Minimization of Symbolic Transducers

Olli Saarikivi[1] and Margus Veanes[2]

[1] Aalto University and Helsinki Institute for Information Technology HIIT
Helsinki, Finland
olli.saarikivi@aalto.fi

[2] Microsoft Research
Redmond, USA
margus@microsoft.com

Abstract. Symbolic transducers extend classical finite state transducers to infinite or large alphabets like Unicode, and are a popular tool in areas requiring reasoning over string transformations where traditional techniques do not scale. Here we develop the theory for and an algorithm for computing quotients of such transducers under indistinguishability preserving equivalence relations over states such as bisimulation. We show that the algorithm is a minimization algorithm in the deterministic finite state case. We evaluate the benefits of the proposed algorithm over real-world stream processing computations where symbolic transducers are formed as a result of repeated compositions.

1 Introduction

Finite state automata and transducers are used in a wide variety of applications, ranging from program compilation and verification [5,21] to computational linguistics [32]. A major limitation of classic automata is that their alphabets need to be finite (and small) for the algorithms to scale. To overcome this limitation several approaches have been proposed to accommodate *infinite* alphabets [28,37]. One approach is to use *predicates* instead of concrete characters on state transitions [37,41]. The theory and algorithms of such *symbolic finite automata* (SFAs) and *symbolic finite transducers* (SFTs) modulo *theories*, has recently received considerable attention [39,17] with applications in areas such as parameterized unit testing [38], web security [25], similarity analysis of binaries [16], and code parallelization [40]. Our interest in *symbolic transducers* (STs or SFTs with registers) is motivated by recent applications in string processing and streaming computations [35] where STs are used to express input to output transformations over data streams and UTF8-encoded text files, and STs are *fused* (composed serially) in order to eliminate intermediate streams.

In many applications the need to minimize the number of states without affecting the semantics is crucial for scalability. Much like product composition of classical finite state automata, for STs as well as SFTs, fusion implies a worst case quadratic blowup in the number of (control) states. Thus, similar to automata frameworks such as MONA [27], it is highly beneficial to be able to reduce the

number of states after fusion. Concretely, our initial motivation for minimizing STs came from studying Huffman decoders [26], which we represented as SFTs. When an ST that implements Huffman decoding is fused with some other ST that ignores a part of the decoders output (e.g. everything that is not a digit), then a subgraph of the fused ST's states will resemble an SFA, i.e., have no outputs and no register updates. More generally, fusion might result in a lot of states being *indistinguishable*, i.e., have equivalent behavior for all inputs. This was the key insight that led us to a general algorithm for reducing the number of states of STs, presented here.

In the case of deterministic SFAs it is possible to reduce minimization to classical DFAs at an upfront exponential cost in the size of the SFA [17]. In the case of SFTs, a similar transformation to finite state transducers is not possible because SFTs allow *copying* of input into the output that breaks many of their classic properties [22]. Despite many differences, several algorithms are decidable for SFTs [39]. Whether SFTs can be *minimized* has been an open problem.

Here we develop a general state reduction algorithm that applies to all STs and guarantees minimality in the case of deterministic SFTs. In order to capture minimality we need to extend the definition of an ST [39] to allow *initial* outputs in addition to *final* outputs. The algorithm builds on and generalizes techniques from [33]. First, an ST A is *quasi-determinized* to an equivalent ST $\textsc{qd}(A)$ where all common outputs occur as early as possible. Second, $\textsc{qd}(A)$ is transformed into an SFA $\textsc{sfa}(\textsc{qd}(A))$ over a complex alphabet theory where in addition to the input, each complex character includes a list of output and a pair of current and next register values; A and $\textsc{sfa}(\textsc{qd}(A))$ have the same set of (control) states. Third, $\textsc{sfa}(\textsc{qd}(A))$ is used to compute a bisimulation relation $\sim$ over states through algorithms in [17,18]. Finally, the quotient $\textsc{qd}(A)_{/\sim}$ is formed to collapse bisimilar states. A series of theorems are stated to establish correctness of this algorithm. In particular, Mohri's minimization theorem [33, Theorem 2] is first generalized and used to show that, for deterministic SFTs, the algorithm produces a minimal SFT, where minimality is defined with respect to number of states. We show that, for STs in general, $\textsc{sfa}(A)$, accepts an *over-approximation* of all the valid transductions of the ST A, but the quotient $A_{/\sim}$ preserves the precise semantics of A for any equivalence relation $\sim$ over states that respects state indistinguishability in $\textsc{sfa}(A)$. We further generalize the algorithm to use register invariants. We evaluate the resulting algorithm on a set of STs produced by composing pipelines consisting of real-world stream processing computations. The results show that the state reduction algorithm is effective at reducing the size of symbolic transducers when applied after composition.

To summarize, our contributions are as follows:

- We extend the minimization theorem [33, Theorem 2] to a larger class of sequential functions (Theorem 1). We generalize the quasi-determinization algorithm from [33] to the symbolic setting. (§ 3)
- We develop a theory of state reductions of STs through an over-approximating encoding to SFAs (§ 4 and § 5).

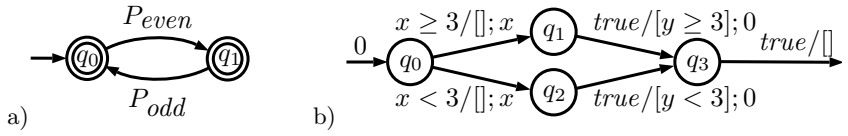

Fig. 1. a) symbolic finite automaton; b) symbolic transducer.

- We describe how to strengthen STs using known invariants to enable register dependent state reductions (§ 6).
- We provide a construction of STs that implement decoders and encoders for prefix codes, e.g., Huffman codes (§ 7.1).
- We show the effectiveness of our state reduction approach on a varied set of STs obtained as compositions of stream processing pipelines (§ 8).

2 Symbolic Automata and Transducers

Here we recall the definitions of a symbolic finite automaton (SFA) and a symbolic transducer (ST) [39]. Before giving the formal definitions below, the underlying intuition behind symbolic automata and transducers is the following. An SFA is like a classical automaton whose concrete characters have been replaced by character predicates. The character predicates are symbolic representations of sets of characters. Such predicates may even denote infinite sets, e.g., when the character domain is the set of integers. The minimal requirements about such predicates are that they are closed under Boolean operations and that checking their satisfiability is decidable.

Example 1. Consider characters that are integers and character predicates as quantifier free formulas over integer linear arithmetic (with modulo-constant operator) containing one free variable x. Define P_{even} as the predicate $x \bmod 2 = 0$, and similarly, define P_{odd} as the predicate $x \bmod 2 = 1$. In this setting the predicate $P_{even} \wedge P_{odd}$ is unsatisfiable and the predicate $P_{even} \vee P_{odd}$ denotes the whole universe. The SFA in Figure 1(a) accepts all sequences of numbers such that every element in an odd position is even and every element in an even position is odd. The first position of a sequence is 1 (thus odd) by definition. ⊠

An ST has, in addition to an input predicate, also an output component, and it may potentially also use a *register*. An ST has finitely many *control states* similar to an SFA, but the register type may be infinite. Therefore an ST may have an infinite state space, where a state is defined as a pair of a control state and a register value. The outputs of an ST are represented symbolically using terms that denote functions from an input and a register to an output.

Example 2. Consider the symbolic transducer in Figure 1(b). A label $\varphi/o; u$ reads as follows: φ is a predicate over (x, y) where x is the input and y the register; o is a sequence of terms denoting functions from (x, y) to output values; u is

a function from (x, y) denoting a register update. For example, $true/[y \geq 3]; 0$ means that the output sequence is the singleton sequence containing the truth value of the current register y being greater than or equal to 3, and the register is reset to 0. A label ρ/o is the label of a *finalizer* leading to an implicit final state, with o being the output upon reaching the end of the input if ρ is true for the register. ⊠

In the following we present formal definitions of SFAs and STs. The notations for SFAs are consistent with [17].

A sequence or list of n elements is denoted by $[e_1, \ldots, e_n]$ or $[e_i]_{i=1}^n$. The empty sequence is [] or ϵ. Concatenation of two sequences u and v is denoted by juxtaposition uv and if e is an element and w a sequence then ew (resp. we) denotes the sequence $[e]w$ (resp. $w[e]$) provided that the types of e and w are clear from the context. Let u and v be sequences and $w = uv$. Then $u^{-1}w \stackrel{\text{def}}{=} v$ denotes the left division of w by u. The relation $u \preccurlyeq v$ means that u is a prefix of v. The operation $u \wedge v$ denotes the maximal common prefix of u and v.

We do not distinguish between a type and the universe that the type denotes, thus treating a type also as a semantic object or set. Given types σ and τ, we write $\mathcal{F}(\sigma{\to}\tau)$ for a given recursively enumerable (r.e.) set of terms f denoting functions $[\![f]\!]$ of type $\sigma \to \tau$. Let the Boolean type be $\mathbb{B}$. Terms in $\mathcal{P}(\sigma) \stackrel{\text{def}}{=} \mathcal{F}(\sigma{\to}\mathbb{B})$ are called (σ-)*predicates*. The type $\sigma{\times}\tau$ is the Cartesian product type of σ and τ.

Example 3. Suppose we use a fixed variable $\mathtt{x}$ for the function argument ($\mathtt{x}$ is possibly a compound argument or a tuple of variables), then an expression such as $\mathtt{x}_1 + \mathtt{x}_2 \in \mathcal{F}(\mathbb{Z}{\times}\mathbb{Z}{\to}\mathbb{Z})$ represents addition, where $\mathtt{x}$ has type $\mathbb{Z}{\times}\mathbb{Z}$ and $\mathtt{x}_i$ represents the i'th element of $\mathtt{x}$ for $i \in \{1, 2\}$. E.g. $\mathtt{x}_1 > \mathtt{0} \wedge \mathtt{x}_2 > \mathtt{0} \in \mathcal{P}(\mathbb{Z}{\times}\mathbb{Z})$ restricts both elements of $\mathtt{x}$ to be positive. ⊠

If S is a set then S^* denotes the Kleene closure of S, i.e., the set of all finite sequences of elements of S. The definitions of symbolic automata and transducers make use of effective Boolean algebras in place of concrete alphabets. An *effective Boolean algebra* $\mathcal{A}$ is a tuple $(U, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$ where U is a non-empty recursively enumerable set of elements called the *universe* of $\mathcal{A}$. Ψ is an r.e. set of *predicates* that is closed under the Boolean connectives, $\vee, \wedge : \Psi \times \Psi \to \Psi$, $\neg : \Psi \to \Psi$, and $\bot, \top \in \Psi$. The *denotation function* $[\![\_]\!] : \Psi \to 2^U$ is r.e. and is such that, $[\![\bot]\!] = \emptyset$, $[\![\top]\!] = U$, for all $\varphi, \psi \in \Psi$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = U \setminus [\![\varphi]\!]$. For $\varphi \in \Psi$, we write $\mathbf{Sat}(\varphi)$ when $[\![\varphi]\!] \neq \emptyset$ and say that φ is *satisfiable*. The algebra $\mathcal{A}$ is *decidable* if $\mathbf{Sat}$ is decidable. We say that $\mathcal{A}$ is *infinite* if U is infinite. In practice, an effective Boolean algebra is implemented with an API having methods that correspond to the Boolean operations.

2.1 Symbolic Finite Automata

Here we recall the definition of a symbolic finite automaton (SFA). The notations are consistent with [17]. We first define a *Σ-automaton* over a (possibly infinite) *alphabet* Σ, a (possibly infinite) *set of states* Q as a tuple $M = (\Sigma, Q, Q^0, F, \Delta)$

where $Q^0 \subseteq Q$ is the set of *initial states*, $F \subseteq Q$ is the set of *final states*, and $\Delta : Q \times \Sigma \times Q$ is the state *transition* relation. A single transition (p, a, q) in Δ is denoted by $p \xrightarrow{a} q$. The transition relation is lifted to $Q \times \Sigma^* \times Q$ as usual: for all $p, q, r \in Q$, $a \in \Sigma$ and $u \in \Sigma^*$: $q \xrightarrow{\epsilon} q$; if $p \xrightarrow{a} q$ and $q \xrightarrow{u} r$ then $p \xrightarrow{au} r$. The *language of* M *at* p is $\mathscr{L}(M, p) = \{w \in \Sigma^* \mid \exists q \in F : p \xrightarrow{w} q\}$. The *language of* M is $\mathscr{L}(M) = \bigcup_{q \in Q^0} \mathscr{L}(M, q)$. M is *deterministic* if $|Q^0| = 1$ and whenever $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ then $q = r$. M is *finite state* or *FA* if Q is finite. *Σ-DFA* stands for *deterministic finite (state) automaton* with alphabet Σ.

Definition 1. $p, q \in Q$ are *indistinguishable*, $p \equiv_M q$, if $\mathscr{L}(M, p) = \mathscr{L}(M, q)$.

If $\equiv$ is an equivalence relation over Q then for $q \in Q$, $q_{/\equiv}$ denotes the $\equiv$-equivalence class containing q and for $X \subseteq Q$, $X_{/\equiv}$ denotes the set of all $q_{/\equiv}$ for $q \in X$. Clearly, $\equiv_M$ is an equivalence relation. The *$\equiv$-quotient* of M is $M_{/\equiv} \stackrel{\text{def}}{=} (\Sigma, Q_{/\equiv}, q^0_{/\equiv}, F_{/\equiv}, \{q_{/\equiv} \xrightarrow{a} p_{/\equiv} \mid q \xrightarrow[M]{a} p\})$. $M_{/\equiv_M}$ is *canonical* and *minimal* among all Σ-DFAs that accept the same language as M [17].

Definition 2. A *symbolic finite automaton* (*SFA*) M is a tuple $(\mathcal{A}, Q, Q^0, F, \Delta)$, where $\mathcal{A}$ is an effective Boolean algebra called the *alphabet*, Q is a finite set of *states*, $Q^0 \subseteq Q$ is the set of *initial states*, $F \subseteq Q$ is the set of *final states*, and Δ is a finite subset of $Q \times \Psi_{\mathcal{A}} \times Q$ called the *transition* relation.

Definition 3. Let $M = (\mathcal{A}, Q, Q^0, F, \Delta)$ be an SFA and $\Sigma = U_{\mathcal{A}}$. The *underlying Σ-FA* of M is $[\![M]\!] \stackrel{\text{def}}{=} (\Sigma, Q, Q^0, F, \{(p, a, q) \mid (p, \varphi, q) \in \Delta, a \in [\![\varphi]\!]\})$.

2.2 Transducers and Sequential Functions

Sequential functions are defined in [33] as functions that can be represented by deterministic finite state transducers that, in order to be algorithmically effective, operate over finite state spaces and finite input and output alphabets. Here we lift the definitions from [33] to the infinite and nondeterministic case. Fortunately, the key results that we need from [33] do not depend on finiteness of alphabets.[3]

Definition 4. A *transducer* is a tuple $\mathbf{f} = (Q, Q^0, F, I, O, \iota, \Delta, \$)$ where Q is a nonempty set of *states*, $Q^0 \subseteq Q$ is the set of *initial states*, $F \subseteq Q$ is the set of *final states*; I and O are nonempty sets called *input alphabet* and output alphabet, $\iota \subseteq Q^0 \times O^*$ is the *initial output relation* or the *initializer*, $\Delta \subseteq Q \times I \times O^* \times Q$ is the *transition relation*, and $\$ \subseteq F \times O^*$ is the *final output relation* or the *finalizer*. $\mathbf{f}$ is *deterministic* if $|Q^0| = 1$, $\iota : Q^0 \to O^*$ and $\$: F \to O^*$ are functions, and $\Delta : Q \times I \to O^* \times Q$ is a partial function.

In the following let $\mathbf{f} = (Q, Q^0, F, I, O, \iota, \Delta, \$)$ be a fixed transducer. The following notations are used: $p \xrightarrow{a/u} q$ stands for $(p, a, u, q) \in \Delta$, and $p \xrightarrow{/u}$ stands for $(p, u) \in \$$, and $\xrightarrow{/u} p$ stands for $(p, u) \in \iota$. The transition relation is lifted to

[3] A technical difference is that Mohri [33] defines the state and the output components of a transition relation separately.

$Q \times I^* \times O^* \times Q$ as follows. For all $p, q, r \in Q, a \in I, v \in I^*, u, w \in O^*$: $p \xrightarrow{\epsilon/\epsilon}\!\!\!\!\rightarrow p$, if $p \xrightarrow{a/u} q$ and $q \xrightarrow{v/w}\!\!\!\!\rightarrow r$ then $p \xrightarrow{av/uw}\!\!\!\!\rightarrow r$. Further, for complete transductions the transition relation is lifted to $Q \times I^* \times O^*$. For all $p, q, \in Q, v \in I^*, u, w \in O^*$, if $p \xrightarrow{v/u}\!\!\!\!\rightarrow q$ and $q \xrightarrow{/w} $ then $p \xrightarrow{v/uw}\!\!\!\!\rightarrow$. The *transduction of* $\mathbf{f}$ *from state* p is the relation $\mathscr{T}(\mathbf{f}, p) \subseteq I^* \times O^*$ such that

$$\mathscr{T}(\mathbf{f}, p) \stackrel{\text{def}}{=} \{(v, w) \mid p \xrightarrow{v/w}\!\!\!\!\rightarrow\}$$

The *transduction of* $\mathbf{f}$ is the relation $\mathscr{T}(\mathbf{f}) \subseteq I^* \times O^*$ such that

$$\mathscr{T}(\mathbf{f}) \stackrel{\text{def}}{=} \{(v, w_0 w) \mid \exists p \in Q^0 \text{ such that } \xrightarrow{/w_0} p \xrightarrow{v/w}\!\!\!\!\rightarrow\}$$

Two transducers are *equivalent* if their transductions are equal. The *domain of* $\mathbf{f}$ *at state* p is the set $\mathscr{D}(\mathbf{f}, p) \stackrel{\text{def}}{=} \{v \in I^* \mid \exists w \in O^* : (v, w) \in \mathscr{T}(\mathbf{f}, p)\}$. The *domain of* $\mathbf{f}$ is $\mathscr{D}(\mathbf{f}) \stackrel{\text{def}}{=} \{v \in I^* \mid \exists w \in O^* : (v, w) \in \mathscr{T}(\mathbf{f})\}$. For any state $q \in Q$ define $P_{\mathbf{f},q}$, or P_q when $\mathbf{f}$ is clear, as the *longest common prefix* of all outputs from q in $\mathbf{f}$:

$$P_{\mathbf{f},q} \stackrel{\text{def}}{=} \bigwedge\{w \mid \exists v : (v, w) \in \mathscr{T}(\mathbf{f}, q)\} \quad \text{where } \bigwedge \emptyset \stackrel{\text{def}}{=} \epsilon.$$

Transform the initializer, the finalizer and the transition relation by promoting the common output prefixes to occur as early as possible as follows:

$$\begin{aligned}
\hat{\iota} &\stackrel{\text{def}}{=} \{(q, wP_q) \mid (q, w) \in \iota\} \\
\hat{\Delta} &\stackrel{\text{def}}{=} \{(p, a, P_p^{-1} w P_q, q) \mid (p, a, w, q) \in \Delta\} \\
\hat{\$} &\stackrel{\text{def}}{=} \{(q, P_q^{-1} w) \mid (q, w) \in \$\}
\end{aligned}$$

The corresponding transformation of $\mathbf{f}$ is defined as follows.

Definition 5. *Quasi-determinization* of $\mathbf{f}$ is $\mathbf{qd}(\mathbf{f}) \stackrel{\text{def}}{=} (Q, Q^0, F, I, O, \hat{\iota}, \hat{\Delta}, \hat{\$})$.

Quasi-determinization of $\mathbf{f}$ can be seen as a way to reduce nondeterminism in the output part and the following proposition follows from the definitions.

Proposition 1. $\mathscr{T}(\mathbf{f}) = \mathscr{T}(\mathbf{qd}(\mathbf{f}))$ *and* $\mathbf{qd}(\mathbf{qd}(\mathbf{f})\ ) = \mathbf{qd}(\mathbf{f})$.

When $\mathbf{f}$ is deterministic we write $\mathscr{T}_{\mathbf{f},p} : \mathscr{D}(\mathbf{f}, p) \to O^*$ for $\mathscr{T}(\mathbf{f}, p)$ and $\mathscr{T}_{\mathbf{f}} : \mathscr{D}(\mathbf{f}) \to O^*$ for $\mathscr{T}(\mathbf{f})$ as functions. In particular, $\mathscr{T}_{\mathbf{f}}(v) = w$ means $(v, w) \in \mathscr{T}(\mathbf{f})$ and similarly for $\mathscr{T}(\mathbf{f}, p)$. Moreover, let $\mathbf{f}(v) \stackrel{\text{def}}{=} \mathscr{T}_{\mathbf{f}}(v)$ for $v \in \mathscr{D}(\mathbf{f})$.

Definition 6. A *sequential transducer* is a deterministic transducer with finitely many states. A *sequential function* is the transduction of some sequential transducer. A sequential transducer is *minimal* if there exists no equivalent sequential transducer with fewer states.

The initial output is needed for minimality, while the finalizer increases expressiveness.

Example 4. Consider an HTML decoder that replaces every pattern `<` with `<`; e.g. the string `"<&lt"` is mapped to `"<&lt"`. This is a sequential function whose sequential transducer requires the use of a finalizer, unless I is extended with a new end-of-input symbol that is used to terminate all input sequences. ⊠

Let $\mathbf{f} = (Q, Q^0, F, I, O, \iota, \Delta, \$)$ be a transducer. The underlying automaton of $\mathbf{f}$ combines inputs and outputs into single labels. Let $q^0, q^\bullet \notin Q$ be distinct new states and let Σ be the alphabet:

$$\Sigma = \{c_w \mid \exists q : (q, w) \in \iota \cup \$\} \cup \{c_w^a \mid \exists p, q : (p, a, w, q) \in \Delta\}$$

The *Σ-automaton of* $\mathbf{f}$ is $\mathbf{aut}(\mathbf{f}) \stackrel{\text{def}}{=} (\Sigma, Q \cup \{q^0, q^\bullet\}, \{q^0\}, \{q^\bullet\}, \Delta_0 \cup \Delta_1 \cup \Delta_2\}$, where $\Delta_0 = \{(q^0, c_w, p) \mid (p, w) \in \iota\}$, $\Delta_1 = \{(p, c_w^a, q) \mid (p, a, w, q) \in \Delta\}$, and $\Delta_2 = \{(p, c_w, q^\bullet) \mid (p, w) \in \$\}$.

Minimization of a sequential transducer $\mathbf{f} = (Q, Q^0, F, I, O, \iota, \Delta, \$)$ proceeds now in two steps. First, $\mathbf{f}$ is quasi-determinized to $\mathbf{qd}(\mathbf{f})$. Second, $\mathbf{qd}(\mathbf{f})$ is minimized by collapsing states that are indistinguishable with respect to $\mathbf{aut}(\mathbf{qd}(\mathbf{f}))$. Let $\equiv$ be $\equiv_{\mathbf{aut}(\mathbf{qd}(\mathbf{f}))}$ in:

$$\begin{aligned}
\mathbf{qd}(\mathbf{f})_{/\equiv} &= (Q_{/\equiv}, Q^0_{/\equiv}, F_{/\equiv}, I, O, \hat{\iota}_{/\equiv}, \hat{\Delta}_{/\equiv}, \hat{\$}_{/\equiv}) \\
\hat{\iota}_{/\equiv} &= \{(q_{/\equiv}, w) \mid (q, w) \in \hat{\iota}\} \\
\hat{\$}_{/\equiv} &= \{(q_{/\equiv}, w) \mid (q, w) \in \hat{\$}\} \\
\hat{\Delta}_{/\equiv} &= \{(p_{/\equiv}, a, w, q_{/\equiv}) \mid (p, a, w, q) \in \hat{\Delta}\}
\end{aligned}$$

The following is a generalized form of Mohri's theorem that allows finalizers and infinite alphabets. For our purposes it therefore captures minimality at the semantic level rather than providing a decision procedure for minimization.

Theorem 1 (Mohri). *If* $\mathbf{f}$ *is a sequential transducer then* $\mathbf{qd}(\mathbf{f})_{/\equiv_{\mathbf{aut}(\mathbf{qd}(\mathbf{f}))}}$ *is a minimal sequential transducer that is equivalent to* $\mathbf{f}$.

2.3 Symbolic Transducers

A *symbolic transducer* (ST) represents a streaming computation over finite input sequences, where the input elements belong to some not-necessarily bounded domain. Let $X \subseteq_{\text{fin}} Y$ stand for X is a *finite subset* of Y.

Definition 7. A *symbolic transducer* is a tuple $A = (I, O, Q, q^0, o^0, T, F, R, r^0)$ where I is an *input element type*, O is an *output element type*, R is a *register type*, and Q is a *finite set* of *control states*, and where $q^0 \in Q$ is the *initial control state*, $r^0 \in R$ is the *initial register*, $o^0 \in O^*$ is the *initial output*,

$$\begin{aligned}
T &\subseteq_{\text{fin}} Q \times (\mathcal{P}(I \times R) \times \mathcal{F}(I \times R \to O)^* \times \mathcal{F}(I \times R \to R)) \times Q \\
F &\subseteq_{\text{fin}} Q \times (\mathcal{P}(R) \times \mathcal{F}(R \to O)^*)
\end{aligned}$$

T is the *transition relation*, and F is the *finalizer*.

Let $\mathscr{D}(F)$ denote the set of all $(q, r) \in F \times R$ such that there exists a final rule $(q, \varphi, \bar{v}) \in F$ and $r \in [\![\varphi]\!]$. Given $\bar{v} = [v_i]_{i=1}^n \in \mathcal{F}(\tau_1 \to \tau_2)^*$ we let $[\![\bar{v}]\!]$ denote the function from τ_1 to τ_2^* such that for $a \in \tau_1$, $[\![\bar{v}]\!](a) = [[\![v_i]\!](a)]_{i=1}^n$.

Definition 8. The *underlying transducer of* A, denoted by $[\![A]\!]$, is defined as the transducer $(Q \times R, \{(q^0, r^0)\}, \mathscr{D}(F), I, O, \{(q^0, r^0) \mapsto o^0\}, \Delta, \$)$ where

$$\Delta = \{(p, r) \xrightarrow{a/[\![\bar{v}]\!](a,r)} (q, [\![u]\!](a, r)) \mid (p, (\varphi, \bar{v}, u), q) \in T, (a, r) \in [\![\varphi]\!]\}$$
$$\$ = \{(p, r) \xrightarrow{/[\![\bar{v}]\!](r)} \mid (p, (\varphi, \bar{v})) \in F, r \in [\![\varphi]\!]\}$$

A is *deterministic* if $[\![A]\!]$ is deterministic. Let $A = (I, O, Q, q^0, o^0, T, F, R, r^0)$ be a fixed deterministic ST.

Definition 9. A is a symbolic *finite* transducer or *SFT* if $|R| = 1$. We omit the trivial register type and omit the corresponding components when A is an SFT and let $A = (I, O, Q, q^0, o^0, T, F)$ where $T \subseteq_{\text{fin}} Q \times (\mathcal{P}(I) \times \mathcal{F}(I \to O)^*) \times Q$ and $F \subseteq_{\text{fin}} Q \times O^*$. A deterministic SFT A is *minimal* if $[\![A]\!]$ is minimal.

A deterministic SFT is the symbolic counterpart of a sequential transducer. Observe that in any symbolic transducer with a finite register type we can eliminate the register component by fusing it with the control state component and thus turn the ST into an SFT by using a state exploration algorithm [40].

Example 5. See Figure 2. *Smileyfy* is a deterministic SFT whose input type and output type is Unicode.[4] The purpose of *Smileyfy* is to decode each pattern `:-)` into a smiley symbol[5] and to leave the input unchanged otherwise. For example *Smileyfy*(`"☺:-):-"`) = `"☺☺:-"`. *Unsmileyfy* is an SFT that replaces each smiley with the pattern `:-)` and leaves the input unchanged otherwise. ⊠

3 Quasi-Determinization of Symbolic Transducers

Let $A = (I, O, Q, q^0, o^0, T, F, R, r^0)$ be a fixed ST. Assume that the ST is *clean*, meaning that all predicates that occur in the rules of A are satisfiable. Given a rule r in T or F we can effectively decide if some element of the output has a fixed value that is independent of the input and the register. Such constant value analysis is performed as follows. Consider $(p, (\lambda x.\varphi(x), [\lambda x.v_i(x)]_{i=1}^n, u), q) \in T$. Recall that $x : I \times R$ and $v_i(x) : O$. In order to decide if $\forall x x' : \varphi(x) \wedge \varphi(x') \Rightarrow v_i(x) = v_i(x')$ check unsatisfiability of $\varphi(x) \wedge \varphi(x') \wedge v_i(x) \neq v_i(x')$. If the formula is unsatisfiable we know that this implies that $v_i(x)$ is a fixed value for any x such that $\varphi(x)$ holds because $\varphi(x)$ is satisfiable since the ST is clean. We can then select an arbitrary model $\mathfrak{A} \models \varphi(x)$ and evaluate $v_i(x)$ in that model, say $a_i = v_i(x)^{\mathfrak{A}}$ and replace v_i by a_i in the output of the rule (as a preprocessing step of A).

[4] The Unicode alphabet is finite but very large, over a million characters. For most practical purposes it is considered as the set on natural numbers $\mathbb{N}$.

[5] For example Unicode codepoint $\texttt{263A}_{16}$ or a smiley in the emoticon alphabet [2].

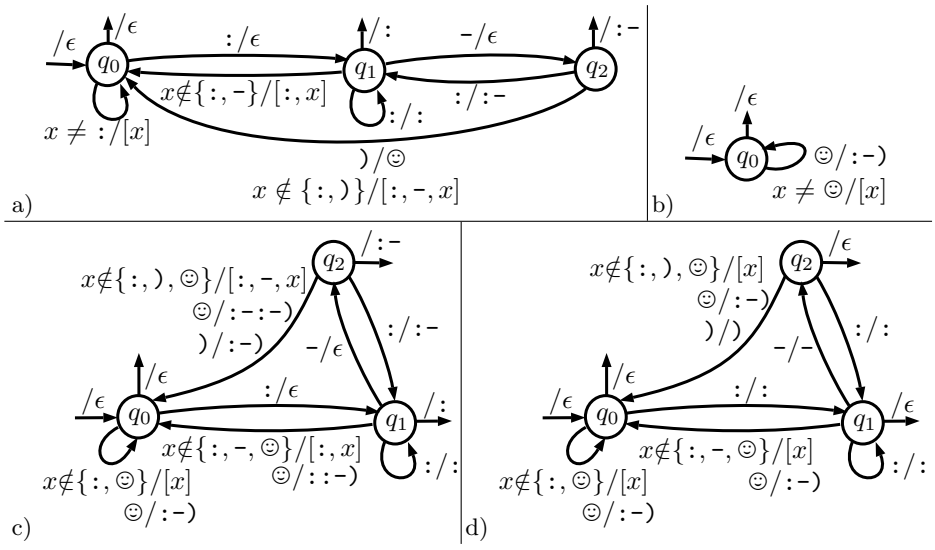

Fig. 2. SFTs: a) *Smileyfy*; b) *Unsmileyfy*; c) $SU = Smileyfy \circ Unsmileyfy$; d) $\textsc{qd}(SU)$.

If, on the other hand, $\varphi(x) \wedge \varphi(x') \wedge v_i(x) \neq v_i(x')$ is satisfiable it means that there exist at least two different outputs (for some different inputs for x and x', respectively). Let $\mathtt{1}$ and $\mathtt{2}$ be two fixed distinct symbols. Create multi-symbol NFA transitions $p \xrightarrow{c_1 \cdots c_n} q$ where $c_i = a_i$ in the first case and $c_i \in \{\mathtt{1}, \mathtt{2}\}$ in the second case. This yields an NFA over the finite alphabet $O \cup \{\mathtt{1}, \mathtt{2}\}$ that can be quasi-determinized [33] to compute the maximal common prefixes $P_{A,p}$, or P_p when A is clear, for all $p \in Q$. Observe that $P_p \in O^*$ because the symbols $\mathtt{1}$ and $\mathtt{2}$ cannot occur in any common prefix since they conflict with each other. Next, the rules of A can be transformed to quasi-determinize A as follows. In each transition from p to q with output $\bar{v}$, replace $\bar{v}$ by $P_p^{-1}\bar{v}P_q$. In every final rule from p with output $\bar{v}$, replace $\bar{v}$ by $P_p^{-1}\bar{v}$. The initial output becomes $\hat{o}^0 = o^0 P_{q^0}$. Let the resulting ST be $\textsc{qd}(A) \stackrel{\text{def}}{=} (I, O, Q, q^0, \hat{o}^0, \hat{T}, \hat{F}, R, r^0)$.

Lemma 1. $\mathscr{T}([\![\textsc{qd}(A)]\!]) = \mathscr{T}(\mathbf{qd}([\![A]\!]))$.

Lemma 2. *If A is an SFT then* $[\![\textsc{qd}(A)]\!] = \mathbf{qd}([\![A]\!])$.

The following example illustrates the effect of quasi-determinization on SFTs. The example gives a simplified but realistic scenario involving composition of string manipulating functions. Chains of string transformations where data has been encoded and is being decoded before further analysis are frequent and may lead to extensive computation overheads [35].

Example 6. Recall Figure 2. Composition of *Smileyfy* with *Unsmileyfy* is an SFT SU that first applies *Smileyfy* and then *Unsmileyfy*. SU is shown in Figure 2(c). If we calculate the maximal output prefixes for all the states in SU we get that

$P_{q_0} = \epsilon$, $P_{q_1} = \texttt{":"}$, and $P_{q_2} = \texttt{":-"}$. After quasi-determinizing SU we get the SFT in Figure 2(d). For example, consider the transition from q_2 to q_1 in SU. Then $P_{q_2}^{-1}\texttt{":-"}P_{q_1} = \texttt{":-"}^{-1}\texttt{":-:"} = \texttt{":"}$. So in $\textsc{qd}(SU)$ we have $q_2 \xrightarrow{:/:} q_1$. ⊠

To enable more quasi-determinization in the presence of registers the QD-algorithm above can be modified to also move outputs that are only independent of the input, but not the register. Instead of checking for a constant value, yields are checked for input-independence: $\forall a, a', r : \varphi((a,r)) \wedge \varphi((a',r)) \Rightarrow v_i((a,r)) = v_i((a',r))$. The modified QD must also find common prefixes under the equivalence of yield formulas. To move register-dependent yields the register update of the transition the yield is moved over must be substituted into the formula. Furthermore, the yield formulas may be equivalent only under the context of their transitions' guards, and therefore a representative for an equivalence class of yields may need to be constructed from the constituent formulas.

4 SFA encoding of symbolic transducers

Here we provide a translation that lifts STs to SFAs. This translation is used to reduce state reduction of STs to minimization of SFAs and plays therefore a central role in the paper. Given an ST $A = (I, O, Q, q^0, o^0, T, F, R, r^0)$ we construct an SFA $\textsc{sfa}(A)$ for A by representing the labels of all the rules of A as predicates in a set $\mathcal{P}(L)$ where L is a type that encodes the labels. Let the effective Boolean algebra be $\mathcal{A}$, whose universe is L and whose set of predicates is $\mathcal{P}(L)$. We write $[\sigma]$ for the type of finite sequences or *lists* of elements of type σ. We access the i'th element of an element x having Cartesian product type (or tuple type) by x_1, x_2, x_3, etc. We define L as the disjoint union type $\texttt{T}((I{\times}R){\times}[O]{\times}R) \cup \texttt{F}(R{\times}[O])$. The intent behind the type L is the following. A concrete label $\texttt{T}((a,r),b,r')$ is an instance of the label of a transition such that for input a and register r the transition produces the output sequence b and the updated register r'. A concrete label $\texttt{F}(r,b)$ is an instance of the label of a finalizer such that for register r the final output sequence is b.

Definition 10. The *predicate encoding of a label* l is the following L-predicate ϕ_l. For $l = (\varphi, [f_i]_{i=1}^n, g) \in \mathcal{P}(I{\times}R) \times \mathcal{F}(I{\times}R{\to}O)^* \times \mathcal{F}(I{\times}R{\to}R)$:

$$\phi_l(x) \stackrel{\text{def}}{=} \texttt{IsT}(x) \wedge \varphi(x_1) \wedge x_2 = [f_i(x_1)]_{i=1}^n \wedge x_3 = g(x_1).$$

For $l = (\varphi, [f_i]_{i=1}^n) \in \mathcal{P}(R) \times \mathcal{F}(R{\to}O)^*$:

$$\phi_l(x) \stackrel{\text{def}}{=} \texttt{IsF}(x) \wedge \varphi(x_1) \wedge x_2 = [f_i(x_1)]_{i=1}^n.$$

An important aspect of ϕ_l is that it is quantifier free and that its satisfiability is *decidable* provided that $\mathcal{A}$ is decidable. Moreover, $\neg\phi_l$ is a quantifier free predicate in $\mathcal{P}(L)$ by virtue of $\mathcal{P}(L)$ being closed under complement.

Definition 11. The *SFA of* A, denoted $\textsc{sfa}(A)$, is the following SFA:

$$\textsc{sfa}(A) \stackrel{\text{def}}{=} (\mathcal{P}(L), Q \cup \{q^\bullet\}, q^0, \{q^\bullet\}, \Delta_{\textsc{sfa}(A)})$$
$$\Delta_{\textsc{sfa}(A)} = \{(p, \phi_l, q) \mid (p, l, q) \in T\} \cup \{(p, \phi_l, q^\bullet) \mid (p, l) \in F\}$$

The following theorem relates the transduction semantics of an ST with the language of the corresponding SFA.

Theorem 2 (Control State Abstraction). *The following are equivalent for all $u = (a_1 \cdots a_n) \in I^*$ and $v \in O^*$:*

1. $(u, o^0 v) \in \mathscr{T}(\llbracket A \rrbracket)$
2. *There exist $r_0 = r_A^0$, $e \in O^*$, and, for $1 \leq i \leq n$, $v_i \in O^*$, $r_i \in R$, such that $[\mathtt{T}((a_i, r_{i-1}), v_i, r_i)]_{i=1}^n [\mathtt{F}(r_n, e)] \in \mathscr{L}(\textsc{sfa}(A))$ and $v = v_1 v_2 \cdots v_n e$.*

Proof. Any L-predicate over $\mathtt{T}$-elements can be written equivalently as

$$\lambda\, \mathtt{T}((x, y), z, w).\gamma(x, y) \wedge z = f(x, y) \wedge w = g(x, y)$$

which maps one-to-one with the ST transition label $\gamma / f; g$. Similarly for $\mathtt{F}$-elements. We now state the following key property between A and $\textsc{sfa}(A)$ that directly relates the trace semantics of A with the language of $\textsc{sfa}(A)$. The proof of (*) follows from the definitions.

(*) For all $p, q \in Q$, $r, s \in R$, $a \in I$ and $v \in O^*$:

$$(p, r) \xrightarrow[\llbracket A \rrbracket]{a/v} (q, s) \;\Leftrightarrow\; p \xrightarrow[\llbracket \textsc{sfa}(A) \rrbracket]{\mathtt{T}((a,r),v,s)} q \qquad \text{and} \qquad (p, r) \xrightarrow[\llbracket A \rrbracket]{\varepsilon/v} \Leftrightarrow\; p \xrightarrow[\llbracket \textsc{sfa}(A) \rrbracket]{\mathtt{F}(r,v)} q^\bullet.$$

Theorem 2 is proved by induction over the length of u and by using (*). ⊠

We refer to Theorem 2 as the *ST control state abstraction theorem* because $\mathscr{L}(\textsc{sfa}(A))$ abstracts the use of the particular control states in any run of A. Note that while Theorem 2 ensures that $\mathscr{L}(\textsc{sfa}(A))$ includes all valid transductions, $\mathscr{L}(\textsc{sfa}(A))$ may also include sequences that do not correspond to valid transductions due to the register not evolving consistently, i.e., sequences containing a subsequence $[\mathtt{T}((a_i, r_i), v_i, r_i), \mathtt{T}((a_{i+1}, r_i'), v_{i+1}, r_{i+1})]$ where $r_i \neq r_i'$. We will see in the next section that it is still safe to use $\textsc{sfa}(A)$ for control state reduction in A.

5 Minimization

We use the following algorithm for reducing the number of control states of an ST A. We first quasi-determinize A and then transform $\textsc{qd}(A)$ into an SFA $\textsc{sfa}(\textsc{qd}(A))$ and use existing algorithms to reduce the number of states of $\textsc{sfa}(\textsc{qd}(A))$. The reduction of $\textsc{sfa}(\textsc{qd}(A))$ provides us with an equivalence relation $\sim$ over Q that can be used to merge $\sim$-equivalent states in A while preserving the transduction semantics of A. If $\sim$ is an equivalence relation over Q then the $\sim$-quotient of A is the ST

$$A_{/\sim} \stackrel{\text{def}}{=} (I, O, Q_{/\sim}, q^0_{/\sim}, o^0, \{(p_{/\sim}, l, q_{/\sim}) \mid (p, l, q) \in T\}, \{(p_{/\sim}, l) \mid (p, l) \in F\}, R, r^0).$$

The following theorem states that we can merge control states in A that are indistinguishable in $\textsc{sfa}(A)$ into one state, without affecting the transduction semantics of A.

Theorem 3. *For all $q \in Q_A$, $r \in R$, $u \in I^*$, $v \in O^*$, and equivalence relations $\sim \subseteq \equiv_{\text{SFA}(A)}$ this holds: $(u,v) \in \mathscr{T}([\![A]\!], (q,r)) \Leftrightarrow (u,v) \in \mathscr{T}([\![A_{/\sim}]\!], (q_{/\sim}, r))$*

Proof. Let $u = [a_i]_{i=1}^n$. Suppose $p \sim q$. We have the following equivalences:

$$
\begin{aligned}
(p,r) \xrightarrow[[\![A]\!]]{u/v}\!\!\!\!\rightarrow \Leftrightarrow\ & \exists p_1 \dots p_n, r_1 \dots r_n, v_1 \dots v_n, e : v = v_1 + v_2 + \dots + v_n + e, \\
& (p,r) \xrightarrow[[\![A]\!]]{a_1/v_1} (p_1,r_1) \xrightarrow[[\![A]\!]]{a_2/v_2} (p_2,r_2) \cdots (p_n,r_n) \xrightarrow[[\![A]\!]]{\varepsilon/e} \\
\Leftrightarrow\ & \exists p_1 \dots p_n, r_1 \dots r_n, v_1 \dots v_n, e : v = v_1 + v_2 + \dots + v_n + e, \\
& p \xrightarrow[[\![\text{SFA}(A)]\!]]{\mathtt{T}((a_1,r),v_1,r_1)} p_1 \xrightarrow[[\![\text{SFA}(A)]\!]]{\mathtt{T}((a_2,r_1),v_2,r_2)} p_2 \cdots p_n \xrightarrow[[\![\text{SFA}(A)]\!]]{\mathtt{F}(r_n,e)} q^\bullet \\
\Leftrightarrow\ & \exists p'_1 \dots p'_n, r_1 \dots r_n, v_1 \dots v_n, e : v = v_1 + v_2 + \dots + v_n + e, \\
& q \xrightarrow[[\![\text{SFA}(A)]\!]]{\mathtt{T}((a_1,r),v_1,r_1)} p'_1 \xrightarrow[[\![\text{SFA}(A)]\!]]{\mathtt{T}((a_2,r_1),v_2,r_2)} p'_2 \cdots p'_n \xrightarrow[[\![\text{SFA}(A)]\!]]{\mathtt{F}(r_n,e)} q^\bullet \\
\Leftrightarrow\ & \exists p'_1 \dots p'_n, r_1 \dots r_n, v_1 \dots v_n, e : v = v_1 + v_2 + \dots + v_n + e, \\
& (q,r) \xrightarrow[[\![A]\!]]{a_1/v_1} (p'_1,r_1) \xrightarrow[[\![A]\!]]{a_2/v_2} (p'_2,r_2) \cdots (p'_n,r_n) \xrightarrow[[\![A]\!]]{\varepsilon/e} \\
\Leftrightarrow\ & (q,r) \xrightarrow[[\![A]\!]]{u/v}\!\!\!\!\rightarrow
\end{aligned}
$$

where the first equivalence holds by definition, the second equivalence uses Theorem 2(*), the third equivalence uses $p \equiv_{\text{SFA}(A)} q$, the fourth equivalence uses Theorem 2(*) again, and the last equivalence holds by definition. Therefore we can replace q by $q_{/\sim}$ without affecting the transduction semantics. ⊠

The key implication for A is that we can replace all indistinguishable control states with a single fixed representative of the indistinguishability equivalence class. The most typical use for minimization arises as a post-processing step after composition. The following example illustrates a simplified scenario. The fusion composition of A and B, denoted $A \circ B$, has the classic semantics of relation composition: $(w,v) \in \mathscr{T}_{A \circ B} \Leftrightarrow \exists z : (w,z) \in \mathscr{T}_A \wedge (z,v) \in \mathscr{T}_B$.

Example 7. If we apply the SFA minimization algorithm from [17] to the SFA $\text{SFA}(\text{QD}(SU))$, with $\text{QD}(SU)$ as in in Figure 2(c), we get an equivalence relation where all the states are indistinguishable. It turns out that the composed SFT in Figure 2(d) is equivalent to the minimal SFT in Figure 2(b). ⊠

We get the following general state reduction theorem for STs by combining the above theorems. In the special case of deterministic SFTs it is a minimization theorem that provides a partial answer to the open problem of whether SFTs can be effectively minimized. For the case of functional (aka. single-valued) but possibly nondeterministic SFTs is still an open problem if an effective minimization procedure exists.

Theorem 4. *Let $A = (I, O, Q, q^0, o^0, T, F, R, r^0)$ be an ST. The following holds.*
a) If $\sim \subseteq \equiv_{\text{SFA}(\text{QD}(A))}$ and $\sim$ is an equivalence relation then $\mathscr{T}(\text{QD}(A)_{/\sim}) = \mathscr{T}(A)$.
b) If A is a deterministic SFT then $\text{QD}(A)_{/\equiv_{\text{SFA}(\text{QD}(A))}}$ is minimal.

Proof. We prove (a) first. Let $\sim \subseteq \equiv_{\mathrm{SFA}(\mathrm{QD}(A))}$ be an equivalence relation, $u \in I^*$, and $w \in O^*$. Recall that, for any ST B, $\mathscr{T}(B) \stackrel{\text{def}}{=} \mathscr{T}([\![B]\!])$. Let $o = o^0 P_{A,q^0}$. We get that

$$\begin{aligned}
(u,w) \in \mathscr{T}([\![\mathrm{QD}(A)_{/\sim}]\!]) &\iff o \preccurlyeq w \text{ and } (u, o^{-1}w) \in \mathscr{T}([\![\mathrm{QD}(A)_{/\sim}]\!], (q^0_{/\sim}, r^0)) \\
&\stackrel{\text{Thm 3}}{\iff} o \preccurlyeq w \text{ and } (u, o^{-1}w) \in \mathscr{T}([\![\mathrm{QD}(A)]\!], (q^0, r^0)) \\
&\iff (u,w) \in \mathscr{T}([\![\mathrm{QD}(A)]\!]) \\
&\stackrel{\text{Lma 1}}{\iff} (u,w) \in \mathscr{T}(\mathbf{qd}([\![A]\!])) \\
&\stackrel{\text{Prop 1}}{\iff} (u,w) \in \mathscr{T}([\![A]\!])
\end{aligned}$$

We prove (b) next. Let $\sim$ be $\equiv_{\mathrm{SFA}(\mathrm{QD}(A))}$. By [17, Theorem 2] and Lemma 2 we have that $[\![\mathrm{QD}(A)]\!] = \mathbf{qd}([\![A]\!])$ and so $\sim = \equiv_{\mathbf{aut}(\mathbf{qd}([\![A]\!]))}$. Theorem 1 implies now that $\mathbf{qd}([\![A]\!])_{/\sim}$ is minimal and $\mathscr{T}(\mathbf{qd}([\![A]\!])_{/\sim}) = \mathscr{T}(A)$ which implies that $\mathrm{QD}(A)_{/\sim}$ is minimal since $[\![\mathrm{QD}(A)]\!]_{/\sim} = [\![\mathrm{QD}(A)_{/\sim}]\!] = \mathbf{qd}([\![A]\!])_{/\sim}$ where we may assume, without loss of generality, that the state space of an SFT A is Q. ⊠

We can apply Theorem 4(a) to deterministic STs by using the minimization algorithms from [17] to compute $\equiv_{\mathrm{SFA}(\mathrm{QD}(A))}$, since determinism is preserved by the SFA transformation. It is also clear that $\mathrm{QD}(\cdot)$ preserves determinism.

Theorem 4(a) also holds for nondeterministic STs. Practical significance of Theorem 4(b) is that most SFTs that are being used in the context of string encoding, string decoding and string sanitization routines [25] are indeed deterministic and composition of SFTs are used frequently for example for composing different encoding routines and minimization is one technique to optimize such generated code.

While Theorem 4 provides a way to minimize the number of states in an SFT, the transitions may still have a non-minimal representation. The techniques and complexity for minimizing guards and output formulas will depend on what the effective Boolean algebra in question is. For example for BDDs choosing the variable order that minimizes the size is NP-complete [11], while general Boolean formula minimization is NP^{P}-complete [13].

6 Register-carried indistinguishability

The SFA encoding presented in Section 4 does not handle indistinguishability arising from register carried dependencies.

Example 8. In the SFA encoding of the ST in Figure 1(b) the states q_1 and q_2 are distinguishable, since the encoding of the transition $q_1 \xrightarrow{true/[x \geq 3];\, 0} q_3$ matches the set of concrete labels $\{\, \mathtt{T}((a,r),[b],0) \mid a \in I, r \in R, b = (r \geq 3) \,\}$, which is distinct from the concrete labels $\{\, \mathtt{T}((a,r),[b],0) \mid a \in I, r \in R, b = (r < 3) \,\}$ matched by the encoding of the transition $q_2 \xrightarrow{true/[x < 3];\, 0} q_3$. However, in the ST the transition $q_0 \xrightarrow{x \geq 3/[];\, x} q_1$ is the only incoming transition for q_1 and thus the register value at q_1 will always be ≥ 3, which implies that the transition from

q_1 to q_3 can only output $[true]$. By a similar argument the same holds for the transition from q_2 to q_3. Therefore the two states are indistinguishable when the state invariants implied by the incoming transitions are taken into account. ⊠

Assuming such invariants are available they can be used to strengthen an ST to make more state reduction available.

Definition 12. Let there be an ST $A = (I, O, Q, q^0, o^0, T, F, R, r^0)$ and a function $\zeta : Q \to \mathcal{P}(R)$ such that for all $p \in Q$, $r \in R$, $v \in I^*$ and $w \in O^*$ it holds that

$$(q^0, r^0) \xrightarrow[[\![A]\!]]{v/w}\!\!\!\!\rightarrow (p, r) \;\Rightarrow\; \zeta(p)(r)$$

Intuitively ζ gives per-control state invariants for all reachable register values. Now a corresponding strengthened ST $\textsc{inv}^\zeta(A)$ can be constructed as:

$$\begin{aligned}
\textsc{inv}^\zeta(A) &\stackrel{\text{def}}{=} (I, O, Q, q^0, o^0, T', F', R, r^0)\\
T' &= \{\, p \xrightarrow{\zeta(p) \wedge \varphi / v;\, u} q \mid (p, \varphi, v, u, q) \in T \,\}\\
F' &= \{\, p \xrightarrow{\zeta(p) \wedge \varphi / v} \mid (p, \varphi, v) \in F \,\}
\end{aligned}$$

Theorem 5. $\mathscr{T}(\textsc{inv}^\zeta(A)) = \mathscr{T}(A)$

Proof. Recall the assumption that for all $p \in Q$, $r \in R$, $v \in I^*$ and $w \in O^*$ it holds that $(q^0, r^0) \xrightarrow[[\![A]\!]]{v/w}\!\!\!\!\rightarrow (p, r) \;\Rightarrow\; \zeta(p)(r)$. Now for any (p, r) appearing in a trace of $[\![A]\!]$ we have $\zeta(p)(r)$ and, therefore, by Definition 8 $[\![A]\!]$ and $[\![\textsc{inv}^\zeta(A)]\!]$ have the same outgoing transitions from (p, r). Thus for all $p \in Q$, $r \in R$, $v \in I$ and $w_0, w, w_1 \in O^*$ we have

$$\begin{aligned}
&\xrightarrow[[\![A]\!]]{/w_0} (p^0, r^0) \xrightarrow[[\![A]\!]]{v/w}\!\!\!\!\rightarrow (p, r) \xrightarrow[[\![A]\!]]{/w_1}\\
\Leftrightarrow\; &\xrightarrow[[\![\textsc{inv}^\zeta(A)]\!]]{/w_0} (p^0, r^0) \xrightarrow[[\![\textsc{inv}^\zeta(A)]\!]]{v/w}\!\!\!\!\rightarrow (p, r) \xrightarrow[[\![\textsc{inv}^\zeta(A)]\!]]{/w_1}
\end{aligned}$$

Therefore $\mathscr{T}(\textsc{inv}^\zeta(A)) = \mathscr{T}(A)$. ⊠

Using this strengthening the ST in Example 8 could be further reduced with the invariants $\zeta(q_1) \stackrel{\text{def}}{=} y \geq 3$, $\zeta(q_2) \stackrel{\text{def}}{=} y < 3$ and $\zeta(q_0) \stackrel{\text{def}}{=} \zeta(q_3) \stackrel{\text{def}}{=} y = 0$. In Example 8 these invariants immediately follow from the conjunction of constraints from incoming transitions for each control state. In general reachability analysis techniques, such as PDR [12], or other invariant condition generation algorithms could be used. This strengthening technique also implies that transitions for STs should be written in a non-defensive way to enable the most reduction.

7 Implementation

We have implemented an ST state reduction tool that builds upon a framework and algorithms developed in [35] that are available in the open source Microsoft

Automata library [1]. The tool is an integrated part of a tool chain which composes pipelines of STs and generates efficient code for them.

The tool allows STs to be specified as (i) imperative code in a subset of `C#`, (ii) XPath expressions or Regular expressions with capture groups hierarchically composed to other STs, (iii) compositions of other STs. For compositions the tool produces a single ST using a fusion algorithm that uses Z3 to prune unsatisfiable transitions.

7.1 Huffman coding

We have extended the tool with support for generating SFTs that perform Huffman encoding and decoding [26]. Huffman coding is an optimal prefix code that assigns variable length bit patterns to symbols. Symbols are assigned bit patterns according to their frequency in such a way that more common symbols are represented with shorter bit patterns. Huffman coding is only one class of prefix codes. We will now give constructions of SFT decoders and encoders for any prefix code.

Definition 13. A *prefix code tree* is a tuple $(Q, E, q_0, \Sigma, S, l_\Sigma, l_S)$, where Q is a set of at least two vertices, $q_0 \in Q$ is the root, $E \subset Q \times Q$ s.t. (Q, E, q_0) is a tree rooted at q_0 with all edges in E directed away from the root, Σ is the *coding alphabet* and S is the *symbol alphabet*.

$l_\Sigma : E \to \Sigma$ is a function s.t. $\forall (p,q),(p,q') \in E : l_\Sigma(p,q) \neq l_\Sigma(p,q')$. Let Q_{leaves} be the leaves of the tree (nodes with no outgoing edges). $l_S : Q_{leaves} \to S$ associates leaves to symbols.

Given a prefix code tree P the *decoder for* P is an SFT $(\Sigma, S, Q \setminus Q_{leaves}, q_0, \epsilon, T, \{(q_0, true, \epsilon)\})$ where:

$$\begin{aligned} T = &\{\, p \xrightarrow{x = l_\Sigma(p,q)/\epsilon} q \mid (p,q) \in E \setminus (Q \times Q_{leaves}) \,\} \\ &\cup \{\, p \xrightarrow{x = l_\Sigma(p,q)/[l_S(q)]} q_0 \mid (p,q) \in E \cap (Q \times Q_{leaves}) \,\} \end{aligned}$$

The *encoder for* P is an SFT $(S, \Sigma, \{p_0\}, p_0, \epsilon, T, \{(p_0, true, \epsilon)\})$ where:

$$\begin{aligned} T &= \{\, p_0 \xrightarrow{x = l_S(q)/code(q)} p_0 \mid q \in Q_{leaves} \,\} \\ code(q) &= \text{let } q_1, \ldots, q_n \text{ be the unique path in } E \text{ from } q_0 \text{ to } q \text{ in} \\ &\quad [l_\Sigma(q_0,q_1), l_\Sigma(q_1,q_2), \ldots, l_\Sigma(q_n,q)] \end{aligned}$$

We will show in our evaluation that Huffman decoders in particular are very amenable to state reductions when composed with other transducers.

8 Evaluation

We evaluate the tool on STs drawn from [35]. These STs are fused pipelines consisting of real-world stream processing computations. The first four pipelines

represent various stream processing scenarios: **Base64-avg** calculates a running average (window of 10) for Base64[6] encoded `int`s and re-encodes the results in Base64. **CSV-max** decodes an UTF-8 encoded CSV file to UTF-16, extracts the third column with a regular expression and finds the maximum length of these strings. The output is a single UTF-8 encoded decimal formatted integer. **Base64-delta** reads Base64 encoded `int`s and outputs deltas of successive inputs as UTF-8 encoded decimal integers on separate lines. **UTF8-lines** decodes an UTF-8 encoded file to UTF-16 and counts the number of newline characters. The output is a single UTF-8 encoded decimal formatted integer.

The following pipelines focus on CSV parsing scenarios using the regex based parsing offered by the tool: **CC-id** is written for a dataset of consumer complaints received by the U.S. Consumer Financial Protection Bureau. The pipeline produces the maximum value for the ID column. **CHSI-cancer** is written for a dataset on health indicators from the U.S. Department of Health & Human Services. The pipeline produces the average lung cancer deaths for counties in the dataset. **SBO-employees** is written for a dataset on business owners from the U.S. Census Bureau. The pipeline finds the maximum number of employees for businesses in the dataset.

Each of these pipelines consist of four phases: (i) decode UTF-8 to UTF-16, (ii) parse a column as an `int` using a regular expression based parser, (iii) run a query (maximum, minimum or average), and (iv) output the result as a sequence of bytes. The pipelines differ only in the regular expression and query used.

The following pipelines are written for XML processing scenarios and use an XPath based transducer for extracting the relevant data: **TPC-DI-SQL** The dataset was generated by a tool from the TPC-DI benchmark [34]. The pipeline extracts ids of accounts from customer records and for each outputs an SQL insert statement. **PIR-proteins** The dataset is a protein dataset from the U.S. based National Biomedical Research Foundation. The pipeline extracts the lengths of all proteins in the dataset and outputs the average length. **DBLP-oldest** The dataset is bibliographic information from the Digital Bibliography Library Project. The pipeline extracts the publication year of each article and outputs the earliest year. **MONDIAL-pop** Mondial is a dataset extracted from various geographical Web data sources. The pipeline extracts the population of each city in the dataset and outputs the highest population.

Additionally we evaluate one pipeline using the new Huffman decoding described in Section 7.1: **Huffman** decodes a Huffman encoded ASCII file and counts the newline characters. The data for creating the Huffman tree is Herman Melville's "Moby Dick".

For each pipeline in our evaluation we produce a single ST as the composition of the whole pipeline and apply the state reduction algorithm to it. In Figure 3 we report the number of control states removed, the number of control states remaining and the time taken by the state reduction.

For the pipelines in Figure 3 an average of 25% of the control states are removed. The amount of state reduction available is highly variable: for Huffman

[6] See `https://en.wikipedia.org/wiki/Base64`.

Pipeline	Removed	$\|Q\|$	Time	Pipeline	Removed	$\|Q\|$	Time
Base64-delta	10	18	39.9 s	SBO-employees	4	36	0.2 s
CSV-max	4	26	18.0 s	TPC-DI-SQL	68	457	44.1 s
Base64-avg	114	166	99.6 s	PIR-proteins	80	355	196.1 s
UTF8-lines	0	5	0.03 s	DBLP-oldest	36	219	9.8 s
CC-id	2024	983	4.4 s	MONDIAL-pop	56	319	12.4 s
CHSI-cancer	12	558	2.2 s	Huffman	915	360	2.6 s

Fig. 3. Control states removed and remaining, and total time taken.

72% of its control states are removed, as counting lines makes all control states that for all inputs output something else than an end-of-line character indistinguishable. On the other hand for UTF8-lines there is nothing left to remove as neither of the single control state line counting or integer formatting STs composed onto the UTF8 decoder make any control states (that correspond to encodings of different lengths) indistinguishable.

In general we see our state reduction algorithm being effective when some control states become indistinguishable due to composition. For example we can see great reduction in the regex and XML processing pipelines due to multi-byte encodings from the UTF8-to-UTF16 decoder being handled equivalently in parts of the regex or XPath matchers.

9 Related work

Minimization of finite state transducers. Minimality of sequential transducers was first studied by Choffrut [14]. Mohri's original work on minimizing sequential finite state transducers appears in [31] and introduces the notion of quasi-determinization of NFAs, that is similar to classical shortest paths problems in weighted directed graphs. An incremental algorithm of minimizing acyclic finite state transducers is described in [30]. A notion of minimization of finite state transducers in natural language processing is studied in [20] by using flag diacritics. We stated Mohri's minimization algorithm so it applies to sequential transducers with final outputs. The notion of sequential functions with final outputs are often called *subsequential* functions and were originally introduced in [36]. Some algorithms for finitely subsequential transducers are investigated in [6].

Minimization of symbolic automata. The concept of automata with predicates instead of concrete symbols was first mentioned in [41] and was first discussed in [37] in the context of natural language processing. An algorithm for minimizing SFAs, based on Hopcroft's partition refinement, is developed in [17]. The MONA implementation [23] provides decision procedures for monadic second-order logic, and uses also highly-optimized and minimized BDD-based representation of automata [27]. The SFA minimization problem is also related to minimizing control flow graphs of programs, which is studied in [15] by reduction to a variant of classical automata minimization.

Nondeterministic case. Our main theorem, Theorem 4, allows the ST or SFT to be nondeterministic and the resulting SFA may, likewise, be nondeterministic. Recently a state reduction algorithm has been developed for nondeterministic SFAs that is based on computing forward bisimulations [18]. A forward bisimulation $\sim$ preserves state indistinguishability and therefore Theorem 4(a) applies. There are numerous other algorithms, developed for nondeterministic automata [4,3,7,24,29] that may likewise be extensible for SFAs.

Transducers with registers. Streaming string transducers [9] are another type of transducer that include a register as part of their state. A significant departure from symbolic transducers is that the contents of a string held in a register can be included in the output as a flattened part of the output sequence, thus making output in a single transition be potentially variable in length. It is unclear how our techniques would apply to streaming string transducers. In particular, streaming string transducers with data values are in general not closed under composition [9, Proposition 4]. Register minimization is a form of resource minimization that aims at reducing the number of registers and has been studied for streaming string transducers [10]. Register minimization has also been studied for cost register automata [8,19].

10 Conclusions

Similarly to products of DFAs and subset constructions of NFAs, compositions of symbolic transducers (STs) present an important target for minimization. Composition can often introduce indistinguishable control states, which makes it possible to leverage minimization algorithms for symbolic finite automata (SFAs) through an encoding approach. Combined with a quasi-determinization step our approach guarantees minimality for symbolic finite transducers (SFTs) when they are deterministic.

Minimizing an SFA encoding of an ST provides a very general control state reduction approach, which is agnostic to how the SFA is minimized as long as indistinguishable equivalence classes of control states are identified. The approach is even agnostic to nondeterminism and as such enables nondeterministic STs to be targeted as minimization algorithms for nondeterministic SFAs become available. To allow state reduction in STs where indistinguishability is due to register carried constraints, an ST can be strengthened using known invariants on the register.

On a set of STs composed from real-world streaming computations our state reduction algorithm removes an average of 25% showing that the approach is effective even with the over-approximation involved in the SFA encoding.

References

1. Automata library. `https://github.com/AutomataDotNet/Automata`.
2. Emoticons, Unicode standard v9.0. `http://unicode.org/charts/PDF/U1F600.pdf`.

3. P. Abdulla, J. Deneux, L. Kaati, and M. Nilsson. Minimization of non-deterministic automata with large alphabets. In *10th International Conference on Implementation and Application of Automata (CIAA'05), proceedings*, pages 31–42. Springer, 2006.
4. P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Composed bisimulation for tree automata. In *13th International Conference on Implementation and Applications of Automata (CIAA'08), proceedings*, pages 212–222. Springer, 2008.
5. A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley, 2006.
6. C. Allauzen and M. Mohri. Finitely subsequential transducers. *International Journal of Foundations of Computer Science*, 14(6):983–994, 2003.
7. R. Almeida, L. Holík, and R. Mayr. Reduction of nondeterministic tree automata. In *22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16), proceedings*, pages 717–735. Springer, 2016.
8. R. Alur and M. Raghothaman. Decision problems for additive regular functions. In *40th International Conference on Automata, Languages, and Programming - Volume Part II (ICALP'13), proceedings*, pages 37–48. Springer, 2013.
9. R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. *SIGPLAN Not. - POPL'11*, 46(1):599–610, 2011.
10. F. Baschenis, O. Gauwin, A. Muscholl, and G. Puppis. Minimizing resources of sweeping and streaming string transducers. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
11. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comput.*, 45(9):993–1002, 1996.
12. A. R. Bradley. SAT-based model checking without unrolling. In *12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'11), proceedings*, pages 70–87. Springer, 2011.
13. D. Buchfuhrer and C. Umans. The complexity of Boolean formula minimization. *J. Comput. Syst. Sci.*, 77(1):142–153, 2011.
14. C. Choffrut. *Contribution à l'étude de quelques familles remarquables de fonctions rationnelles*. PhD thesis, Universit Paris 7, Paris, France, 1978.
15. T. Colcombet and P. Fradet. Enforcing trace properties by program transformation. In *27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'00), proceedings*, pages 54–66. ACM, 2000.
16. M. Dalla Preda, R. Giacobazzi, A. Lakhotia, and I. Mastroeni. Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables. *SIGPLAN Not. - POPL'15*, 50(1):329–341, 2015.
17. L. D'Antoni and M. Veanes. Minimization of symbolic automata. *SIGPLAN Not. - POPL'14*, 49(1):541–553, 2014.
18. L. D'Antoni and M. Veanes. Forward bisimulations for nondeterministic symbolic finite automata. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17), proceedings*. Springer, 2017.
19. L. Daviaud, P.-A. Reynier, and J.-M. Talbot. A generalised twinning property for minimisation of cost register automata. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16), proceedings*, pages 857–866. ACM, 2016.
20. S. Drobac, K. Lindén, T. Pirinen, and M. Silfverberg. Heuristic hyper-minimization of finite state lexicons. In *Ninth International Conference on Language Resources and Evaluation (LREC'14), proceedings*. ELRA, 2014.
21. D. D'Souza and P. Shankar, editors. *Modern Applications of Automata Theory*, volume 2 of *IISc Research Monographs Series*. World Scientific, 2012.

22. Z. Fülöp and H. Vogler. Forward and backward application of symbolic tree transducers. *Acta Informatica*, 51(5):297–325, 2014.
23. J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems: First International Workshop (TACAS'95), selected papers*, pages 89–110. Springer, 1995.
24. J. Högberg, A. Maletti, and J. May. Backward and forward bisimulation minimisation of tree automata. In *12th International Conference on Implementation and Application of Automata (CIAA'07), proceedings*, pages 109–121. Springer, 2007.
25. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Fast and precise sanitizer analysis with Bek. In *20th USENIX Conference on Security (SEC'11), proceedings*. USENIX Association, 2011.
26. D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
27. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. *International Journal of Foundations of Computer Science*, 13(4):571–586, 2002.
28. A. Manuel and R. Ramanujam. Automata over infinite alphabets. In D. D'Souza and P. Shankar, editors, *Modern Applications of Automata Theory*, pages 529–554. World Scientific, 2012.
29. R. Mayr and L. Clemente. Advanced automata minimization. *SIGPLAN Not. - POPL'13*, 48(1):63–74, 2013.
30. S. Mesfar and M. Silberztein. Transducer minimization and information compression for NooJ dictionaries. In *2009 Conference on Finite-State Methods and Natural Language Processing (FSMNLP'2008), proceedings*, pages 110–121. IOS Press, 2009.
31. M. Mohri. Minimization of sequential transducers. In *Combinatorial Pattern Matching: 5th Annual Symposium (CPM'94)*, pages 151–163. Springer, 1994.
32. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
33. M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1-2):177–201, 2000.
34. M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield. TPC-DI: The first industry benchmark for data integration. *Proceedings of the VLDB Endowment*, 7(13):1367–1378, 2014.
35. O. Saarikivi, M. Veanes, T. Mytkowicz, and M. Musuvathi. Fusing effectful comprehensions. In *38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'17), proceedings*. ACM, 2017.
36. M. P. Schützenberger. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4:47–57, 1977.
37. G. van Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.
38. M. Veanes, P. d. Halleux, and N. Tillmann. Rex: Symbolic regular expression explorer. In *2010 Third International Conference on Software Testing, Verification and Validation (ICST'10), proceedings*, pages 498–507. IEEE, 2010.
39. M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner. Symbolic finite state transducers: Algorithms and applications. *SIGPLAN Not. - POPL'12*, 47(1):137–150, 2012.
40. M. Veanes, T. Mytkowicz, D. Molnar, and B. Livshits. Data-parallel string-manipulating programs. *SIGPLAN Not. - POPL'15*, 50(1):139–152, 2015.
41. B. W. Watson. Implementing and using finite automata toolkits. In *Extended finite state models of language*, pages 19–36. Cambridge University Press, 1999.