $\sim$ satisfying $\odot(x, \odot(y, z)) \sim \odot(\odot(x, y), z)$ and $\odot(x, \varepsilon) \sim x \sim \odot(\varepsilon, x)$. Accordingly, updates on tuples of strings, such as those of the string register algebra, can be simulated by term substitutions: for example, concatenating w_1 and w_2 amounts to applying the term update $(w_1, w_2) \mapsto t[x_i/w_i]_{i=1,2}$, with $t = \odot(x_1, x_2)$.

4.2 Streaming transducers

In the following, we fix an arbitrary data structure $\mathbb{D}$. While different transducers from the same class will share the same data structure $\mathbb{D}$, it is convenient to allow different types of data at different control states of a transducer. For example, a transducer can store tuples of different arities at different states.

We denote by Q a set of *control states*. Each state $q \in Q$ is assigned a data type $\tau_Q(q)$, and hence a domain $\mathbb{D}_{\tau_Q(q)}$. Note that the assignment may depend on Q, and so the same state q may be assigned different types in different state sets. When Q is clear from the context, we drop the subscript Q from the notations $\tau_Q(q)$ and $\mathbb{D}_{\tau_Q(q)}$. The *configuration space generated by Q* is the set

$$\langle Q \rangle \;=\; \big\{(q, d) \;:\; q \in Q,\; d \in \mathbb{D}_{\tau(q)}\big\}.$$

A *configuration* is a state-data pair $(q, d) \in \langle Q \rangle$.

A *transformation* is a partial function $f : \langle Q \rangle \to \langle Q' \rangle$ between configuration spaces that is *specified* by a pair $(\hat{f}, \check{f})$, where:

- $\hat{f} : Q \to Q'$ is a partial function from Q to Q', describing the possible transitions between control states;
- $\check{f}$ is a partial function, with the same domain of definition as $\hat{f}$ (namely, $\hat{f}$ is defined on q iff $\check{f}$ is defined on q) that maps every control state $q \in \mathrm{Dom}(\check{f})$ to a corresponding update $\check{f}(q) \in \mathbb{D}_{\tau(q) \to \tau(r)}$, where $r = \hat{f}(q)$.

The transformation f specified by $(\hat{f}, \check{f})$ has for domain the set $\langle \mathrm{Dom}(\hat{f}) \rangle \subseteq \langle Q \rangle$ and maps any configuration $(q, d) \in \mathrm{Dom}(f)$ to the configuration $\big(\hat{f}(q),\, \check{f}(q)(d)\big) \in \langle Q' \rangle$. Note that the notion of transformation between configuration spaces is a restriction of that of a partial function, in two respects. First, the transformation relativized to a particular state must

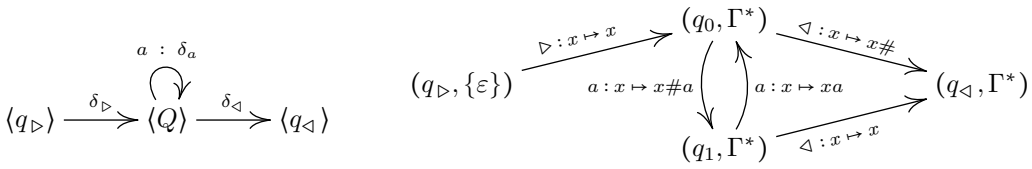


Figure 1 Diagrams of a transducer.

correspond to an update, which might already be a restricted form of function between data. Second, there might be a dependency of $\check{f}$ from $\hat{f}$, but not the other way around. For example, we can injectively map configurations with different states to configurations with the same state and different data, but not vice versa.

Transformations inherit from updates their closure under composition (proof omitted):

▶ **Lemma 5 (composition of transformations).** *Given two transformations $f : \langle Q\rangle \to \langle Q'\rangle$ and $g : \langle Q'\rangle \to \langle Q''\rangle$, specified respectively by $(\hat{f}, \check{f})$ and $(\hat{g}, \check{g})$, their composition is the transformation $h : \langle Q\rangle \to \langle Q''\rangle$ specified by $(\hat{h}, \check{h})$, where $\hat{h} = \hat{f};\hat{g}$ and $\check{h}(q) = \check{f}(q)\,;\check{g}(\hat{f}(q))$ for all $q \in Q$.*

To model the initial and final transformations of a transducer it is convenient to introduce two special states, $q_\triangleright$ and $q_\triangleleft$, and assume they are fixed as part of the transducer model, just as the underlying data structure $\mathbb{D}$ is. The state $q_\triangleright$ is assigned a special data type $\triangleright$ with a singleton domain $\mathbb{D}_\triangleright = \{d_\triangleright\}$. For example, sometimes we identify $\triangleright$ with the numeric type 0, and $d_\triangleright$ with the empty tuple $()$. Symmetrically, the state $q_\triangleleft$ is assigned type $\triangleleft$ (e.g. the number 1), but its domain usually contain multiple data, that is, the possible output values.

The states $q_\triangleright$ and $q_\triangleleft$ determine, respectively, the *initial configuration space* $\langle q_\triangleright\rangle = \{(q_\triangleright, d_\triangleright)\}$ and the *final configuration space* $\langle q_\triangleleft\rangle = \{(q_\triangleleft, d) \;:\; d \in \mathbb{D}_\triangleleft\}$. Both spaces are part of the underlying setup and hence are fixed for the entire class of transducers under consideration.

A *deterministic streaming transducer* (hereafter simply *transducer*) over an input alphabet Σ is a tuple $A = (Q, \delta_\triangleright, (\delta_a)_{a\in\Sigma}, \delta_\triangleleft)$, where:

- Q is a typed set of control states, generating $\langle Q\rangle$,
- $\delta_\triangleright : \langle q_\triangleright\rangle \to \langle Q\rangle$ is an *initial transformation*,
- $\delta_a : \langle Q\rangle \to \langle Q\rangle$ is an *internal transformation*, for each $a \in \Sigma$,
- $\delta_\triangleleft : \langle Q\rangle \to \langle q_\triangleleft\rangle$ is a *final transformation*.

Recall that transformations are allowed to be partial functions. Intuitively, $\delta_\triangleright(q_\triangleright, d_\triangleright)$, if defined, determines the first configuration of the run of the transducer; δ_a describes how the transducer moves from one configuration to another when reading an input letter a; and, $\delta_\triangleleft$ describes how the final output is constructed from the last configuration of the run. We remark that the state set Q of a transducer does not need to be finite; we will explicitly say *finite transducer* when we want Q to be finite.

The general structure of a transducer model is shown on the left of Figure 1. For a specific transducer, however, it is often useful to annotate control states with their data domains, and transitions with their input symbols and updates. The right-hand side of Figure 1 illustrates this representation with an example of a sequential transducer (the model is defined further below), which copies the input to the output while inserting a separator # before each pair of consumed input symbols. Note that the data domain associated with each control state is Γ^*, where $\Gamma = \Sigma \uplus \{\#\}$, and every update appends a constant string to the source data.

Given a transducer $A = (Q, \delta_\triangleright, (\delta_a)_{a\in\Sigma}, \delta_\triangleleft)$ and some input $w = a_1 \dots a_n \in \Sigma^*$, the *transformation induced* by w is the composition $\delta_w = \delta_{a_1}; \dots; \delta_{a_n}$ of the internal transformations induced by the letters in w. This composition can be prolonged to include the initial and/or the final transformation, resulting for instance in the transformation $\delta_{\triangleright w\triangleleft} = \delta_\triangleright; \delta_w; \delta_\triangleleft : \langle q_\triangleright \rangle \to \langle q_\triangleleft \rangle$. In particular, $\delta_{\triangleright w\triangleleft}(q_\triangleright, d_\triangleright) = \delta_\triangleleft(\delta_w(\delta_\triangleright(q_\triangleright, d_\triangleright)))$, and if defined, results in a configuration of the form $(q_\triangleleft, d_w)$, for some $d_w \in \mathbb{D}_\triangleleft$. In this case, d_w is called the *output* of A on w. The *transduction realized* by A is the partial function $\varphi : \Sigma^* \to \mathbb{D}_\triangleleft$ that maps any $w \in \mathrm{Dom}(\varphi)$ to the corresponding output d_w, as described above. We say that two transducers are *equivalent* if they realize the same transduction.

We conclude the section by showing how different models of transducers proposed in the literature can be seen as instances of our model, for a suitable choice of the data structure $\mathbb{D}$:

- ***Sequential transducers*** [61]. These are transducers over a data structure whose data are words over an output alphabet Γ, and whose updates are of the form $f_w : x \mapsto xw$, appending a fixed string w to the input string x. Figure 1 shows an instance of this model. Besides the special type $\triangleright$, with singleton domain $\mathbb{D}_\triangleright = \{\varepsilon\}$, the data structure has a single numeric type $\tau = 1$, whose domain is $\mathbb{D}_1 = \Gamma^*$.
- ***Polynomial automata*** [16]. These are transducers over the polynomial register algebra. A slight generalization of this class, called *cost register automata*, was introduced in [4] as a transducer that manipulates numeric registers using generic arithmetic operations. The latter model is also very similar in spirit to our notion of transducer over an abstract data structure.
- ***Streaming string-to-string transducers*** (***SSTs***) [2]. These are transducers over the string register algebra. For this model, it is worth recalling that there is a particularly interesting subclass of SSTs with copyless updates, which captures precisely the string-to-string transductions definable in monadic second-order logic [29].
- ***Streaming string-to-term transducers*** (***STTs***). These are transducers that consume input words, manipulate terms by first-order substitutions, and eventually output ground terms. They are essentially the model of [3, Section 3.6], except that our output trees are ranked, whereas those of [3] are unranked. Related models over unranked, unordered trees have also been studied in [18]. As discussed in Section 4.1, substitutions yield at least two natural data structures for defining updates on terms, and hence two variants of STTs. The first, called *downward STT*, uses the leaf substitution algebra and constructs outputs by extending linear terms at the leaves. The second, called *upward STT*, uses the free term algebra and extends tuples of ground terms at the roots. When updates of the latter variant are restricted to be copyless (as for SSTs, and as enforced in [3]), the downward and upward variants, both enhanced with *regular look-ahead*[4], are equivalent in expressive power and capture precisely the string-to-term transductions definable in monadic second-order logic [3, Theorem 16].

4.3 Transducer morphisms

Let $A = (Q_A \delta_\triangleright, (\delta_a)_{a\in\Sigma}, \delta_\triangleleft)$ and $B = (Q_B, \kappa_\triangleright, (\kappa_a)_{a\in\Sigma}, \kappa_\triangleleft)$ be two transducers from the same class, that is, with the same input alphabet Σ and the same data structure $\mathbb{D}$. A *transducer morphism* from A to B is a transformation $h : \langle Q_A \rangle \to \langle Q_B \rangle$ such that

- h is a total function,

[4] Regular look-ahead is the ability of a transducer to annotate its input with a co-deterministic Mealy machine, prior to consuming it.

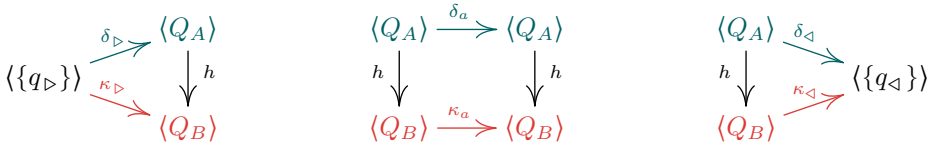


Figure 2 Commutative diagrams for a transducer morphism h.

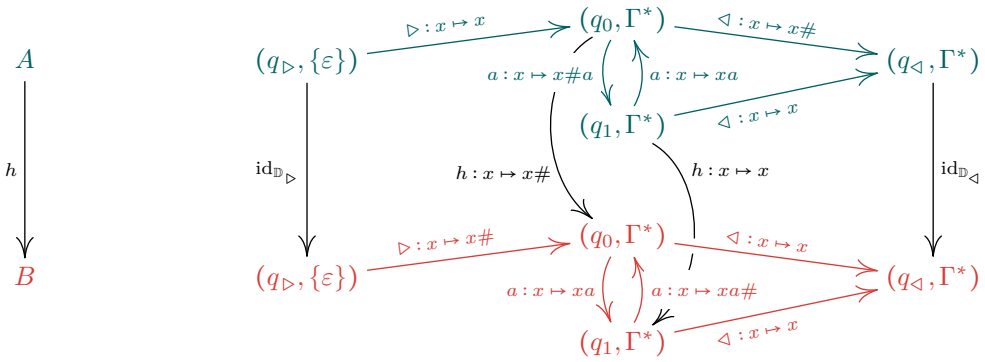


Figure 3 Example of morphism between two transducers.

- $\kappa_\triangleright = \delta_\triangleright ; h$,
- $h ; \kappa_a = \delta_a ; h$, for every $a \in \Sigma$,
- $h ; \kappa_\triangleleft = \delta_\triangleleft$.

The above identities are understood as equalities between the respective domains and codomains of the partial functions, conjoined with the pointwise equalities of the images, that is, $f = g$ means $\mathrm{Dom}(f) = \mathrm{Dom}(g)$, $\mathrm{Cod}(f) = \mathrm{Cod}(g)$, and $f(x) = g(x)$ for all $x \in \mathrm{Dom}(f)$. In particular, a morphism $h : A \to B$ can be equally defined as a total transformation that makes the diagrams in Figure 2 commute. Of course, from the commutativity of these diagrams it also follows that the transducers A and B are equivalent.

To conclude the section and support intuition, Figure 3 presents an example of a morphism between two instances of the sequential transducer model. The transducer at the top is the same as the one shown in Figure 1: it inserts a separator symbol # before each pair of consumed input symbols. The transducer at the bottom is an equivalent one that anticipates the insertion at the previous state. Notice that, in order to satisfy the commutativity conditions, the morphism transforms the data component of the former transducer into that of the latter in a non-trivial way. We also observe that no inverse morphism exists from the bottom transducer to the top one, since updates appending non-empty strings are not reversible in the underlying data structure.

5 Simple criteria for transducer minimization

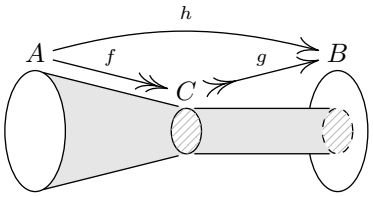
We fix again a data structure $\mathbb{D}$ and the induced class of transducers. We also fix a transduction φ realizable by transducers in this class, and think of the sub-class of transducers that realize φ as a category, with arrows denoting the possible morphisms between transducers.

To establish the existence of an algebraically minimal transducer that computes φ, we

use the approach outlined in Section 3, which boils down to showing that the category of transducers that compute φ admits strong factorizations of transducer morphisms, together with induced images, an initial object, and also a final object in the sub-category of initial images. In the next subsections we provide precisely the constructions of such objects.

5.1 Images of transducer morphisms

Recall that the image of a morphism $h : A \to B$ is an object C together with an epi $f : A \twoheadrightarrow C$ and a strong mono $g : C \rightarrowtail B$ such that $h = f;g$. For an ordinary function between sets, this is just the usual range construction: $C = \mathrm{Rng}(h)$, the map f is obtained from h by replacing its codomain with C, and g is the inclusion map of C into B, as depicted below:



In our setting, where A and B are transducers, a similar idea applies: taking the image of h amounts at restricting the configuration space of B to those elements that can be reached via h from the configuration space of A. This restriction acts at two levels: on the control states, by pruning the unreachable ones, and on the data domains associated with them, by removing (most of the) unreachable data. While pruning control states is easy, it is not clear at the moment how one can restrict a data domain. Our approach is to *approximate* the set of reachable data as the solution set of a suitable system of equations. Indeed, an exact representation of the set of reachable data is not necessary here, provided that updates cannot distinguish the exact set from its approximation. To formalize this idea, we shall expand our data structure $\mathbb{D}$ with abstract types, each associated with a domain that is defined by a system of equations.

We define a *constraint* of type τ as a (possibly infinite) set γ of pairs (f, g) of updates with τ as input type. Any such pair (f, g) represents an equation[5] of the form $f(x) = g(x)$, where x is a variable denoting arbitrary data from the unrestricted domain $\mathbb{D}_\tau$. The *solution set* of the constraint γ is defined as

$$\mathbb{D}_\gamma \;=\; \big\{d \in \mathbb{D}_\tau \;:\; \forall (f,g) \in \gamma \quad f(d) = g(d)\big\}.$$

The constraint γ can be added to the data structure $\mathbb{D}$ as a new type (actually a sub-type of τ), and its solution set $\mathbb{D}_\gamma$ can be thought of as the data domain associated with it. We call these new types and domains, respectively, *constrained types* and *constrained domains*.

The updates from a constrained type γ are nothing but the updates from the underlying basic type τ restricted to $\mathbb{D}_\gamma$. Of course, also the codomain of an update can be constrained, provided that it still covers all the images of the update. The other definitions of configuration space, transformation, transducer, and transducer morphism are adapted, almost verbatim, in presence of constrained domains.

▶ **Example 6.** Let $\mathbb{D}$ be the polynomial register algebra and consider the updates $f, g : \mathbb{D}_2 \to \mathbb{D}_1$ defined by $f(x, y) = x^2 + y^2$ and $g(x, y) = 1$. The constraint $f(x, y) = g(x, y)$ induces a new domain $\mathbb{D}_{\{(f,g)\}} \subseteq \mathbb{D}_2$ that coincides with the unit circle.

[5] This idea is similar to the use of equalizers in category theory; in particular, a constraint can be seen as a special case of a multi-equalizer.

By design, constrained domains are closed under inverses of updates and intersections:

▶ **Lemma 7 (closures of constrained domains).** *Constrained domains are effectively closed under inverses of updates and under finite intersections. They are also closed (non-effectively) under infinite intersections.*

The above result is useful for deriving a closure operator that over-approximates any set D of data of uniform type τ by the intersection of all the possible constrained domains that contain D. We call this over-approximation the *closure of D*, and denote it by $\mathrm{cl}(D)$. Equivalently, one can define $\mathrm{cl}(D)$ as the solution set $\mathbb{D}_{\gamma_D}$ of the constraint $\gamma_D = \{(f,g) : \forall d \in D\ f(d) = g(d)\}$ that contains all equations that are valid over D.

▶ **Example 8.** Let $\mathbb{D}$ be again the polynomial register algebra and $D = \left\{\left(\frac{1-n^2}{1+n^2}, \frac{2n}{1+n^2}\right) : n \in \mathbb{N}\right\} \subseteq \mathbb{D}_2$. This set contains infinitely many points on the unit circle from Example 6, and its closure coincides with circle itself. We also observe that every finite system of polynomial equations, and hence every constrained domain of the polynomial register algebra, can be represented by a single constraint of the form $f(x) = g(x)$, for suitable updates $f, g : \mathbb{D}_n \to \mathbb{D}_m$. Moreover, using the fact that the set of roots of a product of polynomials is the union of the sets of roots of its factors, one can show that constrained domains are closed under finite unions. In particular, every finite set is its own closure. Thus, over the polynomial register algebra, the closure operator behaves like a topological closure — as a matter of fact, this is known as Zariski topology [30]. This behavior, however, is specific to the polynomial setting and should not be expected for arbitrary data structures. For example, it fails for the data structure with linear or affine updates and also for the free term algebra, as shown in the next example.

▶ **Example 9.** Let $\mathbb{D}$ be the free term algebra over a ranked alphabet Γ. In this data structure, the only constrained domains contained in $\mathbb{D}_1$ are $\varnothing$, the singletons, and $\mathbb{D}_1$ itself; this follows from Lemma 25, presented later in Section 6.1. In particular, if $D \subseteq \mathbb{D}_1$ contains at least two distinct terms, then $\mathrm{cl}(D) = \mathbb{D}_1$. Non-trivial closures may nevertheless arise at higher numerical types. For instance, let $D = \{(a,a),(b,b)\} \subseteq \mathbb{D}_2$, where a, b are distinct rank-0 symbols of Γ. If Γ also contains a binary symbol c, then $\mathrm{cl}(D) = \{(t,t) : t \in \mathbb{D}_1\}$, as witnessed by the equation $f(x,y) = g(x,y)$, with $f : (x,y) \mapsto c(x,y)$ and $g : (x,y) \mapsto c(y,x)$.

It is easy to see that the closure operator $\mathrm{cl}(-)$ is *extensive* (i.e. $D \subseteq \mathrm{cl}(D)$), *monotone* (i.e. $D \subseteq D'$ implies $\mathrm{cl}(D) \subseteq \mathrm{cl}(D')$), and *idempotent* (i.e. $\mathrm{cl}(\mathrm{cl}(D)) = \mathrm{cl}(D)$), and hence the following lemma holds:

▶ **Lemma 10 (closure vs union).** *For all sets $D, D' \subseteq \mathbb{D}_\tau$, $\mathrm{cl}(D \cup D') = \mathrm{cl}\big(\mathrm{cl}(D) \cup \mathrm{cl}(D')\big)$.*

It is also convenient to generalize the closure operator from sets of data to sets of configurations. Given $S \subseteq \langle Q \rangle$, we define $\mathrm{cl}(S) = \{(q,d) : q \in Q,\ d \in \mathrm{cl}(D_{S,q})\}$, where $D_{S,q} = \{d' \in \mathbb{D}_{\tau(q)} : (q,d') \in S\}$. Note that $\mathrm{cl}(S)$ can be seen as a new configuration space $\langle \tilde{Q} \rangle$, where $\tilde{Q}$ contains a copy $\tilde{q}$ of each state $q \in Q$ associated with a constrained domain $\mathbb{D}_{\tau(\tilde{q})} = \mathrm{cl}(S_q)$.

We will soon see that the image of a transformation h between configuration spaces, and hence in particular of a transducer morphism, is obtained as the closure of the range of h. The current definition of constrained domain, however, may lack a finite presentation, since it allows infinite systems of equations. This is harmless for proving the existence of images of transducer morphisms, but it raises the question of whether the same result holds when all components of a transducer are required to be finitely presentable. A natural setting is

obtained by restricting to *finitary constrained domains*, namely constrained domains that are solution sets of finite systems of equations. Since finite systems are not, by themselves, closed under arbitrary intersections, we rely on the following compactness assumption for the data structure $\mathbb{D}$ under consideration:

▶ **Assumption 1 (compactness).** *For every numeric type τ of $\mathbb{D}$ and for every constraint γ of type τ, there is a finite subset γ' of γ that is a constrained type of $\mathbb{D}$ and has the same solution set as γ. Therefore, all constrained domains are allowed in $\mathbb{D}$ and they are finitary.*

The canonical example of a data structure satisfying this assumption is, again, the polynomial register algebra. For this structure, Hilbert's finite basis theorem [42] implies that every constraint is equivalent to a finite subset of it. This result serves as a foundation for deriving similar compactness properties for other data structures, such as the string register algebra or the free term algebra. We shall use Hilbert's theorem later, when we will present concrete classes of transducers that admit algebraically minimal objects.

Now, turning to the proof of existence of images of transducer morphisms, we are going to exploit closure properties of constrained domains and the derived closure operator. We first introduce a few technical lemmas, one showing a continuity property of transformations w.r.t. closures, another characterizing transformations that are epis, and a third one giving sufficient conditions for transformations to be strong mono.

▶ **Lemma 11 (closure-continuity of transformations).** *For every transformation $f : \langle Q\rangle \to \langle Q'\rangle$ and every subset S of $\langle Q\rangle$, we have $f(\mathrm{cl}(S)) \subseteq \mathrm{cl}(f(S))$.*

▶ **Lemma 12 (epi transformations).** *Let $f : \langle Q\rangle \to \langle Q'\rangle$ be a transformation specified by $(\hat{f}, \check{f})$. The following conditions are equivalent:*

- *f is epi,*
- *$\mathrm{Cod}(f) \subseteq \mathrm{cl}(\mathrm{Rng}(f))$,*
- *$\hat{f} : Q \to Q'$ surjective and $\mathbb{D}_{\tau(r)} \subseteq \mathrm{cl}\left(\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\right)$ for every $r \in Q'$.*

▶ **Lemma 13 (strong mono transformations).** *Every inclusion map $g : \langle Q\rangle \to \langle Q'\rangle$, where $\langle Q\rangle \subseteq \langle Q'\rangle$ and $g(q, d) = (q, d)$ for all $(q, d) \in \langle Q\rangle$, is a strong mono.*

Since transducer morphisms are particular forms of transformations, the two lemmas above imply similar sufficient conditions for a transducer morphism to be epi, resp. strong mono.[6] We remark, however, that these results do not give exact characterizations of epis and strong monos among transducer morphisms. For instance, because the definition of epi involves a universal quantification over the available arrows, a transducer morphism may be epi relative to the class of transducer morphisms, while failing to be epi relative to the larger class of transformations.

Using the above results, we can now prove the existence of strong factorizations of transducer morphisms.

▶ **Proposition 14 (images).** *Every transducer morphism $h : A \to B$ has a strong factorization $A \overset{f}{\twoheadrightarrow} C \overset{g}{\rightarrowtail} B$, with f epi and g strong mono.*

[6] In fact, the existential quantification in the diagonal fill-in property for the definition of strong mono requires to check that the resulting diagonal d is a transducer morphism. This is readily done by inspecting the proof of Lemma 13 in the long version of the paper, which constructs d from a given transformation h' by restricting the codomain. Clearly, this construction is also applicable to transducer morphisms.

Proof sketch. The transducer C is defined as a "pruning" of B, and specifically by replacing the configuration space of B with the closure $\mathrm{cl}(\mathrm{Rng}(h))$ of the range of the morphism $h : A \to B$. Accordingly, the morphism h is factorized as $A \overset{f}{\twoheadrightarrow} C \overset{g}{\rightarrowtail} B$, where the epi f is obtained by restricting the codomain of h to $\mathrm{cl}(\mathrm{Rng}(h))$ and the strong mono g is the inclusion map from $\mathrm{cl}(\mathrm{Rng}(h))$ to $\mathrm{Cod}(h)$. ◀

5.2 The initial transducer

The construction of an initial transducer does not require additional assumptions and it is a straightforward generalization of the construction of the initial object in the category of finite state automata. Intuitively, the states of the initial transducer are identified with the input words over Σ and there is no internal memory. The output is produced by simply mapping each state w to the corresponding value $\varphi(w)$, if defined. As a matter of fact, any partial function $\varphi : \Sigma^* \to \mathbb{D}_\triangleleft$ can be realized by such a transducer.

▶ **Proposition 15 (initial transducer).** *There is an (infinite) transducer I, with words over Σ as control states, that is initial, namely, that admits a unique morphism $!^A : I \to A$ towards each transducer A that realizes φ.*

5.3 The final transducer for the initial images

The construction of a final transducer M is the most delicate part, even after restricting to the sub-category of images of initial morphisms. The goal is to extract from the transduction φ some configurations that "cover" the reachable configurations of every transducer A realizing φ, so as to obtain a final morphism from $\mathrm{Reach}(A)$ to M. For this morphism to be unique, the values stored in M must be as large as possible so as to cover the values stored by any equivalent transducer at the corresponding control states.

This brings us to one of the main difficulties of the minimization result. Given a set U of updates, we need factorizations of the form $u = g; v_u$ for every $u \in U$. In this case, the update g is called a common (inner) divisor of U and the updates v_u are the corresponding (outer) residuals. Moreover, g should be a *greatest* common divisor, namely, every other common divisor g' of U must itself be a divisor of g. This requirement already appears in the minimization of sequential transducers [24], where the longest common prefix of a set of outputs plays the role of a greatest common divisor in the data structure that manipulates strings by appending constants.

The additional difficulty here is that we work with an abstract, sorted monoid of updates, not just with a free monoid. Thus, greatest common divisors need not be unique, and different common divisors may induce different residuals. A common way to address this issue is to assume that the monoid of updates comes equipped with a canonical choice of greatest common divisor and residuals. This approach is pursued, for instance, in [47, 40, 8, 7], and, to some extent, also in [36], where canonical factorizations of sets of updates underpin both minimization and automata learning results.

Here we follow a different route. Instead of postulating canonical factorizations of updates, whose choice may be non-obvious or unnatural in many data structures, we use strong factorizations (Proposition 14) and the diagonal fill-in property. This allows us to restrict, without loss of generality, to divisors of updates that are epis, and to show that greatest common divisors and their residuals —whenever they exist— behave canonically enough for the minimization construction to go through.

We recap the important notions that we are going to use. We call *vector* a family U of updates indexed by words $t \in \Sigma^*$. These updates must share the same input type α and

output type β, and are thus of the form $U[t] : \mathbb{D}_\alpha \to \mathbb{D}_\beta$. For convenience, we allow vectors to contain undefined entries, which we denote by writing $U[t] = \bot$ for some $t \in \Sigma^*$. We also tacitly assume that the set of indices t where the entries of U are defined is prefix-closed, namely, if $U[t] \neq \bot$ and t' is a prefix of t, then $U[t'] \neq \bot$. We lift composition from functions to vectors in a pointwise manner: given an update f and a vector U, where the domains of the updates $U[t]$ are all equal to the codomain of f, we let $f;U$ be the vector defined by $(f;U)[t] = f;(U[t])$ for all $t \in \Sigma^*$, with the convention that $f;\bot = \bot$. We call *sub-vector* of $U = (U[t])_{t\in\Sigma^*}$ a vector of the form $U[a-] = (U[at])_{t\in\Sigma^*}$, for any letter $a \in \Sigma$.

Given a factorization $u = g;v$, we say that g is a *divisor* of u and v is a *residual* of u via g. If g is also an epi, we call it an *epi divisor*, and remark that in this case there is a *unique* residual of u via g, which we denote by $g\backslash u$. A *common divisor* of a vector U is a divisor g of every $u \in U$. If g is also epi, it is called an *epi common divisor*, and the *residual vector* of U via g is the vector $g\backslash U$ defined by $(g\backslash U)[t] = g\backslash U[t]$ for all $t \in \Sigma^*$. A *greatest common divisor* (*GCD* for short) of U is a common divisor of U that is divided by every other common divisor of U. By convention, if $U[t] = \bot$ for all $t \in \Sigma^*$, then we let $\bot$ be its unique GCD. Similarly, we define an *epi greatest common divisor* (*EGCD*) of U as an epi common divisor of U that is divided by every other epi common divisor of U.

Note that an EGCD of U is a special case of a common divisor of U, but it is not necessarily greatest w.r.t. all possible common divisors of U. In particular, the notions of GCD and EGCD are incomparable. Despite this, we can still relate these two notions via the the following result, which exploits the existence of strong factorizations (Proposition 14) and the diagonal fill-in property:

▶ **Lemma 16 (GCD vs EGCD).** *If g is a GCD of a vector U, then there is a strong factorization $g = g';g''$ of it, where g' is an EGCD of U.*

We now introduce our second assumption, to be proven later for specific classes of updates:

▶ **Assumption 2 (existence of GCDs).** *Every set of updates of uniform input and output types admits a GCD.*

Note that if Assumption 2 holds, then every vector has a GCD, and by Lemma 16 also an EGCD. *The results that follow tacitly rely on this assumption, and in particular on the existence of EGCDs of vectors.*

▶ **Lemma 17 (uniqueness of EGCDs).** *The EGCDs of any given vector U are pairwise isomorphic.*

By Lemma 17, for every two EGCDs g, g' of U, there is an isomorphism j such that $g;j = g'$. In particular, g, g' have the same domain and isomorphic codomains. Moreover, if $V = g\backslash U$ and $V' = g'\backslash U$ are the corresponding residual vectors, then, because $g;V = U = g';V'$, the same j above is also an isomorphism between $V[t]$ and $V'[t]$, namely, $V[t] = j;V'[t]$ for all $t \in \Sigma^*$. We shall denote this latter property by writing $V \overset{j}{\cong} V'$.

▶ **Lemma 18 (distributivity of EGCDs).** *If $U = g;V$, for some epi g and some vector V, and h is an EGCD of V, then $g;h$ is an EGCD of U.*

▶ **Corollary 19 (EGCDs of sub-vectors).** *Let g is an EGCD of U and let $V = g\backslash U$ be the associated residual vector. For every $a \in \Sigma$, if h is an EGCD of the sub-vector $V[a-]$, then $g;h$ is an EGCD of the sub-vector $U[a-]$.*

We can now turn towards the construction of the final transducer. We shall work with several vectors at once, which are conveniently represented by *matrices*, namely, families $H = (H[s,t])_{s,t\in\Sigma^*}$ of updates (or undefined entries) indexed by pairs of words $s,t \in \Sigma^*$. Each row $H[s,-]$ of H is a vector, and thus, the domains and codomains of its updates are all the same. A column of H is not necessarily a vector, namely, its updates may have different domains or codomains: we call it a *column vector*.

Specifically, we shall consider the *Hankel matrix* H of the transduction φ, whose entries are defined as $H[s,t] = \varphi(st)$ for every $s,t \in \Sigma^*$, where $\varphi(st)$ is seen as an update from the initial data type $\mathbb{D}_\triangleright = \{d_\triangleright\}$ to the final data type $\mathbb{D}_\triangleleft$. The Hankel matrix is nothing but a convenient way to describe a transduction, and eventually extract EGCDs and corresponding residual vectors. Also observe that every sub-vector $H[s,a-]$ of a row of the Hankel matrix coincides with a "successor" row $H[sa,-]$.

Next, we apply Assumption 2 to each row $H[s,-]$ of the Hankel matrix, obtaining an EGCD g_s and a corresponding residual vector $V_s = g\backslash H[s,-]$ for every $s \in \Sigma^*$. We organize these objects into a column vector $G = (g_s)_{s\in\Sigma^*}$ and a new matrix $R = (V_s[t])_{s,t\in\Sigma^*}$. G and R are called, respectively, the *divisor vector* and the *residual matrix* of H. The $\cong$-equivalence classes of the rows of the residual matrix R will represent the control states of our final transducer, and can be equally seen as the classes of a one-sided Myhill-Nerode equivalence. In a similar way, the divisor vector G will be used to define the transformations of the final transducer. These constructions crucially rely on suitable compatibility properties that are reminiscent of the right-invariance property of the classical Myhill-Nerode equivalence, and that are established in Lemma 20 below.

▶ **Lemma 20 (right-invariance).** *Under Assumption 2, one can find some column vectors G and ∂_a (one for every letter $a \in \Sigma$), and a matrix R such that, for all $s \in \Sigma^*$ and $a \in \Sigma$:*

- *$G[s]$ is an EGCD of $H[s,-]$ and $R[s,-] = G[s]\backslash H[s,-]$,*
- *$\partial_a[s]$ is an EGCD of $R[s,a-]$ and $R[sa,-] = \partial_a[s]\backslash R[s,a-]$,*
- *$G[sa] = G[s];\partial_a[s]$.*

In particular, $G[s]$ and $R[s,-]$ are uniquely determined from $H[s,-]$, and $\partial_a[s]$ is uniquely determined from $R[s,-]$.

We conclude with the construction of the final transducer M, which provides the last ingredient for the existence of algebraically minimal transducers:

▶ **Proposition 21 (final transducer).** *There is a transducer M that is final in the sub-category of images of initial morphisms, namely, for every transducer A that realizes φ, there is a unique morphism $!_{\mathrm{Reach}(A)} : \mathrm{Reach}(A) \to M$.*

Proof sketch. The final transducer M is obtained as a variation of the classical Myhill-Nerode construction: the states are strings r_s serving as representatives of $\cong$-classes of residual vectors of the form $R[s,-]$, and the transitions are induced by the corresponding "derivatives" $\partial_a[r_s]$. Some bookkeeping on the isomorphisms that witness $\cong$-equivalences between residual vectors is required to ensure that the transducer produces the exact output $\varphi(w)$, rather than only an isomorphic one. The existence of a unique morphism from every initial image $\mathrm{Reach}(A)$ to M follows from a standard reachability argument. Every state q of $\mathrm{Reach}(A)$ is reached by some string, and all strings reaching q induce residual vectors in the same $\cong$-class. Hence the final morphism maps q to a representative r_s of the $\cong$-class of $R[s,-]$, for any string s that reaches q. The update attached to this mapping is the residual obtained when the update produced by $\mathrm{Reach}(A)$ after reading s is factored through the EGCD $G[r_s]$. ◀

5.4 Minimization results and effectiveness

A first minimization result now follows directly from Propositions 4, 14, 15, and 21:

▶ **Theorem 22 (minimal transducer).** *If $\mathbb{D}$ is a data structure with constrained domains that satisfies Assumption 2, then every transduction realized by a transducer over $\mathbb{D}$ is also realized by one that is algebraically minimal.*

We can also prove an analogous result for the subclass of *finite* transducers, using the additional assumption that all constrained domains are finitary:

▶ **Theorem 23 (minimal finite transducer).** *If $\mathbb{D}$ is a data structure with constrained domains that satisfies both Assumptions 1 and 2, then every transduction realized by a finite transducer over $\mathbb{D}$ is also realized by one that is finite and algebraically minimal.*

Proof. First of all, we claim that if a transduction is realized by some transducer A with finitely many control states, then every algebraically minimal transducer M also has finitely many control states. This is because $\mathrm{Reach}(A)$ is a subobject of A, hence it has no more control states than A, and, similarly, $\mathrm{Obs}(\mathrm{Reach}(A))$ is a quotient of $\mathrm{Reach}(A)$, hence it has no more control states than $\mathrm{Reach}(A)$. So, finiteness of control states transfers from A to $\mathrm{Obs}(\mathrm{Reach}(A)) \simeq M$. As for the constrained domains that may appear inside M, these are finitely presentable under the additional Assumption 1. ◀

Moreover, we can show that Assumption 2 is also necessary for the existence of algebraically minimal models, at least over standard data structures. By *standard data structure* we mean a data structure where every domain $\mathbb{D}_\alpha$ and every set of updates $\mathbb{D}_{\alpha\to\beta}$ is countable, and such that every common divisor of $\mathbb{D}_{\alpha\to\beta}$ is isomorphic to the identity on $\mathbb{D}_\alpha$. When the latter condition holds, we also say that the updates in $\mathbb{D}_{\alpha\to\beta}$ are *jointly coprime*. Most data structures —in particular all examples considered here— are standard. For instance, the updates in $\mathbb{D}_{2\to 1}$ of the free term algebra $\mathbb{D}$ are jointly coprime, since already for $u_1 : (x, y) \mapsto a(x, y)$ and $u_2 : (x, y) \mapsto b(x, y)$ the only common divisors are the identity $(x, y) \mapsto (x, y)$ and the swap $(x, y) \mapsto (y, x)$.

▶ **Theorem 24 (necessity of GCDs).** *Assumption 2 holds over any standard data structure $\mathbb{D}$ whenever, for every transduction φ, the class of transducers over $\mathbb{D}$ realizing φ contains an algebraically minimal model.*

Proof sketch. Let $U \subseteq \mathbb{D}_{\alpha\to\beta}$ be a set of updates. Using countability of $\mathbb{D}_\alpha$ and $\mathbb{D}_{\alpha\to\beta}$, we encode each data $d \in \mathbb{D}_\alpha$ by a string $\tilde{d}$, and each update $u \in \mathbb{D}_{\alpha\to\beta}$ by a string $\tilde{u}$. We then extend the encoding alphabet with fresh symbols $\blacksquare, \blacktriangle, \blacktriangledown$, and define a transduction φ as follows: every input $\tilde{d}\blacksquare\blacktriangle\tilde{u}$ is mapped to $u(d)$, while $\tilde{d}\blacksquare\blacktriangledown\tilde{u}$ is mapped to $u(d)$ only if $u \in U$; all other inputs have undefined outputs.

For every common divisor f of U, there is a transducer A_f realizing φ as follows. After reading a prefix $\tilde{d}\blacksquare$, A_f reaches a distinguished state q_f storing d. From q_f, a $\blacktriangle$-labelled transition applies the identity on $\mathbb{D}_\alpha$ and reaches a component that consumes suffixes $\tilde{u}$, eventually producing $u(d)$. Similarly, from q_f, a $\blacktriangledown$-labelled transition applies the update f and reaches a component that consumes suffixes $\tilde{u}$, eventually producing $u(d)$ only when $u \in U$.

Next, we exploit the existence of an algebraically minimal transducer M realizing the same transduction φ. Since each A_f is pruned by design, i.e. $A_f = \mathrm{Reach}(A_f)$, M is a quotient of A_f. Hence the morphism from A_f to M maps the $\blacktriangledown$-labelled transition of A_f to a fixed $\blacktriangledown$-labelled transition of M, whose update we denote by g. This forces g to factor through

every common divisor f of U. It remains to see that g is a common divisor of U. The runs of M labelled by $\blacktriangle\tilde{u}$ ensure that no non-trivial common divisor can be accumulated before reading $\blacktriangledown$: otherwise, such a non-trivial accumulated update would divide all updates in $\mathbb{D}_{\alpha\to\beta}$, contradicting joint coprimality. Therefore g must divide every update in U, implying that it is indeed a GCD of U. ◀

Finally, it is natural to ask whether an algebraically minimal transducer M can be *computed* from a given finite transducer A. Here there is no general recipe: computability depends on the class of transducers at hand and on suitable effectiveness assumptions. Nevertheless, the proof of Theorem 22, in particular Propositions 14 and 21, suggests the following general procedure.

1. For each state q of $A = (Q, \delta_{\triangleright}, (\delta_a)_{a\in\Sigma}, \delta_{\triangleleft})$, compute a GCD f_q of the vector U_q of updates induced by maximal runs that depart from q, namely, $U_q[t] = \check{\delta}_{t\triangleleft}(q)$ for all $t \in \Sigma^*$. This can be done, assuming an effective version of Assumption 2, by viewing the f_q's as unknowns in the system of equations

$$f_q \;=\; GCD\big(\{\check{\delta}_{\triangleleft}(q)\} \cup \{\check{\delta}_a(q); f_{q'} \,:\, a \in \Sigma,\, q' = \hat{\delta}_a(q)\}\big) \qquad (\star)$$

   and by solving it via a greatest fixpoint computation. More precisely, one starts by setting all variables f_q to the undefined update $\bot$, and repeatedly assigns each variable f_q the update in the right-hand side of $(\star)$. Termination is guaranteed when the divisor preorder is well-founded, since the updates f_q can only decrease w.r.t. the divisor preorder.
2. Use Lemma 16 to turn the GCDs f_q into corresponding EGCDs g_q, and normalize the updates of A as follows. The initial update $\check{\delta}_{\triangleright}(q_{\triangleright})$ is post-composed with g_q, where $q = \hat{\delta}_{\triangleright}(q_{\triangleright})$. Similarly, for each a-labelled transition from q to $q' = \hat{\delta}_a(q)$, compute the residual of $\check{\delta}_a(q)$ via g_q, and then post-compose with $g_{q'}$, basically replacing $\check{\delta}_a(q)$ by $(g_q \backslash \check{\delta}_a(q)); g_{q'}$. Finally, replace each final update $\check{\delta}_{\triangleleft}(q)$ by $g_q \backslash \check{\delta}_{\triangleleft}(q)$. This yields an equivalent transducer B, with the same control states as A, and such that, for every $q \in Q$, the EGCD of the vector V_q of updates induced by the maximal runs that depart from q is isomorphic to the identity.
3. Construct the initial image $C = \mathrm{Reach}(B)$, as in Proposition 14, by restricting the configuration space of B to the closure of the set of reachable configurations. Equivalently, this closure is the smallest inductive invariant of $B = (Q, \kappa_{\triangleright}, (\kappa_a)_{a\in\Sigma}, \kappa_{\triangleleft})$, obtained as the smallest fixpoint of

$$F\colon\; S \;\mapsto\; \mathrm{cl}\big(S \cup \{\kappa_{\triangleright}(q_{\triangleright}, d_{\triangleright})\} \cup \textstyle\bigcup_{a\in\Sigma} \kappa_a(S)\big).$$

   Computing this fixpoint may require data-specific methods, such as adaptations of techniques from linear-algebra (cf. [45]).
4. Finally, merge every two states q, q' of C such that $V_q \overset{j}{\cong} V_{q'}$ for some isomorphism $j : \mathrm{Dom}(V_q[\varepsilon]) \to \mathrm{Dom}(V_{q'}[\varepsilon])$. The difficulty here is to find the witnessing isomorphism j, if it exists. When only finitely many candidate isomorphisms j exist, one can enumerate them and test which j satisfies $V_q = j; V_{q'}$. This test reduces to a variant of the equivalence problem between two copies of C, which can often be solved by a backward fixpoint computation inspired by [45] (see also [31, 16, 50, 62]).
   Concretely, one starts from the equation $x = x'$ between the final outputs of the two copies of C and propagates it backwards along pairs of equally labelled transitions, using effective closure of systems of equations under inverse images of updates and finite intersections (Lemma 7). This yields increasingly stronger systems of equations associated with pairs of control states; under Assumption 1, and assuming decidable entailment, the fixpoint

can be effectively detected. One then checks whether the system associated with (q, q') entails the equation $x = v(x')$, where x, x' denote the possible data at q, q'.

Once $V_q \overset{j}{\cong} V_{q'}$ is detected, one merges q' into q by redirecting every transition entering (resp. exiting) q' to enter (resp. exit) q, and by pre-composing its update with j (resp. post-composing its update with j^{-1}). By Proposition 21, the resulting transducer is precisely $\mathrm{Obs}(C) = \mathrm{Obs}(\mathrm{Reach}(B))$. This transducer is also isomorphic to $\mathrm{Obs}(\mathrm{Reach}(A))$ since A is equivalent to B (see remark after Proposition 4), and therefore to M.

For these steps to be effective, Assumptions 1 and 2 must be strengthened by computability requirements. Some are straightforward: one typically observes that composition of updates, residuals by epi divisors, GCDs of finite sets, and entailment between systems of equations are computable. These suffice for the first two steps. The remaining steps are more subtle. Computing the initial image $C = \mathrm{Reach}(B)$ may already be hard. For example, for a single-state transducer B over the polynomial register algebra, it amounts to computing the smallest algebraic invariant of polynomial maps, which is not feasible in general [44, Theorem 18]. Likewise, there is no general method for enumerating isomorphisms between domains, as needed to test $V_q \cong V_{q'}$. Effective solutions are therefore model-dependent.

6 Applications to string-to-term transducers

In this part we use Theorems 22 and 23 to show that two classes of transducers, namely, downward STTs and upward STTs, admit algebraically minimal models, which can moreover be computed from finite transducers.

6.1 Minimization of downward STT

Recall that downward STTs are transducers over the leaf substitution algebra, whose updates transform linear terms by applying substitutions at the leaves.

The first step towards effective minimization is to show that Assumption 1 (compactness of constrained domains) holds over the leaf substitution algebra. In fact, in this particular data structure, it can be shown that all constrained domains are trivial, namely, they either coincide with the full domain over which they are defined or they are empty. This collapse, as well as the fact that all updates of the data structure are epis, is a consequence of a non-overlap property of substitutions, which we state below without proof:

▶ **Lemma 25 (non-overlap property).** *For all terms t over a τ-tuple $\bar{x}$ of variables and for all τ-tuples of terms $\bar{u}$ and $\bar{v}$, $t[\bar{x}/\bar{u}] = t[\bar{x}/\bar{v}]$ implies $\bar{u} = \bar{v}$.*

▶ **Corollary 26 (compactness).** *Assumption 1 holds trivially for the leaf substitution algebra.*

▶ **Corollary 27 (epi updates).** *Every update of the leaf substitution algebra is an epi.*

Another consequence of the above results is that the initial image $\mathrm{Reach}(A)$ can be computed from a given finite transducer A by simply pruning the unreachable control states:

▶ **Corollary 28 (effective initial images).** *Given a finite downward STT A, one can compute its initial image* $\mathrm{Reach}(A)$.

As for Assumption 2 (existence of GCDs), recall that updates over the leaf substitution algebra are specified by tuples of linear terms. The divisor relation between such updates coincides with the standard notion of generalization under first-order substitution: given two tuples $\bar{u} = (u_1, \ldots, u_\alpha)$ and $\bar{v} = (v_1, \ldots, v_\alpha)$ of linear terms, the update specified by $\bar{u}$ divides

the one specified by $\bar{v}$ iff $\bar{u}$ *generalizes* $\bar{v}$, that is, iff there is a tuple $\bar{w} = (w_1, \ldots, w_\beta)$ of linear terms such that $u_i[\bar{w}] = v_i$ for all $i = 1, \ldots, \alpha$. Accordingly, a GCD of a set of updates over the leaf substitution algebra is precisely a *least general generalization* (*anti-unifier* for short) of the corresponding tuples of terms. Anti-unification has been studied since the 70's: Plotkin and Reynolds [53, 54] independently proved that finite sets of terms admit anti-unifiers, unique up to variable renaming. Their theory adapts directly to our linear terms, by requiring the witnesses of anti-unification to be linear as well. Although the classical results concern finite sets, the same existence proof extends to infinite sets; in that case, the anti-unifier is no longer computable in general, but can still be constructed by induction.

▶ **Lemma 29 (see for instance [53, Theorem 1]).** *Assumption 2 holds for the leaf substitution algebra, namely, all sets of updates over the leaf substitution algebra admit GCDs. Moreover, these GCDs can be computed when the sets are finite.*

Not only one can compute GCDs of finite sets of updates: using the fixpoint method outlined in Section 5.4 (Step (2)), one can also compute GCDs of vectors of updates induced by the control states of a finite transducer.

Summing up, the existence of minimal STTs and the possibility of computing them from given finite STTs, follows immediately from Corollaries 26 and 28, Lemmas 29 and 16, Theorems 22 and 23, and from the discussion on effectiveness provided in Section 5.4.

▶ **Theorem 30 (minimal downward STTs).** *Downward STTs admit algebraically minimal transducers. Moreover, minimization can be performed effectively on any given finite transducer.*

We conclude by noting a similarity between the leaf substitution algebra and the algebra of strings with updates that prolong strings by appending constants. The latter algebra can be seen as a particular case of the former, by simply identifying every string w with a unary tree that has a substitution variable at the leaf. In this perspective, Theorem 30 can be seen as a natural generalization of the minimization result for sequential transducers [24].

6.2 Minimization of upward STT

We now turn to the upward variant of STTs, with updates over the free term algebra. Here the results are slightly more difficult, but also interesting. Recall that in the free term algebra, an update from type α to type β maps any α-tuple $\bar{u}$ of ground terms to the β-tuple $\bar{t}[\bar{x}/\bar{u}]$ of ground terms, where $\bar{t}$ only depends on the update under consideration and contains terms over the variables $\bar{x} = (x_1, \ldots, x_\alpha)$.

We first discuss which updates should be allowed in the free term algebra, with the goal of obtaining reasonable minimal instances of upward STTs. On the one hand, erasing updates should be forbidden, since they may produce minimal transducers with useless registers. Indeed, if erasing updates were allowed, then any two updates of the form $f : (x_1, \ldots, x_\alpha) \mapsto (t_1, \ldots, t_\beta)$ and $f' : (x_1, \ldots, x_\alpha) \mapsto (t_1, \ldots, t_\beta, t_{\beta+1})$ would be mutual divisors, implying that one could add to an upward STT arbitrarily many registers loaded with useless data, without violating minimality. Note that the same issue arises in other register-based models, such as the string register algebra. On the other hand, non-linear updates should be allowed. Consider the set U consisting of the two updates $() \mapsto a(u_1, u_2)$ and $() \mapsto b(u_2, u_1)$, for distinct terms u_1, u_2. If only linear updates were allowed, then $() \mapsto u_1$ and $() \mapsto u_2$ would be incomparable maximal common divisors of U, so U would have no greatest common divisor, and hence, by Theorem 24, a minimal upward STT need not exist. With non-linear updates, instead, U admits a greatest common divisor, for example,

$() \mapsto (u_1, u_2)$; indeed the latter update can be composed with $(x_1, x_2) \mapsto a(x_1, x_2)$ and $(x_1, x_2) \mapsto b(x_2, x_1)$ to recover respectively the two original updates of U.

Based on the above observations, two variants of the free term algebra remain admissible: one with copyful, non-erasing updates, and one with copyless, non-erasing updates.

Next, we show that Assumption 1 (compactness of constraints) holds over both variants of the free term algebra. We do so by embedding the free term algebra into the polynomial register algebra, and by relying on Hilbert's finite basis theorem [42].

▶ **Lemma 31 (compactness).** *Assumption 1 holds for the free term algebra, both in the variant that includes copyful non-erasing updates and in the variant that restricts to copyless non-erasing updates.*

Together with Propositions 14 and 15, this gives the existence of the initial image $\mathrm{Reach}(B)$ for every transducer B. We now show that $\mathrm{Reach}(B)$ is computable when B is finite. The key ingredient is the classical theory of unification in the free term algebra [57, 48], which provides finite representations of solutions to systems of equations, and therefore allows one to reason about them effectively. We provide the important definitions below.

Let E be a system of equations over variables $\bar{x} = (x_1, \ldots, x_\alpha)$. A *unifier* of E is an α-tuple[7] $\bar{u}$ of terms over fresh variables $\bar{y} = (y_1, \ldots, y_\beta)$, called *parameters*, such that every solution of E is of the form $\bar{u}[\bar{y}/\bar{t}]$ for some β-tuple $\bar{t}$ of ground terms. A unifier $\bar{u}$ is *most general* (*MGU*) if every other unifier $\bar{v}$ factors through it, namely, $\bar{v} = \bar{u}[\bar{y}/\bar{w}]$ for some tuple $\bar{w}$ of terms over the parameters of $\bar{v}$. Since an MGU, when it exists, is unique up to renaming of parameters, we shall often speak of *the* MGU of E. For example, the MGU of the system of equations $a(b(x_1), x_2, x_3) = a(x_2, b(x_1), x_3)$ and $a(c(x_1), x_2, x_3) = a(x_3, x_2, c(x_1))$ is the tuple $\bar{u} = (y_1, b(y_1), c(y_1))$ over the single parameter y_1.

The next lemma is a classical result of the theory of unification in the free term algebra [57, 48]. It shows that the solutions of a finite system of equations, if they exist, can be effectively represented in a solved (parametric) form —essentially an MGU— and this enables deciding entailment between systems of equations:

▶ **Lemma 32 ([57, Unification Theorem], [48, Theorem 2.3]).** *One can decide whether a given finite system of equations E over $\bar{x}$ is satisfiable (i.e. whether $\mathrm{Sol}(E) \neq \varnothing$), and in this case compute an MGU $\bar{u}$ of E with parameters $\bar{y}$ such that*

$$\mathrm{Sol}(E) \;=\; \big\{\bar{u}[\bar{y}/\bar{w}] \;:\; \bar{w} \text{ tuple of arbitrary ground terms}\big\}.$$

Moreover, E entails an equation $\bar{t} = \bar{t}'$ over $\bar{x}$ iff $\bar{t}[\bar{x}/\bar{u}]$ and $\bar{t}'[\bar{x}/\bar{u}]$ are syntactically equal.

Besides implying decidability of entailment between systems of equations, the above lemma is also crucial for computing $\mathrm{Reach}(B)$ from a given finite transducer B. The computability of $\mathrm{Reach}(B)$ indeed relies on the ideas outlined at the end of Section 5.4, and in particular on a variant of an algorithm from [45] for constructing inductive invariants in the free term algebra. The algorithm boils down to repeatedly computing closures of finite unions of constrained domains and of images of constrained domains via updates, which is possible thanks to Lemma 32.

▶ **Lemma 33 (effective initial images).** *Given a finite upward STT B, one can compute its initial image $\mathrm{Reach}(B)$.*

[7] In [57, 48], a unifier is defined as a mapping from variables to terms. Here we use an equivalent definition based on a tuple of terms, where the input variables are omitted but can be recovered from the arity.

Finally, we prove the existence of GCDs for sets of updates over the free term algebra with *copyless*, non-erasing updates; later, we show that GCDs may fail for the copyful, non-erasing variant. The key step is an interpolation property for the divisor relation: for any two updates f, f' with the same domain, there is an update g such that f and f' both divide g, and such that g divides every update h divided by both f and f'. This interpolation property relies on two crucial facts: (1) the divisor preorder can be characterized as a suitable embedding relation between multisets of terms, and (2) overlapping occurrences of subterms are necessarily nested. Once interpolation is established, one constructs a GCD of a set U of updates by considering the set G of all common divisors of U: any two maximal elements of G, with respect to the divisor preorder, turn out to divide one another, and hence are both *greatest* common divisors of U.

▶ **Lemma 34 (GCDs).** *Assumption 2 holds over the free term algebra with copyless and non-erasing updates. Moreover, GCDs can be computed for finite sets of updates.*

As usual, the minimization result follows from the previous lemmas and from Theorems 22 and 23. As for effectiveness, given a finite upward STT A, one can compute the EGCDs associated with its control states, by using (1) the fixpoint algorithm described at the end of Section 5.4 and (2) the effectiveness of GCDs of finite sets. We recall that this enables a normalization of A into an equivalent transducer B. We also recall from Lemma 33 that one can compute $C = \mathrm{Reach}(B)$. Finally, states with $\cong$-equivalent residual vectors can be detected, and then merged, because (1) for any two vectors V, V', we have $V \cong V'$ iff there is a permutation π of the components of the tuples of the domain of V such that $V[t](d_1, \ldots, d_n) = V'[t](d_{\pi(1)}, \ldots, d_{\pi(n)})$, for all $t \in \Sigma^*$ with $V[t] \neq \bot$ and all $(d_1, \ldots, d_n) \in \mathrm{Dom}(V[t])$, (2) the latter condition is a variant of an equivalence problem, which can be decided using the standard techniques described at the end Section 5.4, and (3) there are only finitely many such permutations π, which can then be verified one by one.

▶ **Theorem 35 (minimal upward STTs).** *Upward STTs, with updates restricted to be copyless and non-erasing, admit algebraically minimal transducers. Moreover, minimization can be performed effectively on any given finite transducer.*

Concerning the possibility of a minimization result for upward STTs with *copyful* updates, in view of Theorem 24, it suffices to show that GCDs not always exist. Consider the updates $f, f', g \in \mathbb{D}_{1\to 2}$ and $g' \in \mathbb{D}_{1\to 3}$ defined by

$$\begin{array}{llll} f: & x \mapsto \big(a(x), b(a(x))\big) & \qquad g: & x \mapsto \big(b(a(x)), c(a(x))\big) \\ f': & x \mapsto \big(a(x), c(a(x))\big) & \qquad g': & x \mapsto \big(a(x), b(a(x)), c(a(x))\big). \end{array}$$

We have that both f and f' are common divisors of $U = \{g, g'\}$, but they are not one a divisor or another (this is because we forbid erasing updates). If a GCD h existed for $U = \{g, g'\}$, then this would necessarily be divided by f and f'. Moreover, since neither $b(a(x))$ nor $c(a(x))$ can be obtained from the other by an update, h is also divided by g. Since g' is divided by h, we obtain by transitivity that g' is also divided by g. However, we see that this is not possible, because no update can produce the additional term $a(x)$ of g' without having either x or $a(x)$ in the input (here we are relying on the fact that the variable x can take at least two different values).

We conclude by returning to the minimization result for copyless, upward STTs, in order to highlight a connection with resource minimization. This setting is indeed an example where algebraic minimization partially coincides with resource minimization. It is immediate to see that a finite, algebraically minimal transducer M has a minimum number of control

states among all equivalent transducers: this holds for all models because, by definition, M is a subquotient of every equivalent transducer, and neither a subobject nor a quotient can have a larger number of control states. In the specific case of upward STTs, where every data is a tuple of terms, it is natural to think of each position of the tuple as a register, and thus wonder whether algebraic minimization achieves also register minimization. This holds only to some extent. Indeed, an algebraically minimal upward STT can have a control state whose constrained domain forces the value of a certain register to be constant; this register can then be removed while preserving equivalence. Nevertheless, it can be shown that every algebraically minimal upward STT optimizes, at each control state, the number of *non-constant* registers among all equivalent transducers with the same control structure.

7 Conclusion

We have presented sufficient and necessary conditions for the existence of algebraically minimal models of streaming transducers over a general data structure $\mathbb{D}$. The first condition is the possibility of enriching $\mathbb{D}$ with constrained domains, defined by systems of equations, providing the closure properties needed to construct images of transducer morphisms. The second condition is the existence of greatest common divisors for non-empty sets of updates, providing the algebraic ingredient needed to construct the final object underlying minimization.

After developing the abstract framework, we instantiated it to concrete data structures and obtained effective minimization results for two variants of string-to-term transducers, which construct their outputs incrementally by extending terms either at leaves (downward STTs) or at the roots (upward STTs). In both cases, the minimization procedure can be understood as a non-trivial generalization of Choffrut's minimization of sequential transducers [24], with anti-unification and unification playing the respective roles needed to compute quotients and subobjects.

Beyond the minimization results established here, the proposed framework opens several directions for further investigation. First, our results suggest possible applications to logical characterizations. Specifically, based on the characterization of the first-order fragment of string-to-string order-preserving transductions in [37], corresponding to sequential transducers with regular look-ahead, it seems plausible that the minimization results developed here could be used to obtain analogous characterizations for STTs with regular look-ahead.

A second direction seeks a common generalization of downward and upward STTs within a richer model that still admits minimization. One possible route is to allow updates defined by second-order term substitutions, or by suitable restricted variants of them. Along this line, the use of GCDs in our framework appears to connect with open problems on anti-unification for second-order term substitution, also known as higher-order anti-unification [52, 11].

It would also be interesting to understand how far the present approach extends to other data structures, such as the polynomial register algebra or the string register algebra. In this direction, we already have preliminary minimization results for transducers with univariate polynomial updates, based on Ritt's decomposition theory of polynomials [55, 58]. Another related question is which equational theories can be imposed on the free term algebra while preserving the existence of GCDs. For instance, such theories could allow strings to be represented as terms modulo associativity of concatenation.

Finally, the algorithmic aspects of the theory deserve further study. In particular, while our results give effective minimization procedures for the classes considered here, the complexity of computing minimal transducers from finite ones remains to be analyzed systematically.

References

1 J. Adámek, H. Herrlich, and G.E. Strecker. *Abstract and Concrete Categories: The Joy of Cats*. John Wiley & Sons, 1990. URL: http://katmat.math.uni-bremen.de/acc.

2 R. Alur and P. Cerný. Expressiveness of streaming string transducer. In *FSTTCS'10*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

3 R. Alur and L. D'Antoni. Streaming tree transducers. *J. ACM*, 64(5):31:1–31:55, 2017. doi:10.1145/3092842.

4 R. Alur, L. D'Antoni, J. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *LICS'13*, pages 13–22. IEEE, 2013.

5 R. Alur and M. Raghothaman. Decision problems for additive regular functions. In *ICALP'20*, volume 7966 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2013. URL: https://arxiv.org/abs/1304.7029, doi:10.1007/978-3-642-39212-2_8.

6 D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.

7 Q. Aristote. Functorial approach to minimizing and learning deterministic transducers with outputs in arbitrary monoids. 2024. URL: https://ens.hal.science/hal-04172251.

8 Q. Aristote. Active learning of deterministic transducers with outputs in arbitrary monoids. *Logical Methods in Computer Science*, 21(4), 2025. doi:10.46298/lmcs-21(4:7)2025.

9 M. Balachander, E. Filiot, R. Gentilini, and N. Tzevelekos. Register automata with permutations. In P. Gawrychowski, F. Mazowiecki, and M. Skrzypczak, editors, *MFCS'25*, volume 345 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. URL: 10.4230/LIPIcs.MFCS.2025.14, doi:10.4230/LIPICS.MFCS.2025.14.

10 F. Baschenis, O. Gauwin, A. Muscholl, and G. Puppis. Minimizing resources of sweeping and streaming string transducers. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *ICALP'16*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2016.114, doi:10.4230/LIPIcs.ICALP.2016.114.

11 A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Higher-order pattern anti-unification in linear time. *Journal of Automated Reasoning*, 58(2):293–310, 2017. doi:10.1007/s10817-016-9383-3.

12 J.P. Bell and D. Smertnig. Noncommutative rational pólya series. *Selecta Mathematica*, 27(3), 2021. doi:10.1007/s00029-021-00629-2.

13 J.P. Bell and D. Smertnig. Computing the linear hull: Deciding deterministic? and unambiguous? for weighted automata over fields. In *LICS'23*, pages 1–13. IEEE, 2023. URL: https://arxiv.org/abs/2209.02260, doi:10.1109/LICS56636.2023.10175691.

14 Y.I. Benalioua, N. Lhote, and P.-A. Reynier. Minimizing cost register automata over a field. In R. Královič and A. Kučera, editors, *MFCS'24*, volume 306 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2024.23, doi:10.4230/LIPIcs.MFCS.2024.23.

15 Y.I. Benalioua, N. Lhote, and P.-A. Reynier. Minimizing streaming string transducers: An algebraic approach, 2026. arXiv:2604.11567.

16 M. Benedikt, T. Duff, A. Sharad, and J. Worrell. Polynomial automata: Zeroness and applications. In *LICS'17*, pages 1–12. IEEE, 2017.

17 M. Benedikt, C. Ley, and G. Puppis. What you must remember when processing data words. In *PODS'10*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

18 A. Boiret, R. Piórkowski, and J. Schmude. Reducing transducer equivalence to register automata problems solved by Hilbert method. In *FSTTCS'18*, volume 122 of *Leibniz International*

Proceedings in Informatics (LIPIcs), pages 48:1–48:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

19 M. Bojańczyk. Transducers with origin information. In *ICALP'14*, number 8572 in Lecture Notes in Computer Science, pages 26–37. Springer, 2014.

20 M. Bojańczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. *LMCS*, 10(3), 2014.

21 J.W. Carlyle and A. Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

22 O. Carton, T. Colcombet, and G. Puppis. An algebraic approach to MSO-definability on countable linear orderings. *Journal of Symbolic Logic*, 83(3):1147–1189, 2018.

23 S. Cassel, F. Howar, B. Jonsson, and B. Steffen. Active learning for extended finite state machines. *Form. Asp. Comput.*, 28(2):233–263, 2016. doi:10.1007/s00165-016-0355-5.

24 C. Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.

25 T. Colcombet, C. Ley, and G. Puppis. Logics with rigidly guarded data tests. *Logical Methods in Computer Science*, http://arxiv.org/abs/1410.2022, 2015.

26 T. Colcombet and D. Petrisan. Automata minimization: a functorial approach. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:32)2020.

27 T. Colcombet, D. Petrisan, and R. Stabile. Learning automata and transducers: A categorical approach. In C. Baier and J. Goubault-Larrecq, editors, *CSL'21*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CSL.2021.15.

28 T. Colcombet, S. van Gool, and R. Morvan. First-order separation over countable ordinals. In P. Bouyer and L. Schröder, editors, *FoSSaCS'20*, volume 13242 of *Lecture Notes in Computer Science*, pages 264–284. Springer, 2022. doi:10.1007/978-3-030-99253-8_14.

29 B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Transformations: Foundations*, volume 1, pages 165–254. World Scientific, 1997.

30 D.A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 4 edition, 2015.

31 K. Culik II and J. Karhumäki. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theor. Comput. Sci.*, 47:71–84, 1986.

32 L. Daviaud, P.-A. Reynier, and J.-M. Talbot. A generalised twinning property for minimisation of cost register automata. In *LICS'16*, pages 857–866. ACM, 2016. doi:10.1145/2933575.2934549.

33 C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

34 S. Eilenberg. *Automata, Languages, and Machines. Volume B*. Academic Press, 1976.

35 S. Eilenberg and M.-P. Schützenberger. On pseudovarieties. *Advances in Mathematics*, 19(3):413–418, 1976. doi:10.1016/0001-8708(76)90029-3.

36 J. Eisner. Simpler and more general minimization for weighted finite-state automata. In *NAACL-HLT'03*, NAACL'03, pages 64–71. Association for Computational Linguistics, 2003. doi:10.3115/1073445.1073454.

37 E. Filiot, O. Gauwin, and N. Lhote. First-order definability of rational transductions: An algebraic approach. In *LICS'16*, pages 387–396. ACM, 2016. doi:10.1145/2933575.2934520.

38 E. Filiot, S.N. Krishna, and A. Trivedi. First-order definable string transformations. In V. Raman and S.P. Suresh, editors, *FSTTCS'20*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 147–159. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICS.FSTTCS.2014.147.

39 M. Fliess. Matrices de hankel. *Journal de Mathématiques Pures et Appliquées*, 53(9):197–222, 1974. Corrigendum: Journal de Mathématiques Pures et Appliquées, vol. 54, p. 481, 1975.

40 S. Gerdjikov. A general class of monoids supporting canonisation and minimisation of (sub)sequential transducers. In S.T. Klein, C. Martín-Vide, and D. Shapira, editors, *Language and Automata Theory and Applications*, pages 143–155. Springer, 2018. doi:10.1007/978-3-319-77313-1_11.

41 J.A. Goguen. Minimal realization of machines in closed categories. *Bulletin of the American Mathematical Society*, 78(5):777–783, 1972.

42 D. Hilbert. Ueber die theorie der algebraischen formen. *Mathematische Annalen*, 36(4):473–531, 1890. doi:10.1007/BF01208378.

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

44 E. Hrushovski, J. Ouaknine, A. Pouly, and J. Worrell. On strongest algebraic program invariants. *J. ACM*, 70(5), 2023. doi:10.1145/3614319.

45 M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6(2):157–169, 1976. doi:10.1007/BF00268497.

46 G. Laurence. *Normalisation et apprentissage de transductions d'arbres en mots*. PhD thesis, École doctorale Sciences pour l'Ingénieur (Lille), 2014. Thèse de doctorat. URL: https://hal.science/tel-01053084/.

47 A. Maletti. Myhill-Nerode theorem for sequential transducers over unique GCD-monoids. In *International Conference on Implementation and Application of Automata*, pages 323–324. Springer, 2004.

48 A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982. doi:10.1145/357162.357169.

49 J. Moerman, M. Sammartino, A. Silva, B. Klin, and M. Szynwelski. Learning nominal automata. In G. Castagna and A.D. Gordon, editors, *POPL'17*, pages 613–625. ACM, 2017. doi:10.1145/3009837.3009879.

50 A. Muscholl and G. Puppis. Equivalence of finite-valued streaming string transducers is decidable. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *ICALP'19*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 122:1–122:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.122.

51 A. Muscholl and G. Puppis. The many facets of string transducers (invited paper). In *STACS'19*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:22. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

52 F. Pfenning. Unification and anti-unification in the calculus of constructions. In *LICS'91*, pages 74–85. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151632.

53 G.D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

54 J.C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence*, 5(1):135–151, 1970.

55 J.F. Ritt. Prime and composite polynomials. *Transactions of the American Mathematical Society*, 23(1):51–66, 1922. doi:10.2307/1988911.

56 R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993. doi:10.1006/inco.1993.1021.

57 J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965. doi:10.1145/321250.321253.

58 C. Corrales Rodrigáñez. A note on Ritt's theorem on decomposition of polynomials. *Journal of Pure and Applied Algebra*, 68:293–296, 1990. doi:10.1016/0022-4049(90)90086-W.

59 O. Saarikivi and M. Veanes. Minimization of symbolic transducers. In R. Majumdar and V. Kuncak, editors, *CAV'20*, volume 10427 of *Lecture Notes in Computer Science*, pages 176–196. Springer, 2017. doi:10.1007/978-3-319-63390-9_10.

60 M.-P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

61 M.P. Schützenberger. Sur une variante des fonctions sequentielles. *TCS*, 4(1):47–57, 1977. doi:10.1016/0304-3975(77)90055-X.

62 H. Seidl, S. Maneth, and G. Kemper. Equivalence of deterministic top-down tree-to-string transducers is decidable. *J. ACM*, 65(4), 2018. doi:10.1145/3182653.

63 M. Shahbaz and R. Groz. Inferring mealy machines. In A. Cavalcanti and D. Dams, editors, *Formal Methods, Second World Congress (FM)*, volume 5850 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2009. doi:10.1007/978-3-642-05089-3_14.

64 J.M. Vilar. Query learning of subsequential transducers. In L. Miclet and C. de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings*, volume 1147 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996. doi:10.1007/BFB0033343.

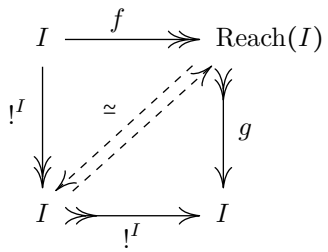
A Proofs for Section 3

▶ **Proposition 4 (variation of [26, Lemma 3.5]).** *Under the previous assumptions that enable the constructions* $\mathrm{Reach}(-)$ *and* $\mathrm{Obs}(\mathrm{Reach}(-))$*, we have that*

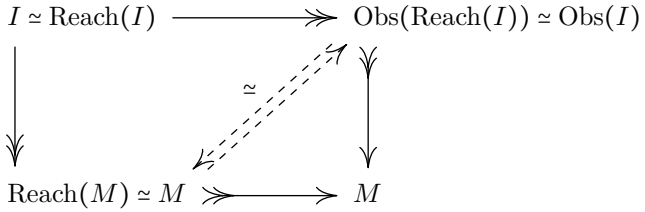
1. $I \simeq \mathrm{Reach}(I)$,
2. $M \simeq \mathrm{Obs}(I)$,
3. $M \simeq \mathrm{Obs}(\mathrm{Reach}(A))$ *for every object* A.

In particular, $\mathrm{Obs}(\mathrm{Reach}(A))$ *is a subquotient of* A*, and because* $M \simeq \mathrm{Obs}(\mathrm{Reach}(A))$*,* M *is a subquotient of every* A*, hence an algebraically minimal object.*

Proof. The first property $I \simeq \mathrm{Reach}(I)$ follows from considering a strong factorization of the initial arrow $!^I : I \to I$, which is also an identity morphism. This gives the initial image $\mathrm{Reach}(I)$ together with an epi $f : I \twoheadrightarrow \mathrm{Reach}(I)$ and a strong mono $g : \mathrm{Reach}(I) \rightarrowtail I$. One can then apply twice the diagonal fill-in property to the emerging diagram:

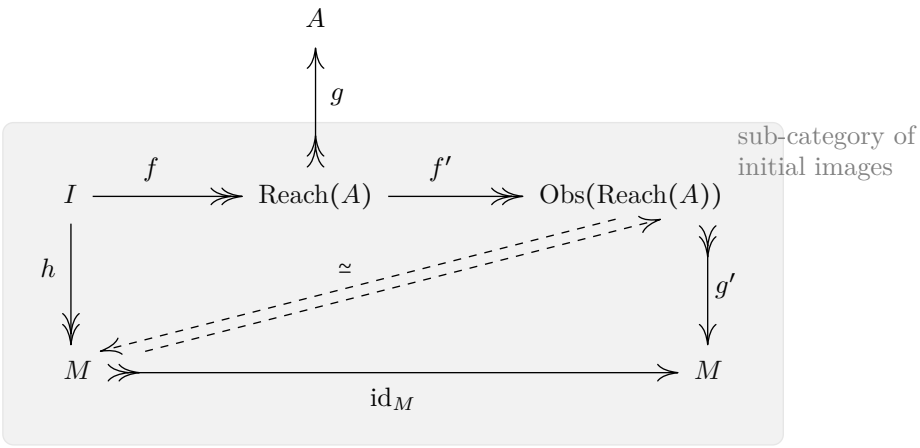


Similarly, the second property $M \simeq \mathrm{Obs}(I)$ follows from considering a strong factorization of the unique morphism h from I to M, together with the induced image. Since the latter morphism h can be interpreted both as the initial arrow for M and as the final arrow for I ($\simeq \mathrm{Reach}(I)$), we obtain that the images $\mathrm{Reach}(M)$ and $\mathrm{Obs}(I)$ are isomorphic. Moreover, we also have $M \simeq \mathrm{Reach}(M)$, simply because M is a final object in the sub-category of initial images, and hence $M \simeq \mathrm{Reach}(M) \simeq \mathrm{Obs}(I)$. These arguments can be equally seen as consequences of the diagonal fill-in property applied to the following diagram:



Finally, for an arbitrary object A, the property $M \simeq \mathrm{Obs}(\mathrm{Reach}(A))$ follows from considering, first, a strong factorization $I \overset{f}{\twoheadrightarrow} \mathrm{Reach}(A) \overset{g}{\rightarrowtail} A$ for the unique initial arrow $!^A : I \to A$, and then another strong factorization $\mathrm{Reach}(A) \overset{f'}{\twoheadrightarrow} \mathrm{Obs}(\mathrm{Reach}(A)) \overset{g'}{\rightarrowtail} M$ for the unique final arrow $!_{\mathrm{Reach}(A)} : \mathrm{Reach}(A) \to M$. By recalling that there is also an epi $h : I \simeq \mathrm{Reach}(I) \twoheadrightarrow \mathrm{Obs}(\mathrm{Reach}(I)) \simeq M$, we get the diagram below, where the diagonal fill-in

property implies $M \simeq \mathrm{Obs}(\mathrm{Reach}(A))$:



◀

B Proofs for Section 4

▶ **Lemma 7 (closures of constrained domains).** *Constrained domains are effectively closed under inverses of updates and under finite intersections. They are also closed (non-effectively) under infinite intersections.*

Proof. To prove closure of constrained domains under inverses of updates, consider a set $E \subseteq \mathbb{D}_{\beta\to\gamma} \times \mathbb{D}_{\beta\to\gamma}$ representing a system of equations inducing a constrained domain $\mathrm{Sol}(E)$ over the basic type β. Next, consider an update $f \in \mathbb{D}_{\alpha\to\beta}$. For an arbitrary data d of type α, we have $d \in f^{-1}(\mathrm{Sol}(E))$ iff $f(d) \in \mathrm{Sol}(E)$. This means that the set $f^{-1}(\mathrm{Sol}(E))$ contains all and only the solutions of the following system of equations:

$$E' = \{(f;u, f;v) \,:\, (u,v) \in E\}.$$

In particular, $f^{-1}(\mathrm{Sol}(E))$ is a constrained domain over the basic type α represented by system E', which can be computed from E and f.

Closure of constraints under intersections is straightforward, as taking the intersection of constraints corresponds to taking the conjunction of corresponding systems of equations. Moreover, if the intersection is finite, then the conjunction is also finite and hence computable. ◀

▶ **Lemma 10 (closure vs union).** *For all sets* $D, D' \subseteq \mathbb{D}_\tau$, $\mathrm{cl}(D \cup D') = \mathrm{cl}\big(\mathrm{cl}(D) \cup \mathrm{cl}(D')\big)$.

Proof. By extensiveness, we have $D \cup D' \subseteq \mathrm{cl}(D) \cup \mathrm{cl}(D')$, and hence, by monotonicity, $\mathrm{cl}(D \cup D') \subseteq \mathrm{cl}(\mathrm{cl}(D) \cup \mathrm{cl}(D'))$. Conversely, $D \subseteq D \cup D'$ and $D' \subseteq D \cup D'$ imply, by monotonicity, $\mathrm{cl}(D) \cup \mathrm{cl}(D') \subseteq \mathrm{cl}(D \cup D')$, and hence $\mathrm{cl}(\mathrm{cl}(D) \cup \mathrm{cl}(D')) \subseteq \mathrm{cl}(\mathrm{cl}(D \cup D'))$. Finally, idempotency implies $\mathrm{cl}(\mathrm{cl}(D \cup D')) = \mathrm{cl}(D \cup D')$. ◀

C Proofs for Section 5

▶ **Lemma 11 (closure-continuity of transformations).** *For every transformation* $f : \langle Q\rangle \to \langle Q'\rangle$ *and every subset* S *of* $\langle Q\rangle$*, we have* $f(\mathrm{cl}(S)) \subseteq \mathrm{cl}(f(S))$.

Proof. We first prove closure-continuity for updates, namely, that for every $f : \mathbb{D}_\alpha \to \mathbb{D}_\beta$ and every $D \subseteq \mathbb{D}_\alpha$,

$$f(\mathrm{cl}(D)) \subseteq \mathrm{cl}(f(D)).$$

For this, we are going to exploit the definition of closure of a set as the intersection of all constrained domains that contain that set. So, consider an arbitrary constrained domain $E \subseteq \mathbb{D}_\beta$ that contains $f(D)$. By Lemma 7, $f^{-1}(E)$ $(\subseteq \mathbb{D}_\alpha)$ is also a constrained domain. Moreover, $D \subseteq f^{-1}(E)$, because $d \in D$ implies $f(d) \in f(D) \subseteq E$, and hence $d \in f^{-1}(E)$. By definition of $\mathrm{cl}(D)$ as the intersection of all constrained domains that contain D, we get $\mathrm{cl}(D) \subseteq f^{-1}(E)$. Applying f and using the general fact that $f(f^{-1}(E)) \subseteq E$, we obtain $f(\mathrm{cl}(D)) \subseteq f(f^{-1}(E)) \subseteq E$. Since the latter containment holds for all constrained domains $E \supseteq f(D)$, it also holds for their intersection, and hence $f(\mathrm{cl}(D)) \subseteq \mathrm{cl}(f(D))$.

Next, we lift the above property to transformations. More precisely, given $f : \langle Q \rangle \to \langle Q' \rangle$, specified by a pair $(\hat{f}, \check{f})$, and given $S \subseteq \langle Q \rangle$, we let $D_{S,q} = \{d \in \mathbb{D}_{\tau(q)} : (q, d) \in S\}$ and derive:

$$\begin{aligned}
f\big(\mathrm{cl}(S)\big) &= \bigcup_{q \in Q} f\big(\{q\} \times \mathrm{cl}(D_{S,q})\big) && \text{(by definition of closure)} \\
&= \bigcup_{q \in Q} \big\{\hat{f}(q)\big\} \times \check{f}\big(\mathrm{cl}(D_{S,q})\big) && \text{(by distributivity of } \cup \text{ and } \times) \\
&\subseteq \bigcup_{q \in Q} \big\{\hat{f}(q)\big\} \times \mathrm{cl}\big(\check{f}(D_{S,q})\big) && \text{(by closure-continuity of updates)} \\
&\subseteq \mathrm{cl}\Big(\bigcup_{q \in Q} \{\hat{f}(q)\} \times \check{f}(D_{S,q})\Big) && \text{(by subadditivity of closure)} \\
&= \mathrm{cl}\big(f(S)\big). && \text{(by definition of closure)}
\end{aligned}$$

Note that the opposite containment does not hold in general, since the closure operator does not commute with union. ◀

▶ **Lemma 12 (epi transformations).** *Let $f : \langle Q \rangle \to \langle Q' \rangle$ be a transformation specified by $(\hat{f}, \check{f})$. The following conditions are equivalent:*

- *f is epi,*
- $\mathrm{Cod}(f) \subseteq \mathrm{cl}(\mathrm{Rng}(f))$,
- *$\hat{f} : Q \to Q'$ surjective and $\mathbb{D}_{\tau(r)} \subseteq \mathrm{cl}\big(\bigcup_{q \in \hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\big)$ for every $r \in Q'$.*

Proof. We start by observing that

$$\mathrm{Cod}(f) = \bigcup_{r \in Q'} \bigcup_{q \in \hat{f}^{-1}(r)} \mathrm{Cod}(\check{f}(q))$$

and

$$\mathrm{Rng}(f) = \bigcup_{r \in Q'} \bigcup_{q \in \hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q)).$$

This implies the equivalence between the latter two conditions.

We now prove the implication from the first condition to the third. Suppose that f is epi and consider an arbitrary state $r \in Q'$. Let h, h' be the transformations over $\langle Q' \rangle$ specified, respectively, by $(\hat{h}, \check{h})$ and $(\hat{h}', \check{h}')$, where $\hat{h}$ and $\check{h}$ are both undefined on the entire set Q', $\hat{h}'$ (resp. $\check{h}'$) is defined only on r and maps it to the same control state r (resp. to the identity update over $\mathbb{D}_{\tau(r)}$). If $r \notin \mathrm{Rng}(\hat{f})$, we would have $f;h = f;h'$, and hence $h = h'$ because f is epi. Since this would contradict the definitions of h and h' and since r was chosen arbitrarily, we must conclude that $\hat{f}$ is surjective. To complete the proof in one

direction, we show that $\mathbb{D}_{\tau(r)} \subseteq \mathrm{cl}\big(\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\big)$. For the sake of brevity, let D be the constrained domain $\mathrm{cl}\big(\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\big)$. Suppose, by way of contradiction, that $\mathbb{D}_{\tau(r)} \not\subseteq D$, and let $d \in \mathbb{D}_{\tau(r)} \setminus D$. Since $d \notin D$, there exist a constrained domain D' and a pair of updates $u, v : \mathbb{D}_{\tau(r)} \to D'$ such that $u(d) \neq v(d)$, and yet $u(d') = v(d')$ for all $d' \in \bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))$. We construct from this two transformations $h, h' : \langle Q'\rangle \to \langle Q''\rangle$ such that $f;h = f;h'$. Formally, we let Q'' be the set that consists of a single control state q'' with associated data type $\mathbb{D}_{\tau(q'')} = D'$, and specify h and h' respectively by the pairs $(\hat{h}, \check{h})$ and $(\hat{h}', \check{h}')$, where

- $\hat{h}$ and $\hat{h}'$ are defined only on r and they both map r to q'',
- $\check{h}$ and $\check{h}'$ are defined only on r and they map r to the update u and v, respectively.

Note that h and h' agree on the images of the configurations $(r, d') \in \langle Q'\rangle$, for all $d' \in \bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))$, and hence, because the latter configurations are precisely the images of f, we have $f;h = f;h'$. On the other hand, by construction, we have $h(q,d) = (q'', u(d)) \neq (q'', v(d)) = h'(q,d)$, and because (q,d) is a configuration of $\langle Q'\rangle$ where both h and h' are defined, we have $h \neq h'$. This contradicts the fact that f is epi, and thus proves that $\mathbb{D}_{\tau(r)}$ is contained in $D = \mathrm{cl}\big(\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\big)$.

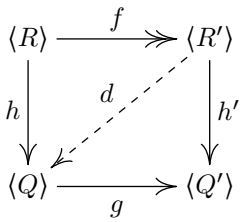
We now prove the converse direction, from the third condition to the first. Suppose that $\hat{f}$ is surjective and that $\mathbb{D}_{\tau(r)}$ is contained in the constrained domain $D = \mathrm{cl}\big(\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\big)$. We head towards proving that f is epi. Consider two transformations $h, h' : \langle Q'\rangle \to \langle Q''\rangle$, for an arbitrary configuration space $\langle Q''\rangle$, and suppose $f;h = f;h'$. Using Lemma 5, the latter equality can be rephrased in terms of the specifications $(\hat{h}, \check{h})$ and $(\hat{h}', \check{h}')$ of h and h':

- $\hat{f};\hat{h} = \hat{f};\hat{h}'$,
- $\check{f}(q);\check{h}(\hat{f}(q)) = \check{f}(q);\check{h}'(\hat{f}(q))$ for all $q \in Q$.

Now, since $\hat{f}$ is surjective, the first item above implies $\hat{h} = \hat{h}'$. Similarly, the second item implies that, for all $r \in Q'$, the two updates $\check{h}(r)$ and $\check{h}'(r)$ coincide on $\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))$. Moreover, because any two updates that agree on a set also agree on its closure, $\check{h}(r)$ and $\check{h}'(r)$ coincide also on $D = \mathrm{cl}\big(\bigcup_{q\in\hat{f}^{-1}(r)} \mathrm{Rng}(\check{f}(q))\big)$. Putting all together and recalling that $\mathbb{D}_{\tau(r)} \subseteq D$, we get $\check{h}(r) = \check{h}'(r)$ for all $r \in Q'$, and hence $h = h'$. Finally, because h, h' were chosen arbitrarily, we conclude that f is epi. ◀

▶ **Lemma 13 (strong mono transformations).** *Every inclusion map $g : \langle Q\rangle \to \langle Q'\rangle$, where $\langle Q\rangle \subseteq \langle Q'\rangle$ and $g(q,d) = (q,d)$ for all $(q,d) \in \langle Q\rangle$, is a strong mono.*

Proof. To prove that the inclusion map $g : \langle Q\rangle \to \langle Q'\rangle$ is a strong mono, we consider some transformations $f : \langle R\rangle \twoheadrightarrow \langle R'\rangle$, with f epi, $h : \langle R\rangle \to \langle Q\rangle$, and $h' : \langle R'\rangle \to \langle Q'\rangle$ such that $f;h' = h;g$, and we construct a transformation $d : \langle R'\rangle \to \langle Q\rangle$ such that $h = f;d$ and $h' = d;g$, as in the following diagram:



Intuitively, d is obtained from h' by restricting its codomain to $\langle Q\rangle$ (recall that $\langle Q\rangle \subseteq \langle Q'\rangle$). For this to make sense, we need to verify that $\mathrm{Rng}(h') \subseteq \langle Q\rangle$. Towards this, we observe that

1. since $f;h' = h;g$, we have $h'(f(\langle R\rangle)) = g(h(\langle R\rangle))$;

2. since g is an inclusion map, we have $g(h(\langle R\rangle)) = h(\langle R\rangle) \subseteq \langle Q\rangle$, and hence, by monotonicity, $\mathrm{cl}(g(h(\langle R\rangle))) \subseteq \mathrm{cl}(\langle Q\rangle) = \langle Q\rangle$;
3. since f is epi, by Lemma 12 we have $\langle R'\rangle \subseteq \mathrm{cl}(f(\langle R\rangle))$.

Putting all together and using Lemma 11 to pull out the closure, we get

$$\mathrm{Rng}(h') = h'(\langle R'\rangle) \subseteq h'(\mathrm{cl}(f(\langle R\rangle))) \subseteq \mathrm{cl}(h'(f(\langle R\rangle))) = \mathrm{cl}(g(h(\langle R\rangle))) \subseteq \langle Q\rangle.$$

We can now define d as h' with the codomain restricted to $\langle Q\rangle$. This can implemented at the level of specifications, as follows. Given a specification $(\hat{h}', \check{h}')$ of h', the specification of d is defined as $(\hat{d}, \check{d})$, where

- $\hat{d} : R' \to Q$ is defined by $\hat{d}(r) = \hat{h}'(r)$ for all $r \in R'$,
- given $r \in \mathrm{Dom}(\hat{h}')$ and $q = \hat{h}'(r)$, $\check{d}(r) : \mathbb{D}_{\tau_{R'}(r)} \to \mathbb{D}_{\tau_Q(q)}$ is the update obtained from $\check{h}'(r) : \mathbb{D}_{\tau_{R'}(r)} \to \mathbb{D}_{\tau_{Q'}(q)}$ by restricting the codomain $\mathbb{D}_{\tau_{Q'}(q)}$ to the set $\mathbb{D}_{\tau_Q(q)} = \mathrm{cl}\left(\bigcup_{q' \in \hat{h}'^{-1}(r)} \mathrm{Rng}(\check{h}'(q'))\right)$ (note that the state q is assigned the data domain $\mathbb{D}_{\tau_{Q'}(q)}$ or, equally, the constrained domain $\mathbb{D}_{\tau_Q(q)}$, depending on whether q is seen as an element of Q' or Q).

By construction, we have $h = f;d$ and $h' = d;g$.

To conclude the proof, it remains to show that the defined transformation d is the only possible one that makes the above diagram commute. This follows easily from the fact that f is an epi; indeed, for every other transformation $d' : \langle R'\rangle \to \langle Q\rangle$ such that $h = f;d'$, we have $f;d = f;d'$ and hence $d = d'$. ◀

▶ **Proposition 14 (images).** *Every transducer morphism $h : A \to B$ has a strong factorization $A \overset{f}{\twoheadrightarrow} C \overset{g}{\rightarrowtail} B$, with f epi and g strong mono.*

Proof. Let $A = (Q_A, \delta_\triangleright, (\delta_a)_{a\in\Sigma}, \delta_\triangleleft)$, $B = (Q_B, \kappa_\triangleright, (\kappa_a)_{a\in\Sigma}, \kappa_\triangleleft)$, and let $(\hat{h}, \check{h})$ be specification of the morphism h. We define the transducer $C = (Q_C, \chi_\triangleright, (\chi_a)_{a\in\Sigma}, \chi_\triangleleft)$ as a "pruning" of B:

- $Q_C = \mathrm{Rng}(\hat{h})$, which is a subset of Q_B. Each control state q in Q_C is assigned a new constrained type with the induced domain $\mathrm{cl}\left(\bigcup_{r\in\hat{h}^{-1}(q)} \mathrm{Rng}(\check{h}(r))\right)$. Note that the same control state q is assigned a possibly larger domain in Q_B, so we use explicit notation to distinguish the two domains: $\mathbb{D}_{\tau_{Q_B}(q)}$ and $\mathbb{D}_{\tau_{Q_C}(q)}$. Also observe that, by using the generalization of the closure operator to sets of configurations, we have $\langle Q_C\rangle = \mathrm{cl}\big(\mathrm{Rng}(h)\big)$.
- $\chi_\triangleright$, χ_a, and $\chi_\triangleleft$ are obtained from the corresponding transformations $\kappa_\triangleright$, κ_a, $\kappa_\triangleleft$ of B by replacing the domain/codomain $\langle Q_B\rangle$ with $\langle Q_C\rangle$. These replacements can be done at the level of specifications. For example, if $(\hat{\kappa}_a, \check{\kappa}_a)$ is the specification of the internal transformation κ_a of B, then the specification of the corresponding transformation χ_a of C is $(\hat{\chi}_a, \check{\chi}_a)$, where $\hat{\chi}_a$ is the restriction of $\hat{\kappa}_a$ to the set Q_C and $\check{\chi}_a$ maps every $q \in Q_C$ to the update $\check{\kappa}_a(q)$, with its domain and codomain restricted to $\mathbb{D}_{\tau_{Q_B}(q)}$ and $\mathbb{D}_{\tau_{Q_B}(\hat{\kappa}_a(q))}$, respectively. This definition guarantees not only $\mathrm{Dom}(\chi_a) \subseteq \langle Q_C\rangle$, but also $\mathrm{Rng}(\chi_a) \subseteq \langle Q_C\rangle$:

$$\begin{aligned}
\mathrm{Rng}(\chi_a) &= \chi_a(\langle Q_C\rangle) && \text{(by definition of range)}\\
&= \chi_a\big(\mathrm{cl}(h(\langle Q_A\rangle))\big) && \text{(by definition of } Q_C)\\
&\subseteq \mathrm{cl}\big(\chi_a(h(\langle Q_A\rangle))\big) && \text{(by Lemma 11)}\\
&= \mathrm{cl}\big(h(\delta_a(\langle Q_A\rangle))\big) && \text{(since } h \text{ is a morphism)}\\
&\subseteq \mathrm{cl}\big(h(\langle Q_A\rangle)\big) && \text{(since } \mathrm{Rng}(\delta_a) \subseteq \langle Q_A\rangle)\\
&= \langle Q_C\rangle. && \text{(again by definition of } Q_C)
\end{aligned}$$

We also define the transformation $f : \langle Q_A \rangle \to \langle Q_C \rangle$ as h, but with its codomain restricted to $\langle Q_C \rangle$. Lemma 12 immediately implies that f is an epi. Moreover, f is a transducer morphism from A to C: indeed, because h is a morphism from A to B and the internal transformation of C is a restriction of that of B, we have $f(\delta_a(q,d)) = h(\delta_a(q,d)) = \kappa_a(h(q,d)) = \chi_a(f(q,d))$, and similarly for $\delta_\triangleright$ and $\delta_\triangleleft$.

To conclude, we define the other part of the factorization of h, namely, the strong mono morphism g from C to B, so that $h = f;g$. The morphism g is simply the inclusion map from $\langle Q_C \rangle = \mathrm{cl}\big(\mathrm{Rng}(h)\big)$ to $\langle Q_B \rangle$. Formally, it is specified by the pair $(\hat{g}, \check{g})$, where $\hat{g} : Q_C \to Q_B$ maps every $q \in \mathrm{Rng}(\hat{h})$ to q itself, and $\check{g}$ maps every $q \in \mathrm{Rng}(\hat{h})$ to the identity update $\check{g}(q) : \mathbb{D}_{\tau_{Q_C}(q)} \to \mathbb{D}_{\tau_{Q_B}(q)}$. By Lemma 13, we know that g is a strong mono. ◀

▶ **Proposition 15 (initial transducer).** *There is an (infinite) transducer I, with words over Σ as control states, that is initial, namely, that admits a unique morphism $!^A : I \to A$ towards each transducer A that realizes φ.*

Proof. As claimed in the proposition, the set control states of I is Σ^*. All control states $w \in \Sigma^*$ are associated with the data type $\triangleright$, i.e. the same that is associated with $q_\triangleright$, and the corresponding domain is the singleton $\mathbb{D}_\triangleright = \{d_\triangleright\}$. Basically, I is an infinite-state transducer without registers. The transformations of I are also easily defined:

- $\delta_\triangleright$ maps $(q_\triangleright, d_\triangleright)$ to the configuration $(\varepsilon, d_\triangleright)$; formally, $\delta_\triangleright$ is specified by a pair of functions $(\hat{\delta}_\triangleright, \check{\delta}_\triangleright)$, where $\hat{\delta}_\triangleright$ (resp. $\check{\delta}_\triangleright$) maps $q_\triangleright$ to the control state ε (resp. to the identity update $d_\triangleright \mapsto d_\triangleright$);
- for every $a \in \Sigma$, δ_a maps any configuration $(w, d_\triangleright)$ to the configuration $(wa, d_\triangleright)$; this is specified by the pair $(\hat{\delta}_a, \check{\delta}_a)$, where $\hat{\delta}_a(w) = wa$ and $\check{\delta}_a(w)$ is again the identity update $d_\triangleright \mapsto d_\triangleright$;
- $\delta_\triangleleft$ maps any configuration $(w, d_\triangleright)$ for which $\varphi(w)$ is defined to the pair $(q_\triangleleft, \varphi(w))$, namely, $\hat{\delta}_\triangleleft(w) = q_\triangleleft$ and $\check{\delta}_\triangleleft(w) : d_\triangleright \mapsto \varphi(w)$; if $\varphi(w)$ undefined, then $\delta_\triangleleft(w, d_\triangleright)$ is undefined too.

This transducer clearly realizes φ. We also observe that the only component that depends on φ is the final transformation $\delta_\triangleleft$.

To prove that this is an initial object, consider any transducer $A = (\langle Q' \rangle, \kappa_\triangleright, (\kappa_a)_{a\in\Sigma}, \kappa_\triangleleft)$ that realizes the same transduction φ. We can construct an initial morphism $!^A : I \to A$ by simply mapping every configuration $(w, d_\triangleright)$ of I to the configuration $\kappa_{\triangleright w}(q_\triangleright, d_\triangleright)$ reached by A after reading w, assuming $\kappa_{\triangleright w}(q_\triangleright, d_\triangleright)$ is defined (otherwise, if it is not defined, we let $!^A(w, d_\triangleright)$ be undefined too). Formally, the morphism $!^A$ is specified by the pair $(\hat{!}^A, \check{!}^A)$ of partial functions that map any $w \in Q$ $(= \Sigma^*)$ respectively to the control state q_w of A and to the update $u_w : d_\triangleright \mapsto d_w$, assuming $(q_w, d_w) = \kappa_{\triangleright w}(q_\triangleright, d_\triangleright)$.

We omit the straightforward proof that $!^A$ is indeed a transducer morphism, and we show instead that $!^A$ is the *unique* possible morphism from I to A, essentially because the definition of $!^A$ was the only admissible one. We prove this by considering another possible morphism $!!^A : I \to A$ and by verifying that $!^A(w, d_\triangleright) = !!^A(w, d_\triangleright)$, using a simple induction on $|w|$. For the base case $w = \varepsilon$, we recall that $\delta_\triangleright;!^A = \kappa_\triangleright = \delta_\triangleright;!!^A$, and hence

$$\begin{aligned} !^A(\varepsilon, d_\triangleright) &= !^A(\delta_\triangleright(q_\triangleright, d_\triangleright)) \\ &= \kappa_\triangleright(q_\triangleright, d_\triangleright) \\ &= !!^A(\delta_\triangleright(q_\triangleright, d_\triangleright)) \\ &= !!^A(\varepsilon, d_\triangleright). \end{aligned}$$

For the inductive step, we assume that $!^A(w, d_\triangleright) = !!^A(w, d_\triangleright)$, we recall that $\kappa_a ; !^A = \delta_a ; !^A$ and $\kappa_a ; !!^A = \delta_a ; !!^A$, and we derive

$$\begin{aligned} !^A(wa, d_\triangleright) &= !^A(\delta_a(w, d_\triangleright)) \\ &= \kappa_a(!^A(w, d_\triangleright)) \\ &= \kappa_a(!!^A(w, d_\triangleright)) \\ &= !!^A(\delta_a(w, d_\triangleright)) \\ &= !!^A(wa, d_\triangleright). \end{aligned}$$

◀

▶ **Lemma 16 (GCD vs EGCD).** *If g is a GCD of a vector U, then there is a strong factorization $g = g' ; g''$ of it, where g' is an EGCD of U.*

Proof. Let U be a vector and g a GCD of U. By applying Proposition 14 to single-state transformations —which correspond to updates—, one obtains a strong factorization $g = g' ; g''$. Clearly, g' is an epi common divisor of U. To prove that g' is also greatest among all epi common divisors of U, consider another epi common divisor f of U. Since g is greatest among all common divisors of U, f is a divisor of g, so $g = f ; f'$ for some f'. Since f is an epi and g'' is a strong mono, by the diagonal fill-in property there is a (unique) update d such that $f ; d = g'$. This shows that f is also a divisor of g', and hence g' is an EGCD of U. ◀

▶ **Lemma 17 (uniqueness of EGCDs).** *The EGCDs of any given vector U are pairwise isomorphic.*

Proof. Suppose that g, g' are two EGCDs of the same vector U. Clearly, they are divisors one of another, and in particular, because g, g' are epis, they induce unique updates j, j' such that $g' = g ; j$ and $g = g' ; j'$. Let $d = j ; j'$ and $d' = j' ; j$, and observe that $g ; d = g$ and $g' = g' ; d'$. However, because, once again, g is an epi, there is only one residual of g via g, which is the identity. So d is the identity, and similarly for d'. This shows that j is an isomorphism from the codomain of g to the codomain of g', and j' is its inverse. Moreover, because $g ; j = g'$, we have that g and g' are isomorphic. ◀

▶ **Lemma 18 (distributivity of EGCDs).** *If $U = g ; V$, for some epi g and some vector V, and h is an EGCD of V, then $g ; h$ is an EGCD of U.*

Proof. We will reason with the following EGCDs, which exist thanks to prior assumptions:

- f is an EGCD of $U = g ; V$, with $T = f \backslash U$ associated residual vector,
- h is an EGCD of V, with $W = h \backslash V$ associated residual vector.

We aim at proving that $g ; h$ is an EGCD of U.

We start by observing that $U = g ; V = g ; h ; W$, so $g ; h$ is a common divisor of U, and it is epi since g and h are epis. It remains to show that $g ; h$ is a *greatest* epi common divisor of U. Recalling that f is an EGCD of U, we also derive that $g ; h$ is a divisor of f. Now, let $\ell = g ; h \backslash f$. Because every residual of an epi is an epi, (cf. [1, Proposition 7.41]), ℓ is an epi. Next, from the equation

$$g ; V \;=\; U \;=\; f ; T \;=\; g ; h ; \ell ; T,$$

we can cancel g from both sides, since it is epi, and obtain

$$V \;=\; h ; \ell ; T.$$

This shows that $h ; \ell$ is a divisor of V, and it is epi because h and ℓ are epi. Because h is an EGCD of V, we deduce that $h ; \ell$ is a divisor of h, and by pre-composing with g, we get that

$g;h;\ell$ ($= g;h;(g;h)\backslash f = f$) is a divisor of $g;h$. For the last stretch, every epi common divisor of U is a divisor of the EGCD f of U, and hence by transitivity it is also a divisor of $g;h$. Since we have already established that $g;h$ is an epi common divisor of U, this shows that $g;h$ is an EGCD of U. ◀

▶ **Corollary 19 (EGCDs of sub-vectors).** *Let g is an EGCD of U and let $V = g\backslash U$ be the associated residual vector. For every $a \in \Sigma$, if h is an EGCD of the sub-vector $V[a-]$, then $g;h$ is an EGCD of the sub-vector $U[a-]$.*

Proof. Let g is the EGCD of U, $V = g\backslash U$, $U' = U[a-]$, $V' = V[a-]$, and let h be the EGCD of V'. We have that g is epi and $U' = g;V'$, and so, by Lemma 18, $g;h$ is the EGCD of U'. ◀

▶ **Lemma 20 (right-invariance).** *Under Assumption 2, one can find some column vectors G and ∂_a (one for every letter $a \in \Sigma$), and a matrix R such that, for all $s \in \Sigma^*$ and $a \in \Sigma$:*

- *$G[s]$ is an EGCD of $H[s,-]$ and $R[s,-] = G[s]\backslash H[s,-]$,*
- *$\partial_a[s]$ is an EGCD of $R[s,a-]$ and $R[sa,-] = \partial_a[s]\backslash R[s,a-]$,*
- *$G[sa] = G[s];\partial_a[s]$.*

In particular, $G[s]$ and $R[s,-]$ are uniquely determined from $H[s,-]$, and $\partial_a[s]$ is uniquely determined from $R[s,-]$.

Proof. We construct our objects $G[s]$, $\partial_a[s]$, and $R[s,-]$ by exploiting an induction on $s \in \Sigma^*$, while preserving the properties stated in the lemma.

For the base case of the induction, we only define $G[\varepsilon]$ and $R[\varepsilon,-]$, as follows: we exploit the existence of EGCDs (which follows from Assumptions 1 and 2 and from Lemma 16) and we let $G[\varepsilon]$ be an EGCD of $H[\varepsilon,-]$ and $R[\varepsilon,-] = G[\varepsilon]\backslash H[\varepsilon,-]$. This trivially satisfies the first property of the lemma for $s = \varepsilon$.

For the inductive step, we consider a word $s \in \Sigma^*$ and a letter $a \in \Sigma$, and we define first $\partial_a[s]$ and $R[sa,-]$, and then $G[sa]$. In doing so we assume as inductive hypothesis that $G[s]$ and $R[s,-]$ are already defined. We then define $\partial_a[s]$ as an EGCD of the sub-vector $R[s,a-]$, and we let $R[sa,-] = \partial_a[s]\backslash R[s,a-]$. These definitions clearly satisfy the second property of the lemma. We also let $G[sa] = G[s];\partial_a[s]$, so as to satisfy the third property. By applying Corollary 19 with $U = H[s,-]$, $g = G[s]$, $V = R[s,-]$, and $h = \partial_a[s]$, we obtain that $g;h = G[sa]$ is an EGCD of $U[a-] = H[s,a-] = H[sa,-]$, and in particular we have

$$\begin{aligned}
G[sa];R[sa,-] &= G[s];\partial_a[s];R[sa,-] && \text{(by definition of } G[sa]\text{)}\\
&= G[s];R[s,a-] && \text{(by definition of } \partial_a[s]\text{)}\\
&= H[s,a-] = && \text{(by inductive hypothesis)}\\
&= H[sa,-]. && \text{(by definition of Hankel matrix)}
\end{aligned}$$

This shows that the first property is preserved during the induction step, and completes the proof. ◀

▶ **Proposition 21 (final transducer).** *There is a transducer M that is final in the sub-category of images of initial morphisms, namely, for every transducer A that realizes φ, there is a unique morphism $!_{\mathrm{Reach}(A)} : \mathrm{Reach}(A) \to M$.*

Proof. We shall use the column vectors G and ∂_a and the residual matrix R provided in Lemma 20. We will only consider the vectors of R that are not constantly $\bot$; we call these vectors *non-trivial*. Because we need to distinguish the residual vectors only up to the equivalence $\cong$ (defined after Lemma 17), it is convenient to fix a *representative* r_w for all the

words $w \in \Sigma^*$ that induce non-trivial residual vectors in the same $\cong$-class. We also fix an isomorphism j_w witnessing $R[w,-] \overset{j_w}{\cong} R[r_w,-]$.

The final transducer. We define the transducer $M = (Q, \delta_\triangleright, (\delta_a)_{a\in\Sigma}, \delta_\triangleleft)$, as follows:

- Q consists of the representatives r_w, now seen as control states with associated types; more precisely, the type $\tau(r_w)$ associated with each state r_w is the same as the type of the domain of any update $R[r_w,t]$, provided it is not $\bot$ — recall that the non-$\bot$ updates of a vector have all the same domain; in particular, we have $\mathbb{D}_{\tau(r_w)} = \mathrm{Dom}(R[r_w,\varepsilon])$;
- $\delta_\triangleright$ maps $(q_\triangleright, d_\triangleright)$ to (r_ε, d), where $d = G[r_\varepsilon](d_\triangleright)$, provided that $R[r_\varepsilon,-]$ is non-trivial — note that $G[r_\varepsilon];R[r_\varepsilon,-] = H[r_\varepsilon,-]$, and hence $G[r_\varepsilon]$ is an update from the singleton domain $\mathbb{D}_\triangleright = \{d_\triangleright\}$ to the domain $\mathbb{D}_{\tau(r_\varepsilon)}$;
- for every $a \in \Sigma$, δ_a maps any configuration (s,d) to the configuration (r_{sa}, d'), where $d' = (\partial_a[s];j_{sa})(d)$, provided that $R[r_{sa},-]$ is a non-trivial residual vector (otherwise, δ_a is undefined on (s,d)) — note that the target configuration (r_{sa},d') is well-defined because, by Lemma 20, $\partial_a[s]$ is uniquely determined from $R[s,-]$ and a and j_{sa} is, by construction, determined by sa;
- $\delta_\triangleleft$ maps any configuration (s,d) to $(q_\triangleleft, d')$, where $d' = R[s,\varepsilon](d)$ — note that, because $R[s,-]$ is non-trivial, $R[s,\varepsilon](d)$ is defined.

A simple induction on the length of w shows that, after reading w, the transducer M reaches the configuration

$$\big(r_w,\, G[r_w](d_\triangleright)\big)$$

provided that $R[r_w,-]$ is a non-trivial residual vector (otherwise, M has no run on w). By construction, j_w is an isomorphism such that $R[w,-] = j_w;R[r_w,-]$. Similarly, one verifies by induction that $G[r_w] = G[w];j_w$. Thus, the output produced by M after reading w is

$$\big(\underbrace{G[r_w]}_{G[w];j_w} ;\ \underbrace{R[r_w,\varepsilon]}_{j_w^{-1};R[w,\varepsilon]}\big)(d_\triangleright) \;=\; \big(G[w];R[w,\varepsilon]\big)(d_\triangleright) \;=\; H[w,\varepsilon] \;=\; \varphi(w).$$

This shows that M realizes exactly the transduction φ.

Existence of final morphism. We show that M is indeed final for the sub-category of initial images. Consider an arbitrary transducer A that also realizes φ, and let $!^A : I \to A$ be the unique initial morphism towards A (Proposition 15). Further let $I \overset{e}{\twoheadrightarrow} B \overset{m}{\rightarrowtail} A$ be a strong factorization of $!^A$ (Proposition 14), and let $(\hat{e}, \check{e})$ be the specification of the epi morphism e. We also need to denote the components of the initial transducer I and those of the initial image B, so we let $I = (\Sigma^*, \kappa_\triangleright, (\kappa_a)_{a\in\Sigma}, \kappa_\triangleleft)$ and $B = (Q_B, \chi_\triangleright, (\chi_a)_{a\in\Sigma}, \chi_\triangleleft)$.

We disclose some important consequences of the fact that e is epi:

1. Since e is epi, by Lemma 12, $\hat{e}$ is a surjective partial function, and hence for every $q' \in Q_B$, the set $\hat{e}^{-1}(q')$ is non-empty (this set contains control states of I, which are words over Σ). Hereafter, s is tacitly assumed to be an arbitrary word from $\hat{e}^{-1}(q')$ that induces a non-trivial residual vector $R[s,-]$.
2. For every $t \in \Sigma^*$, the output $\varphi(st)$ of B, seen as an update from the singleton domain $\mathbb{D}_\triangleright$, factorizes as $f_s;h_t$, where $f_s = \check{\chi}_{\triangleright s}(q_\triangleright)$ and $h_t = \check{\chi}_{t\triangleleft}(q')$, namely, f_s (resp. h_t) is the update induced by $\triangleright s$ (resp. $t\triangleleft$) in B starting from $q_\triangleright$ (resp. q'). Since e is a morphism from I to B, we also have $f_s = \check{e}(s)$.
3. Let $V_{q'} = (h_t)_{t\in\Sigma^*}$. Because B computes the transduction φ, we have

$$H[s,-] \;=\; \big(\varphi(st)\big)_{t\in\Sigma^*} \;=\; f_s;V_{q'}.$$

Moreover, by Lemma 20, $G[s]$ is an EGCD of $H[s,-]$ and $R[s,-]$ is the corresponding residual vector. We thus have

$$R[s,-] \;=\; G[s]\backslash(f_s;V_{q'}).$$

4. Now, let g_s be an EGCD of the vector $V_{q'}$, and recall from Lemma 17 that any two EGCDs of $V_{q'}$ are isomorphic. In particular, since $V_{q'}$ depends only on q', for all $s, s' \in \hat{e}^{-1}(q')$, there exists an isomorphism $i_{s,s'}$ between the codomains of g_s and $g_{s'}$ such that

$$g_s \;=\; g_{s'};i_{s,s'}.$$

5. Let $W_{s,q'} = g_s\backslash V_{q'}$ be the residual vector of $V_{q'}$ via the EGCD g_s. We have

$$H[s,-] \;=\; f_s;V_{q'} \;=\; f_s;W_{s,q'}$$

and similarly for $s' \in \hat{e}^{-1}(q')$. The isomorphism $i_{s,s'}$ witnessing $g_s \cong g_{s'}$ also induces a $\cong$-equivalence between the corresponding residual vectors:

$$R[s,-] \overset{i_{s,s'}}{\cong} R[s',-].$$

In particular, all the vectors $R[s,-]$ for $s \in \hat{e}^{-1}(q')$ lie in the same $\cong$-class.

We can now specify the final morphism $!_B : B \to M$. For the mapping $\hat{!}_B$ from the states of B to the states of M, we let

$$\hat{!}_B(q') \;=\; \begin{cases} r_w & \text{if there is } w \in \hat{e}^{-1}(q') \text{ with } R[w,-] \text{ non-trivial} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Note that this is well-defined because by Property (5) above, all words in $\hat{e}^{-1}(q')$ induce pairwise $\cong$-equivalent residual vectors, and so have the same representative. As for the mapping $\check{!}_B$ from states of B to updates, we recall from Properties (2–4) above that, for every $s \in \hat{e}^{-1}(q')$, g_s is an EGCD of the vector $V_{q'} = (h_t)_{t\in\Sigma^\star}$, which contains the updates $h_t = \check{\chi}_{t\triangleleft}(q')$ induced by B starting from q'. Accordingly, we let

$$\check{!}_B(q') \;=\; \begin{cases} g_{r_w} & \text{if there is } w \in \hat{e}^{-1}(q') \text{ with } R[w,-] \text{ non-trivial} \\ \text{undefined} & \text{otherwise} \end{cases}$$

(this is again well-defined thanks to Property (5)). By construction, the transformation $!_B : \langle Q_B\rangle \to \langle Q\rangle$ specified by the pair $(\hat{!}_B, \check{!}_B)$ satisfies the following property:

$$\begin{array}{ll} \text{if} & e(w, d_\triangleright) = (q', d) \text{ and } R[r_w,-] \text{ is non-trivial} \\ \text{then} & !_B(q', d) = (r_w,\, g_{r_w}(d)). \end{array} \qquad (\star)$$

Below, we show that $!_B$ is also a transducer morphism. We need to show that the solid parts of the diagrams in Figure 4 commute (as for the dashed parts, we already know they commute since e is a morphism from I to B). We are going to exploit Property $(\star)$ above and the fact that e is an epi. More precisely, for the left diagram, we have:

$$\begin{array}{lll} \chi_\triangleright ;!_B \;=\; \kappa_\triangleright ;e ;!_B & & (\text{since } e : I \to A) \\ \qquad\quad =\; \delta_\triangleright. & & (\text{by } (\star) \text{ and by definition of } M) \end{array}$$

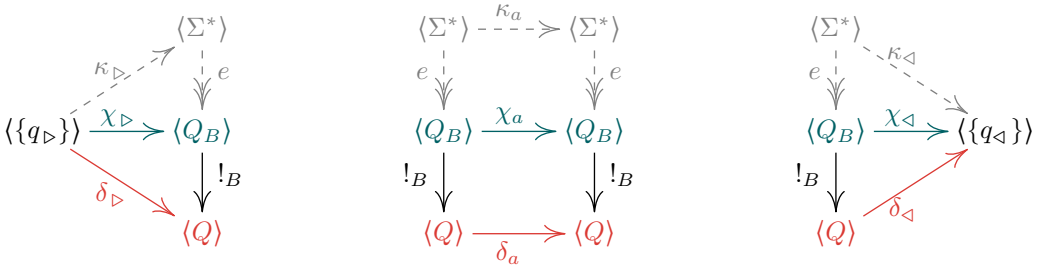


Figure 4 Diagrams describing the final morphism $!_B : B \to M$.

For the middle diagram, we have:

$$\begin{aligned} e\,;\chi_a\,;!_B &= \kappa_a\,;e\,;!_B && \text{(since } e : I \to A\text{)} \\ &= e\,;!_B\,;\delta_a && \text{(by } (\star) \text{ and by definition of } M\text{)} \\ \text{and hence}\quad \chi_a\,;!_B &= !_B\,;\delta_a. && \text{(because } e \text{ is epi)} \end{aligned}$$

Similarly, for the right diagram, we have:

$$\begin{aligned} e\,;\chi_\triangleleft &= \kappa_\triangleleft && \text{(since } e : I \to A\text{)} \\ &= e\,;!_B\,;\delta_\triangleleft && \text{(by } (\star) \text{ and by definition of } M\text{)} \\ \text{and hence}\quad \chi_\triangleleft &= !_B\,;\delta_\triangleleft. && \text{(because } e \text{ is epi)} \end{aligned}$$

This shows that $!_B$ is a transducer morphism from B to M.

Uniqueness of final morphism. Finally, we argue that $!_B$ is the unique possible transducer morphism from B to M. Consider another possible morphism $!!_B : B \to M$. A simple induction on $w \in \Sigma^*$ shows that $(e\,;!_B)(w, d_\triangleright) = (e\,;!!_B)(w, d_\triangleright)$. This shows that $e\,;!_B = e\,;!!_B$, and thus, since e is epi, $!_B = !!_B$. ◀

▶ **Theorem 24 (necessity of GCDs).** *Assumption 2 holds over any standard data structure $\mathbb{D}$ whenever, for every transduction φ, the class of transducers over $\mathbb{D}$ realizing φ contains an algebraically minimal model.*

Proof. We fix a standard data structure $\mathbb{D}$ and assume that every class of transducers over $\mathbb{D}$ realizing a certain transduction contains an algebraically minimal model. Towards proving that $\mathbb{D}$ satisfies Assumption 2, we consider some input and output types α and β and a set $U \subseteq \mathbb{D}_{\alpha\to\beta}$ of updates.

We exploit the fact that both sets $\mathbb{D}_\alpha$ and $\mathbb{D}_{\alpha\to\beta}$ are countable to define string-based encodings of the elements of these sets. Formally, we enumerate $\mathbb{D}_\alpha$ as $\{d_1, d_2, \dots\}$ and we encode each data d_i by the string $\tilde{d}_i = \bullet^i$, where $\bullet$ is a special symbol from the underlying alphabet. Similarly, we enumerate $\mathbb{D}_{\alpha\to\beta}$ as $\{u_1, u_2, \dots\}$ and we encode each update u_j by the string $\bullet^j$. We then assume that the underlying alphabet contains three other symbols, $\blacksquare, \blacktriangle, \blacktriangledown$, and define the transduction $\varphi : \{\bullet, \blacksquare, \blacktriangle, \blacktriangledown\}^* \to \mathbb{D}_\beta$ such that:

- φ maps every string of the form $\bullet^i\blacksquare\blacktriangle\bullet^j$, with $i, j \in \mathbb{N}$, to $u_j(d_i)$;
- φ maps every string of the form $\bullet^i\blacksquare\blacktriangledown\bullet^j$, with $i, j \in \mathbb{N}$ and $u_j \in U$, to $u_j(d_i)$;
- φ is undefined on all other inputs, in particular on inputs $\bullet^i\blacksquare\blacktriangledown\bullet^j$ such that $u_j \notin U$.

Now, for every common divisor f of U, we construct a transducer A_f that realizes φ, as shown in Figure 5 (some identity updates along transitions are omitted for readability). In particular, after reading a prefix $\tilde{d}\blacksquare$, A_f reaches a distinguished state q_f storing the data

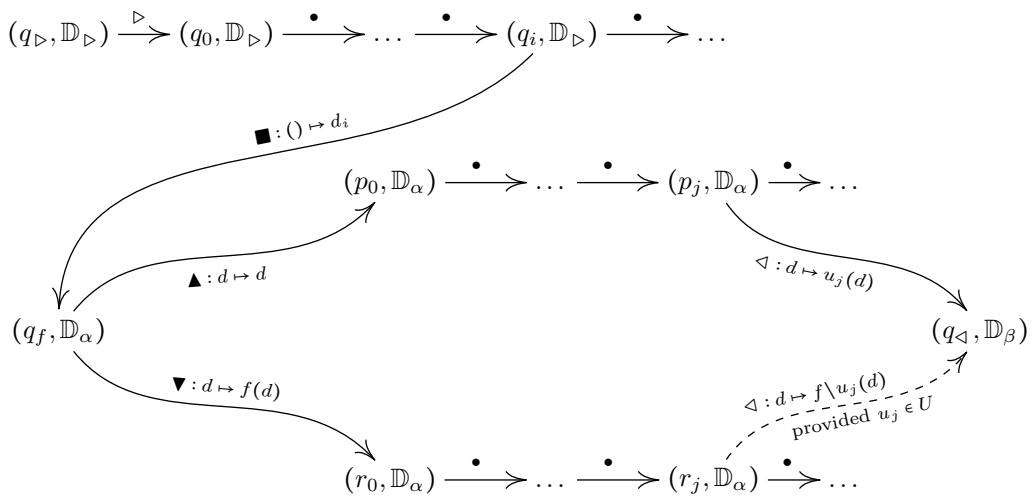


Figure 5 The transducer A_f that emits a common divisor f of U upon reading $\blacktriangledown$.

d encoded by the prefix. From q_f, a $\blacktriangle$-labelled transition applies the identity on $\mathbb{D}_\alpha$ and reaches a component that consumes suffixes $\tilde{u}$, eventually producing $u(d)$. Similarly, from q_f, a $\blacktriangledown$-labelled transition applies the update f and reaches a component that consumes suffixes $\tilde{u}$, eventually producing $u(d)$ only when $u \in U$.

Next, we exploit the existence of an algebraically minimal transducer M realizing the same transduction φ, and from which we will be able to extract a GCD of U. Since each A_f is pruned by design, i.e. $A_f = \mathrm{Reach}(A_f)$, M is a quotient of A_f, and hence there is a morphim h from A_f to M. In particular, this morphism maps the unique $\blacktriangledown$-labelled transition of A_f to a unique $\blacktriangledown$-labelled transition of M, whose update we denote by g. Clearly, this forces g to factor through every common divisor f of U.

It remains to see that g is a common divisor of U. For this, we show that the runs of M labelled by $\blacktriangle\tilde{u}$, for all $u \in \mathbb{D}_{\alpha\to\beta}$, ensure that no non-trivial common divisor can be accumulated before reading $\blacktriangledown$. Formally, we define the accumulated update before $\blacktriangledown$ as the update g_0 that maps any data $d \in \mathbb{D}_\alpha$ to the data of the configuration reached by M after reading the corresponding prefix $\tilde{d}\blacksquare$. By the commutativity properties of transducer morphisms, we know that g_0 must coincide with $\check{h}(q_f)$, namely, the update that is applied by the morphism h when applied to configurations of A_f with q_f as control state. Moreover, by construction, g_0 must divide all the updates $u \in \mathbb{D}_{\alpha\to\beta}$ induced by the possible continuations $\blacktriangle\tilde{u}$. Because the latter updates are jointly coprime, we derive that g_0 is isomorphic to the identity over $\mathbb{D}_\alpha$. Finally, knowing that $g_0; g$ must be a common divisor of U, we conclude that g is also a common divisor of U, and hence a GCD of U. ◀

D Proofs for Section 6

▶ **Corollary 26 (compactness).** *Assumption 1 holds trivially for the leaf substitution algebra.*

Proof. Given an equation $f(x) = g(x)$ in the leaf substitution algebra $\mathbb{D}$, with f and g defined by α-tuples of linear terms of type β, say $\bar{u}$ and $\bar{v}$, respectively, and given another linear term t of same type α, we have that $x = t$ is a solution of the equation iff $t[\bar{u}] = t[\bar{v}]$. By Lemma 25, every such equation is either vacuous or unsatisfiable. ◀

▶ **Corollary 27 (epi updates).** *Every update of the leaf substitution algebra is an epi.*

Proof. By Lemma 25 it also follows that the closure $\mathrm{cl}(D)$ of any set $D \subseteq \mathbb{D}_\tau$ coincides with the entire domain $\mathbb{D}_\tau$ or it is empty. By the characterization provided in Lemma 12, it follows that every update of the leaf substitution algebra is an epi. ◀

▶ **Lemma 31 (compactness).** *Assumption 1 holds for the free term algebra, both in the variant that includes copyful non-erasing updates and in the variant that restricts to copyless non-erasing updates.*

Proof. Let $\mathbb{D}$ be the free term algebra and let Σ be the ranked alphabet over which the ground terms of $\mathbb{D}$ are defined. We fix a numeric data type τ, with the objective of proving that every constraint of type τ is equivalent to a finite subset of it. The plan is to embed the domain $\mathbb{D}_\tau$ into the polynomial register algebra, and then exploit a finite basis property for the systems of polynomial equations that correspond to the constraints of type τ. The embedding is actually formalized as the composition of two standard encodings: one from the free term algebra to the string register algebra, which maps τ-tuples of ground terms to τ-tuples of XML-like strings, and another one from the string register algebra to the polynomial register algebra, which maps τ-tuples of strings to 2τ-tuples of numbers.

To formalize the first encoding, we introduce two unranked copies of Σ, containing open and closed tags, respectively:

$$\underline{\Sigma} \;=\; \{\underline{a} : a \in \Sigma\} \qquad\qquad \overline{\Sigma} \;=\; \{\overline{a} : a \in \Sigma\}.$$

We then define the encoding enc from terms over Σ to strings over $\underline{\Sigma} \uplus \overline{\Sigma}$, inductively as follows:

$$\mathrm{enc}(a(t_1, \ldots, t_n)) \;=\; \underline{a}\,\mathrm{enc}(t_1)\,\ldots\,\mathrm{enc}(t_n)\,\overline{a}.$$

We naturally extend this encoding in a pointwise manner, so that it maps τ-tuples of ground terms to τ-tuples of strings. It is also convenient to further extend the encoding to terms over the variables $\bar{x} = (x_1, \ldots, x_\tau)$, by simply expanding the alphabet $\underline{\Sigma} \uplus \overline{\Sigma}$ with those variables and by letting $\mathrm{enc}(x_i) = x_i$. For example, the term $a(b, c(x_1))$ is encoded as $\underline{a}\,\underline{b}\,\overline{b}\,\underline{c}\,x_1\,\overline{c}\,\overline{a}$.

Accordingly, every update $f \in \mathbb{D}_{\tau\to\beta}$, specified by a tuple of terms $\bar{c} = (c_1, \ldots, c_\beta)$ over $\bar{x}$ and mapping $\bar{t} = (t_1, \ldots, t_\tau)$ to $f(\bar{t}) = \bar{c}[\bar{x}/\bar{t}]$, is simulated by a corresponding function $\mathrm{enc}(f)$ on tuples of strings, so that $\mathrm{enc}(f)(\mathrm{enc}(\bar{t})) = \mathrm{enc}(f(\bar{t}))$. More precisely, $\mathrm{enc}(f)$ maps every τ-tuple $\bar{w} = (w_1, \ldots, w_\tau)$ of strings to the β-tuple $\bar{u}[\bar{x}/\bar{v}]$, where $\bar{u} = \mathrm{enc}(\bar{c})$ and $\bar{u}[\bar{x}/\bar{w}]$ denotes the substitution in $\bar{u}$ of every occurrences of variable x_i by the i-th component of $\bar{w}$, for all $i = 1, \ldots, \tau$. It is easy to see that the latter function $\mathrm{enc}(f)$ is actually a valid update over the string register algebra.

We now define the second encoding enc′ from the string register algebra to the polynomial register algebra. For this we let k be the size of the alphabet $\underline{\Sigma} \uplus \overline{\Sigma}$ and we think of each letter as a digit from 0 to $k-1$. The function enc′ maps every τ-tuple of strings $(w_1, \ldots, w_\tau)$ to the 2τ-tuple of numbers $(r_1, s_1, \ldots, r_\tau, s_\tau)$, where each r_i is the number whose base-k representation is w_i, and $s_i = k^{|w_i|}$. As for the basic operations on tuples of strings, namely, concatenation, insertion, swap, and duplication of registers, these are simulated by corresponding polynomial maps, as follows:

- concatenation $(\ldots, x_i, x_{i+1}, \ldots) \mapsto (\ldots, x_i x_{i+1}, \ldots)$ is simulated by the polynomial map $(\ldots, y_i, z_i, y_{i+1}, z_{i+1}, \ldots) \mapsto (\ldots, y_i z_{i+1} + y_{i+1}, z_i z_{i+1}, \ldots)$;
- insertion $(\ldots, x_i, x_{i+1}, \ldots) \mapsto (\ldots, x_i, a, x_{i+1}, \ldots)$ is simulated by the polynomial map $(\ldots, y_i, z_i, y_{i+1}, z_{i+1}, \ldots) \mapsto (\ldots, y_i, z_i, r_a, k, y_{i+1}, z_{i+1}, \ldots)$, where r_a is the digit that corresponds to the letter a;

- swap $(\dots, x_i, x_{i+1}, \dots) \mapsto (\dots, x_{i+1}, x_i, \dots)$ is simulated by $(\dots, y_i, z_i, y_{i+1}, z_{i+1}, \dots) \mapsto (\dots, y_{i+1}, z_{i+1}, y_i, z_i, \dots)$;
- duplication $(\dots, x_i, \dots) \mapsto (\dots, x_i, x_i, \dots)$ is simulated by the map$(\dots, y_i, z_i, \dots) \mapsto (\dots, y_i, z_i, y_i, z_i, \dots)$.

The above correspondence is naturally extended via composition to all possible updates over the string register algebra.

By composing the encodings enc and enc′, we obtain an embedding enc;enc′ of the free term algebra into the polynomial register algebra. Via this embedding, we can transform every constraint of type τ over the free term algebra to a system of polynomial equations. By Hilbert's basis theorem [42], we know that every such system has an equivalent finite subsystem, which can then be transformed back to a finite constraint over the free term algebra, showing that all constrained domains are finitary. ◀

▶ **Lemma 33 (effective initial images).** *Given a finite upward STT B, one can compute its initial image* $\mathrm{Reach}(B)$.

Proof. We recall from Propositions 14 and 15 that the transducer $\mathrm{Reach}(B)$ is obtained by restricting the configuration space of $B = (Q, \delta_\triangleright, (\delta_a)_{a\in\Sigma}, \delta_\triangleleft)$ to the closure of the set of reachable configurations. This closure can be equivalently described as the least fixpoint of the monotone operator

$$F(S) \;=\; \mathrm{cl}\big(S \,\cup\, \{\delta_\triangleright(q_\triangleright, d_\triangleright)\} \,\cup\, \textstyle\bigcup_{a\in\Sigma} \delta_a(S)\big)$$

Thus, it suffices to show that, in the copyless, non-erasing, free term algebra, the Kleene iteration $S_{i+1} = F(S_i)$, starting with $S_0 = \varnothing$, can be carried out effectively and stabilizes. We will see that the resulting procedure is similar to the one described in [45] for computing the smallest linear inductive invariant of a linear program.

Let us discuss more in detail the step for the Kleene iteration, which boils down to computing $F(S)$ for a given constrained configuration space $S \subseteq \langle Q\rangle$. Recall that such S is of the form $\bigcup_{q\in Q}\{q\} \times D_q$, where D_q is a constrained domain, which, by Lemma 31, is described by a finite system of equations, say $D_q = \mathrm{Sol}(E_q)$. For the sake of brevity, let

$$D_{\triangleright,q} \;=\; \begin{cases} \{d\} & \text{if } \delta_\triangleright(q_\triangleright, d_\triangleright) = (q, d) \\ \varnothing & \text{otherwise} \end{cases} \qquad \text{and} \qquad D_{p,a,q} \;=\; \begin{cases} \check{\delta}_a(p)(D_p) & \text{if } \hat{\delta}_a(p) = q \\ \varnothing & \text{otherwise.} \end{cases}$$

We get

$$\begin{aligned} F(S) \;&=\; \bigcup_{q\in Q} \{q\} \times \mathrm{cl}\big(D_q \,\cup\, D_{\triangleright,q} \,\cup\, \textstyle\bigcup_{p\in Q, a\in\Sigma} D_{p,a,q}\big) \\ &=\; \bigcup_{q\in Q} \{q\} \times \mathrm{cl}\big(D_q \,\cup\, \mathrm{cl}(D_{\triangleright,q}) \,\cup\, \textstyle\bigcup_{p\in Q, a\in\Sigma} \mathrm{cl}(D_{p,a,q})\big). \qquad \text{(by Lemma 10)} \end{aligned}$$

Therefore, in order to compute $F(S)$ it suffices to know how to construct

1. the closure $\mathrm{cl}(D_1 \cup D_2)$ of the union of two constrained domains D_1, D_2,
2. the closure $\mathrm{cl}(f(D))$ of the application of an update f to a constrained domain D.

In the following we shall focus on these two sub-problems, and finally discuss termination of the fixpoint computation. For both sub-problems we shall exploit Lemma 32, which shows that every constrained domain admits an equivalent parametric form computable as the most general unifier (MGU) of the underlying system of equations.

Closure of unions of constrained domains. We explain how to compute $\mathrm{cl}(D_1 \cup D_2)$ for two constrained domains D_1, D_2 represented by finite systems of equations E_1, E_2, respectively.

Without loss of generality, we assume that both systems E_1 and E_2 are satisfiable: this can be decided thanks to Lemma 32; in case any of the two systems turns out to be unsatisfiable, $\mathrm{cl}(D_1 \cup D_2)$ is defined by the other system. Let $\bar{u}_1, \bar{u}_2$ be the MGUs of E_1, E_2, respectively. By the first part of Lemma 32, these MGUs exist and can be computed from E_1, E_2.

By the definition of closure, $\mathrm{cl}(D_1 \cup D_2)$ is determined by the (generally infinite) set of equations that are valid on $D_1 \cup D_2$, namely, those equations that are entailed by *both* systems E_1 and E_2. Using the second part of Lemma 32, entailment of an equation $\bar{t} = \bar{t}'$ by E_i (for $i = 1, 2$) can be decided by checking syntactic equality of the tuples of terms $\bar{t}[\bar{x}/\bar{u}]$ and $\bar{t}'[\bar{x}/\bar{u}]$, where $\bar{u}$ is the MGU of E_i. When this entailment holds we say that the tuples $\bar{t}, \bar{t}'$ are *unified* by $\bar{u}$. The crux here is to avoid enumerating the infinitely many possible pairs of tuples $\bar{t}, \bar{t}'$ that are unified by $\bar{u}$. To avoid this, we can restrict our attention to tuples that are *maximally deconstructed*, namely, tuples $\bar{t} = (t_1, \ldots, t_\beta)$ and $\bar{t}' = (t'_1, \ldots, t'_\beta)$ of terms over $\bar{x}$ such that, for every $1 \le j \le \beta$, either t_j or t'_j is a variable from $\bar{x}$. Indeed, all pairs unified by $\bar{u}$ consist of tuples of the form $\bar{w}[\bar{y}/\bar{t}]$ and $\bar{w}[\bar{y}]\bar{t}'$, for some arbitrary tuple $\bar{w}$ over $\bar{y}$ and some pair $\bar{t}, \bar{t}'$ of maximally deconstructed tuples unified by $\bar{u}$. We illustrate this principle with an example:

▶ **Example 36.** Consider again the MGU $\bar{u} = (x_1, b(x_1), c(x_1))$ over a single parameter x_1. The following are all the possible pairs of tuples of terms over $\bar{x} = (x_1, x_2, x_3)$ that are unified by $\bar{u}$:

- $\bar{t} = (x_1, x_2, x_3)$ and $\bar{t}' = (x_1, x_2, x_3)$ (trivial pair),
- $\bar{t} = (x_2, b(x_1), x_3)$ and $\bar{t}' = (b(x_1), x_2, x_3)$,
- $\bar{t} = (x_3, c(x_1), x_2)$ and $\bar{t}' = (c(x_1), x_3, x_2)$,
- $\bar{w}[\bar{x}/\bar{t}]$ and $\bar{w}[\bar{x}/\bar{t}']$, where $\bar{w}$ is any tuple of terms over $\bar{x}$ and $\bar{t}, \bar{t}'$ is a pair chosen from any of the previous cases.

Up to permutations of components (which are covered by the fourth case), the first three cases are the only pairs of maximally deconstructed tuples unified by $\bar{u}$.

We let the reader verify that the components of two maximally deconstructed tuples $\bar{t}, \bar{t}'$ unified by $\bar{u}$ are either variables from $\bar{x}$ or terms from $\bar{u}$, and so there are only finitely many such pairs $\bar{t}, \bar{t}'$. Moreover, if $\bar{u}_1, \bar{u}_2$ are the MGUs of two systems of equations E_1, E_2, then the closure of $\mathrm{Sol}(E_1) \cup \mathrm{Sol}(E_2)$ is represented by the (finite) system E of equations $\bar{t} = \bar{t}'$, for all pairs $\bar{t}, \bar{t}'$ that are maximally deconstructed and unified by both $\bar{u}_1$ and $\bar{u}_2$. In particular, one can compute from two systems of equations E_1 and E_2, representing the constrained domains $D_1 = \mathrm{Sol}(E_1)$ and $D_2 = \mathrm{Sol}(E_2)$, a new system of equations E representing the constrained domain $D = \mathrm{cl}(D_1 \cup D_2)$.

Closure of application of an update to a constrained domain. The construction of $\mathrm{cl}(f(D))$ for a given update f and a given constrained domain D follows ideas similar to the previous case. More precisely, let $f \in \mathbb{D}_{\alpha \to \beta}$ be a copyless, non-erasing update specified by a β-tuple of terms $\bar{c}$ over an α-tuple of variables $\bar{x}$ (so that $f : \bar{t} \mapsto \bar{c}[\bar{x}/\bar{t}]$), and let E be a finite system of equations over $\bar{x}$ such that $\mathrm{Sol}(E) = D$ Without loss of generality, assume $\mathrm{Sol}(E) \neq \varnothing$. By the first part of Lemma 32, we can construct the MGU $\bar{u}$ of E, and we can then define the tuple $\bar{v} = \bar{c}[\bar{x}/\bar{u}]$. By construction, $\bar{v}$ is the parametric form of the elements in $f(D)$.

Now, let $\bar{y}$ be a tuple of fresh variables of the same arity as $\bar{c}$. Recall that $\mathrm{cl}(f(D))$ is the solution set of the (infinite) system E' consisting of all equations $\bar{t} = \bar{t}'$ over $\bar{y}$ that are valid on $f(D)$. The tuples of terms $\bar{t}, \bar{t}'$ that are equated by E' can be equivalently described as those for which the original system E entails the equation $\bar{t}[\bar{y}/\bar{c}] = \bar{t}'[\bar{y}/\bar{c}]$. In its turn, thanks to the second part of Lemma 32, the latter entailment is characterized by a syntactic

equality between the tuples

$$\underbrace{\bar{t}[\bar{y}/\bar{c}][\bar{x}/\bar{u}]}_{\bar{t}[\bar{y}/\bar{v}]} \quad \text{and} \quad \underbrace{\bar{t}'[\bar{y}/\bar{c}][\bar{x}/\bar{u}]}_{\bar{t}'[\bar{y}/\bar{v}]}$$

namely, $\bar{t}$ and $\bar{t}'$ are unified by $\bar{v} = \bar{c}[\bar{x}/\bar{u}]$. Finally, by the same arguments used in the previous case (closure of union of constrained domains), we can restrict our attention to pairs of tuples $\bar{t}, \bar{t}'$ that are maximally deconstructed. Towards a conclusion, we observe that there are only finitely many equations $\bar{t} = \bar{t}'$ where $\bar{t}, \bar{t}'$ are maximally deconstructed and unified by $\bar{v}$, and hence one can compute a finite system E' of such equations such that $\mathrm{Sol}(E') = \mathrm{cl}(f(\mathrm{Sol}(E))) = \mathrm{cl}(f(D))$.

Termination of the fixpoint procedure. It remains to argue that the Kleene iteration $S_{i+1} = F(S_i)$ stabilizes after finitely many steps. We start by observing that the system of equations $E_{i,q}$ associated with each step i and each state $q \in Q$, and representing the current constrained configuration space S_i, is weakened at every step — formally, we have that each system $E_{i,q}$ entails the next system $E_{i+1,q}$. This implies that the corresponding MGU $\bar{u}_{i,q}$ becomes more and more general — formally, each MGU $\bar{u}_{i,q}$ is an instantiation of the next MGU $\bar{u}_{i+1,q}$, meaning that the former can be obtained from the latter by a variable substitution. Because the instantiation order is well-founded, it is not possible to have, at a given control state, an infinite chain of MGUs of strictly increasing generality. Finally, because there are only finitely many control states, it follows that S_i is eventually constant, allowing the fixpoint procedure to terminate. ◀

▶ **Lemma 34 (GCDs).** *Assumption 2 holds over the free term algebra with copyless and non-erasing updates. Moreover, GCDs can be computed for finite sets of updates.*

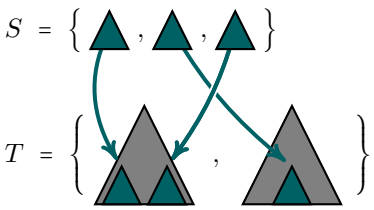
Proof. Let $\mathbb{D}$ denote the free term algebra. Recall that the updates over $\mathbb{D}$ are represented by tuples of terms with variables at the leaves, and that for copyless, non-erasing updates every variable occurs exactly once. Also recall that if two tuples of terms are permutations one of another, then they represent isomorphic updates, which are in particular equivalent w.r.t. divisor relation. For this reason, in this proof it is convenient to further abstract our representations of updates using *multisets of terms*, rather than tuples. More precisely, up to isomorphism, every update $f \in \mathbb{D}_{\alpha \to \beta}$ is represented by a multiset S of cardinality β that contains terms over a number α of variables. We treat the elements of a multiset, and in particular the possible duplicates of the same term, as distinct objects, each associated with a term. A *copy* of a term t inside another term t' (resp. inside a multiset S) is an occurrence of t as a subterm of t' (resp. as a subterm of some element of S). Different occurrences of the same term t are again treated as different objects. We claim without a proof that the divisor relation is characterized as follows:

▶ **Claim 37.** *The update represented by a multiset S is a divisor of the update represented by another multiset T iff there is a function e that maps every element $s \in S$ to a copy of s inside T, in such a way that*

1. *the images of e, seen as occurrences of subterms inside T, are pairwise* disjoint*, meaning they are not contained one in another as subterms;*
2. *the images of e cover all the variables that appear in T.*

A function e as above is called a *subterm embedding* of S into T, and it is denoted for short

by $e : S \hookrightarrow T$. Below is an example:

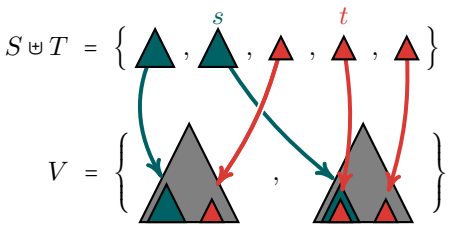


When we write $S, T, \ldots \hookrightarrow U, V, \ldots$ we mean that there are embeddings of every multiset of the left hand-side into every multiset of the right hand-side. We prove the following interpolation property.

▶ **Claim 38.** *For all multisets S, T, there is a multiset U such that $S, T \hookrightarrow U$ and for all multisets V,*

$$S, T \hookrightarrow V \qquad \textit{implies} \qquad S, T \hookrightarrow U \hookrightarrow V.$$

Proof. It is tempting to define U as the disjoint union $S + T$ of S and T, since $S, T \hookrightarrow U$ would hold trivially. However, we cannot claim that if there are embeddings $e_S : S \hookrightarrow V$ and $e_T : T \hookrightarrow V$, then the union $e_S + e_T$ of these embeddings is an embedding $S + T$ into V, essentially because $e_S + e_T$ may violate the requirement of images to be disjoint. Below, we see an example of a violation of this requirement between an element s that originally belonged to S and another element t that originally belonged to T (we use colors to distinguish the *origin*, i.e. S or T, of each element of $S + T$ and the embeddings into V):



To correctly define the interpolant U we need to carefully select the elements from $S + T$. This can be done inductively using a greedy strategy, as follows. We start by listing the elements of $S + T$ based on a total order $<$ that refines the subterm partial order, starting from the largest element, that is: $u_1 > u_2 > \cdots > u_n$. Then, following this fixed order, we add each element u_i to a forest, either as a new root or as a child of a previously added element. In doing so, we shall prefer placing u_i as a child of another element u_j $(j < i)$, provided that the following conditions are satisfied:

- u_j is a root in the current forest,
- u_j contains an occurrence of u_i as a subterm that is disjoint from all current children of u_j (seen as other occurrences of subterms),
- u_j and u_i have opposite origins, namely, either $u_j \in S$ and $u_i \in T$, or, vice versa, $u_j \in T$ and $u_i \in S$.

If more than one element u_j satisfies the above conditions, then we choose any of them, say the first one according to the fixed order. Otherwise, if there is no element u_j that satisfies the above conditions, then we add u_i as a new root of the forest. This clearly produces a forest of height at most 2, in which the children of ever root have opposite origins than the root and represent disjoint subterm occurrences. Finally, we define U as the multiset consisting of the roots of the forest.

It is easy to see that $S \hookrightarrow U$ (and similarly $T \hookrightarrow U$). Indeed, every element s of S must appear in the forest either as a root, say r_s, or as a child, say c_s; in the former case, we can map s to the root r_s itself, which clearly belongs to U; in the latter case, we can map s to a copy inside the parent of c_s, which is a root of the forest and hence belongs to U.

Next, we consider a multiset V admitting embeddings $e_S : S \hookrightarrow V$ and $e_T : T \hookrightarrow V$, and we aim at constructing an embedding $e : U \hookrightarrow V$. We shall define the image $e(u_i)$ of each element u_i of U inductively, following again the total order on $S + T$ that we fixed earlier (recall that U is contained in $S + T$). In doing so, we shall guarantee the following invariant:

> *$e(u_i)$ is a copy of u_i inside $(e_S + e_T)(u_j)$ for some $u_j \in U$ with $j \leq i$ (possibly $j = i$ and $e(u_i) = (e_S + e_T)(u_i)$).*

Suppose, without loss of generality, that u_i has origin in S. We distinguish two cases based on the nesting relationships between the previously defined images $e(u_j)$, for all $u_j \in U$ with $j < i$, and the image $e_S(u_i)$ induced by the original embedding of S into U:

1. $e_S(u_i)$ is disjoint from all previously defined images $e(u_j)$, for all $u_j \in U$ with $j < i$. In this case, we simply let $e(u_i) = e_S(u_i)$.
2. $e_S(u_i)$ is nested inside a previously defined image $e(u_j)$, for some $u_j \in U$ with $j < i$. Without loss of generality, we can assume that u_j is the first element of U whose image $e(u_j)$ contains $e_S(u_i)$. In particular, by the inductive invariant, this means that $e(u_j) = (e_S + e_T)(u_j)$. First, we claim that u_j must have different origin than u_i, namely, $u_j \in T$: indeed, if this were not the case, then e_S would map the two elements u_i and u_j to non-disjoint copies inside U, thus contradicting the definition of embedding. Second, because u_i was added to the forest as a new root, and not as a child of u_j, this means that u_j already contained subterms $u_{k_1}, \ldots, u_{k_m}$, for $m \geq 1$ and $j < k_1, \ldots, k_m < i$, that cover all occurrences of u_i as a subterm of u_j. Of course, the elements $u_{k_1}, \ldots, u_{k_m}$, being children of u_j, do not belong to U, and have different origin than u_j, so they all belong to $S - U$ (i.e. the multiset difference between S and U). Also note that these terms $u_{k_1}, \ldots, u_{k_m}$, being elements of S, must be mapped via e_S to copies inside U. A simple Pigeonhole argument then shows that that at least one such element u_{k_ℓ} must be mapped via e_S to a copy inside U, but *outside* u_j. Accordingly, we define $e(u_i)$ as any occurrence of u_i as a subterm of $e_S(u_{k_\ell})$, which is disjoint from $e(u_j)$.

Based on the above arguments and constructions, the mapping e is a valid embedding of U into V, and thus $S, T \hookrightarrow U \hookrightarrow V$. ◀

It remains to prove, using the above claims, the existence of GCDs for sets of updates, and their computability when the sets are finite. Let H be any set of updates with the same domain, and let F be the set of all common divisors of H. Further let G be the set of *maximal* (not necessarily greatest) elements of F w.r.t. the divisor preorder, namely: G contains an update g iff $g \in F$ and for all $f \in F$, if g is a divisor of f, then f is a divisor of g, meaning that f and g are equivalent w.r.t. the divisor preorder. Of course F, and thus G as well, is not empty, since the identity update is clearly a common divisor of H.

Now, we head towards proving that every update in G is a greatest common divisor of H (this also means that the updates in G are all equivalent w.r.t. the divisor preorder). Let $g \in G$ and $f \in F$. Since both f and g are common divisors of H, by Claims 37 and 38, there is a common divisor g' of H which is divided by both f and g. However, because g was chosen to be maximal in F, we also know that g' is a divisor of g, and hence, by transitivity, f is a divisor of g. This shows that g is a greatest common divisor of H.

Finally, as concerns the computability of such a GCD of H, when H is finite, it suffices to proceed by a fixpoint computation: starting from the identity update, which is clearly

a common divisor of H, one repeatedly extend the update (or equally, one of terms in the multiset that represents it), while preserving the property of being a common divisor, until no further extension is possible. Of course, each extension step is effective and the process will terminate in a finite number of steps. Moreover, even though at each step there might be multiple possible extensions applicable, any maximal sequence of extension steps will eventually produce the same GCD of H up to isomorphism, precisely because of Claim 38. This gives an effective procedure to compute a GCD from a given finite set H of updates. ◀