Equivalence of Extended Symbolic Finite Transducers *

Loris D'Antoni[1] and Margus Veanes[2]

[1] University of Pennsylvania
lorisdan@cis.upenn.edu

[2] Microsoft Research
margus@microsoft.com

Abstract. Symbolic Finite Transducers augment classic transducers with symbolic alphabets represented as parametric theories. Such extension enables succinctness and the use of potentially infinite alphabets while preserving closure and decidability properties. Extended Symbolic Finite Transducers further extend these objects by allowing transitions to read consecutive input elements in a single step. While when the alphabet is finite this extension does not add expressiveness, it does so when the alphabet is symbolic. We show how such increase in expressiveness causes decision problems such as equivalence to become undecidable and closure properties such as composition to stop holding. We also investigate how the automata counterpart, Extended Symbolic Finite Automata, differs from Symbolic Finite Automata. We then introduce the subclass of Cartesian Extended Symbolic Finite Transducers in which guards are limited to conjunctions of unary predicates. Our main result is an equivalence algorithm for such subclass in the single-valued case. Finally, we model real world problems with Cartesian Extended Symbolic Finite Transducers and use the equivalence algorithm to prove their correctness.

1 Introduction

Finite automata have proven to be an effective tool in a wide range of applications, from regular expressions to network packet inspection [17]. Finite transducers extend finite automata with outputs and can model functions from strings to strings such as natural language transformations [13]. Due to their closure and decidability properties, these models are widely used in practice but they have three major disadvantages: 1) their number of transitions usually "blows up" when dealing with large alphabets; 2) they cannot model infinite alphabets; and 3) transitions cannot express relations between adjacent input symbols.

Symbolic Finite Automata/Transducers [18] or SFAs/SFTs respectively, are an extension of traditional automata and transducers that attempts to solve problems 1 and 2 above by allowing transitions to be labeled with arbitrary

* Microsoft Research Technical Report MSR-TR-2013-4

predicates in a specified theory. When such theory is decidable SFAs and SFTs enjoy the same properties of finite automata and transducers, such as closure under composition and decidability of equivalence (for single-valued SFTs). In [18], Symbolic Transducers or STs (SFTs with registers) are proposed in order to cope with the third problem above. STs are however undecidable with respect to most analysis problems, even emptiness.

In our previous work on the topic of analysis of string coders [3], we introduce *Extended Symbolic Finite Automata/Transducers* or ESFAs/ESFTs, that add finite lookahead to SFAs/SFTs. This extension allows to read multiple input symbols in a single transition and combine their values in the output. ESFTs can be viewed as a subclass of STs with a restricted use of registers that mimic "lookbehind". This view is used in [3] to map ESFTs directly to STs in order to overcome the problem that ESFTs are not closed under composition. In other words, it addresses the composition problem by first converting ESFTs to STs, then composing the STs, and finally converting the result back into an ESFT using a semidecision procedure. The formal properties of ESFTs have not been fully understood yet. From the point of view of analysis, the key operations that are desired are *composition* and *equivalence* (for single-valued ESFTs). Then, for example, the functional correctness of a string (*encoder*,*decoder*) pair (E, D) (for example UTF8 to UTF16 encoding) can be decided by checking the equivalence of $\lambda x.D(E(x))$ with $\lambda x.x$. Other properties, such as commutativity and idempotence, also depend on composition and equivalence.

The topic that is left open in [3] is decidability of equivalence checking of ESFTs. Our main theoretical contribution in this paper is a complete classification, in terms of guard complexity and lookahead, of the cases in which the equivalence problem is decidable for ESFTs. We first show that one-equality or equivalence in the single-valued case is in general undecidable, contrasting the finite alphabet setting where lookahead does not matter [20, Theorem 2.17]. We then introduce the notion of Cartesian ESFT, in which transition guards are constrained to be conjunctions of unary predicates, and show that one-equality and equivalence are decidable for single-valued Cartesian ESFTs. This is a proper extension of the decidability result of one-equality of SFTs [18]. The key tool that we need to prove the result is Lemma 2.

We also analyze basic properties of ESFAs and show how they differ from SFAs. We prove ESFAs to be not closed under intersection and show that equivalence and universality of ESFAs are both undecidable problems.

Applications. We present four applications of our models in different areas. We first extend the result of [3] by proving the correctness of four real world string encoders. The new equivalence algorithm is a full decision procedure for the Cartesian case, unlike the semidecision procedure in [3] that may fail to terminate in some incorrect instances. Our second and third applications are in the context of networking and present new classes of programs that can be modeled as ESFAs/ESFTs. We show how 1) ESFAs can be used for the task of deep-packet inspection, and 2) ESFTs can succinctly represent transformations between headers of different network protocols. Our fourth case study shows the use of additional theories for the analysis of list manipulating programs.

Contributions. In summary, we offer the following contributions:

- we study the closure and decidability properties of ESFAs (Section 3.1);
- we study the equivalence problem for ESFTs (Section 3.2):
  - we prove the equivalence of single-valued ESFTs to be undecidable;
  - we present a novel algorithm for the equivalence of single-valued Cartesian ESFTs;
- we extend the negative result on ESFTs composition presented in [3] (Section 3.3); and
- we analyze the performance of the equivalence algorithm for "Cartesian" ESFTs on real examples and propose new applications for ESFAs and ESFTs (Section 4).

We finally summarize previous work and conclude (Section 5 and 6).

2 Extended Symbolic Finite Transducers

We assume a *background structure* that has a recursively enumerable (r.e.) multi-typed carrier set or *background universe* $\mathcal{U}$, and is equipped with a language of function and relation symbols with fixed interpretations. Definitions below are given with $\mathcal{U}$ as an implicit parameter. We assume closure under Boolean operations and equality. We use λ-expressions for representing anonymous functions that we call *λ-terms*. A Boolean λ-term $\lambda x.\varphi(x)$, where x is a variable of type σ is called a σ-predicate. We use standard first-order logic and follow the notational conventions that are consistent with the original definition of symbolic transducers [18]. The universe is multi-typed with $\mathcal{U}^\tau$ denoting the subuniverse of elements of type τ. We write Σ for $\mathcal{U}^\sigma$ and Γ for $\mathcal{U}^\gamma$.

A *label theory* is given by a recursively enumerable set Ψ of formulas that is closed under Boolean operations, substitution, equality and if-then-else terms. A label theory Ψ is *decidable* when satisfiability for $\varphi \in \Psi$, $\mathit{IsSat}(\varphi)$, is decidable.

For σ-predicates φ, we assume an effective *witness* function $\mathscr{W}$ such that, if $\mathit{IsSat}(\varphi)$ then $\mathscr{W}(\varphi) \in [\![\varphi]\!]$; φ is *valid*, $\mathit{IsValid}(\varphi)$, when $\forall x \varphi(x)$ holds.

We are studying in this paper an extension of SFTs with *lookahead*, called *extended* SFTs or *ESFTs*. Originally, ESFTs were introduced in [3] for the purposes of analyzing string encoders and decoders, where a semi-decision procedure was provided for converting STs (SFTs with registers) into ESFTs.

Definition 1. An *Extended Symbolic Finite Transducer (ESFT)* with *input type* σ and *output type* γ is a tuple $A = (Q, q^0, R)$,

- Q is a finite set of *states*;
- $q^0 \in Q$ is the *initial state*;
- R is a finite set of *rules*, $R = \Delta \cup F$, where
- Δ is a set of *transitions* $r = (p, \ell, \varphi, f, q)$, denoted $p \xrightarrow[\ell]{\varphi/f} q$, where

$p \in Q$ is the *start* state of r;
$\ell \geq 1$ is the *lookahead* of r;
φ, the *guard* of r, is a σ^ℓ-predicate;
f, the *output* of r, is a $(\sigma^\ell \to \gamma)$-sequence;
$q \in Q$ is the *continuation* state of r.

– F is a set of *finalizers* $r = (p, \ell, \varphi, f)$, denoted $p \xrightarrow[\ell]{\varphi/f} \bullet$, with components as above and where ℓ may be 0.

The *lookahead* of A is the maximum of all lookaheads of rules in R. An ESFT all of whose rules have output [] is an *Extended Symbolic Finite Automaton* (*ESFA*).

A finalizer is a rule without a continuation state. A finalizer with lookahead ℓ is used when the end of the input sequence has been reached with *exactly* ℓ input elements remaining. A finalizer is a generalization of a final state. In a classical setting, finalizers can be avoided by adding a new symbol to the alphabet that is only used to mark the end of the input. In the presence of arbitrary input types, this is not always possible without affecting the theory, e.g., when the input type is $\mathbb{Z}$ then that symbol would have to be outside $\mathbb{Z}$. The following example represents typical (realistic) ESFTs over a label theory of linear modular arithmetic. We use the following abbreviated notation for rules, by omitting explicit λ's. We write

$$p \xrightarrow[\ell]{\varphi(\bar{x})/[f_1(\bar{x}),\ldots,f_k(\bar{x})]} q \quad \text{for} \quad p \xrightarrow[\ell]{\lambda\bar{x}.\varphi(\bar{x})/\lambda\bar{x}.[f_1(\bar{x}),\ldots,f_k(\bar{x})]} q,$$

where φ and f_i are terms whose free variables are among $\bar{x} = (x_0, \ldots, x_{\ell-1})$.

Example 1. The example illustrates standard Base64 encoding that is used to transfer binary data in textual format, e.g., in email via MIME. The digits of the encoding are chosen in the safe ASCII range of characters that remain unmodified during transport over textual media. Assume that the input type and the output type are both BYTE, that is the set of integers between 0 and 255. *Base64encode* in an ESFT with one state and four rules:

$$p \xrightarrow[3]{\mathit{true}/[\ulcorner b_2^7(x_0)\urcorner,\ \ulcorner(b_0^1(x_0)\ll 4)|b_4^7(x_1)\urcorner,\ \ulcorner(b_0^3(x_1)\ll 2)|b_6^7(x_2)\urcorner,\ \ulcorner b_0^5(x_2)\urcorner]} p$$

$$p \xrightarrow[0]{\mathit{true}/[]} \bullet \qquad p \xrightarrow[1]{\mathit{true}/[\ulcorner b_2^7(x_0)\urcorner,\ \ulcorner b_0^1(x_0)\ll 4\urcorner,\ \text{`=',\ `='}]} \bullet$$

$$p \xrightarrow[2]{\mathit{true}/[\ulcorner b_2^7(x_0)\urcorner,\ \ulcorner(b_0^1(x_0)\ll 4)|b_4^7(x_1)\urcorner,\ \ulcorner b_0^3(x_1)\ll 2\urcorner,\ \text{`='}]} \bullet$$

where $b_n^m(x)$ extracts bits m through n from x, e.g., $b_2^3(13) = 3$, $x|y$ is bitwise OR of x and y, $x \ll k$ is x shifted left by k bits, and $\ulcorner x \urcorner$ is the mapping

$$\ulcorner x \urcorner \stackrel{\text{def}}{=} (x{\leq}25\,?\,x{+}65 : (x{\leq}51\,?\,x{+}71 : (x{\leq}61\,?\,x{-}4 : (x{=}62\,?\,\text{`+'} : \text{`/'}))))$$

of values between 0 and 63 into a standardized sequence of safe ASCII character codes. The last two finalizers correspond to the cases when the length of the input sequence is not a multiple of three. Observe that the length of the output sequence is always a multiple of four. The character **'='** (61) is used as a padding character that is not a base64 digit. i.e., **'='** is not in the range of $\ulcorner x \urcorner$.

Base64decode in an ESFT that decodes a base64 encoded sequence back into the original byte sequence. *Base64decode* has also one state and four rules:

$$q \xrightarrow[4]{\bigwedge_{i=0}^{3} \beta_{64}(x_i)/[(\llcorner x_0 \lrcorner \ll 2)|b_4^5(\llcorner x_1 \lrcorner),\ (b_0^3(\llcorner x_1 \lrcorner)\ll 4)|b_2^5(\llcorner x_2 \lrcorner),\ (b_0^1(\llcorner x_2 \lrcorner)\ll 6)|\llcorner x_3 \lrcorner]} q$$

$$q \xrightarrow[0]{true/[]} \bullet \qquad q \xrightarrow[4]{\beta_{64}(x_0)\wedge\beta'_{64}(x_1)\wedge x_2=\text{'='}\wedge x_3=\text{'='}/[(\llcorner x_0 \lrcorner \ll 2)|b_4^5(\llcorner x_1 \lrcorner)]} \bullet$$

$$q \xrightarrow[4]{\beta_{64}(x_0)\wedge\beta_{64}(x_1)\wedge\beta''_{64}(x_2)\wedge x_3=\text{'='}/[(\llcorner x_0 \lrcorner \ll 2)|b_4^5(\llcorner x_1 \lrcorner),\ (b_0^3(\llcorner x_1 \lrcorner)\ll 4)|b_2^5(\llcorner x_2 \lrcorner)]} \bullet$$

The function $\llcorner y \lrcorner$ is the inverse of $\ulcorner x \urcorner$, i.e., $\llcorner \ulcorner x \urcorner \lrcorner = x$, for $0 \le x \le 63$. The predicate $\beta_{64}(y)$ is true iff y is a valid Base64 digit, i.e., $y = \ulcorner x \urcorner$ for some x, $0 \le x \le 63$. The predicates $\beta'_{64}(y)$ and $\beta''_{64}(y)$ are restricted versions of $\beta_{64}(y)$. Unlike *Base64encode*, *Base64decode* does not accept all input sequences of bytes. Sequences that do not correspond to any encoding are rejected.[3] $\boxtimes$

In the sequel let $A = (Q, q^0, R)$, $R = \Delta \cup F$, be a fixed ESFT with input type σ and output type γ. The semantics of rules in R is as follows:

$$[\![p \xrightarrow[\ell]{\varphi/f} q]\!] \stackrel{\text{def}}{=} \{p \xrightarrow{[a_0,\dots,a_{\ell-1}]/[\![f]\!](a_0,\dots,a_{\ell-1})} q \mid (a_0,\dots,a_{\ell-1}) \in [\![\varphi]\!]\}$$

We write $s_1 \cdot s_2$ for concatenation of two sequences s_1 and s_2.

Definition 2. For $u \in \Sigma^*, v \in \Gamma^*, q \in Q, q' \in Q \cup \{\bullet\}$, define $q \overset{u/v}{\twoheadrightarrow}_A q'$ as follows: there exists $n \ge 0$ and $\{p_i \xrightarrow{u_i/v_i} p_{i+1} \mid i \le n\} \subseteq [\![R]\!]$ such that

$$u = u_0 \cdot u_1 \cdots u_n, \quad v = v_0 \cdot v_1 \cdots v_n, \quad q = p_0, \quad q' = p_{n+1}.$$

Let also $q \overset{[]/[]}{\twoheadrightarrow}_A q$ for all $q \in Q_A$.

Definition 3. The *transduction of* A, $\mathscr{T}_A(u) \stackrel{\text{def}}{=} \{v \mid q^0 \overset{u/v}{\twoheadrightarrow} \bullet\}$.

The following subclass of ESFTs captures transductions that behave as partial functions from Σ^* to Γ^*.

Definition 4. A function $\mathbf{f} : X \to 2^Y$ is *single-valued* if $|\mathbf{f}(x)| \le 1$ for all $x \in X$. An ESFT A is *single-valued* if $\mathscr{T}_A$ is single-valued.

A sufficient condition for single-valuedness is determinism. We define $\varphi \curlywedge \psi$, where φ is a σ^m-predicate and ψ a σ^n-predicate, as the $\sigma^{\max(m,n)}$-predicate $\lambda(x_1,\dots,x_{\max(m,n)}).\varphi(x_1,\dots,x_m) \wedge \psi(x_1,\dots,x_n)$. We define *equivalence of* f *and* g *modulo* φ, $f \equiv_\varphi g$, as: $\mathit{IsValid}(\lambda\bar{x}.(\varphi(\bar{x}) \Rightarrow f(\bar{x}) = g(\bar{x})))$.

Definition 5. A is *deterministic* if for all $p \xrightarrow[\ell]{\varphi/f} q, p \xrightarrow[\ell']{\varphi'/f'} q' \in R$:

(a) Assume $q, q' \in Q$. If $\mathit{IsSat}(\varphi \curlywedge \varphi')$ then $q = q'$, $\ell = \ell'$ and $f \equiv_{\varphi \curlywedge \varphi'} f'$.
(b) Assume $q = q' = \bullet$. If $\mathit{IsSat}(\varphi \curlywedge \varphi')$ and $\ell = \ell'$ then $f \equiv_{\varphi \curlywedge \varphi'} f'$.

[3] For more information see `http://www.rise4fun.com/Bek/tutorial/base64`.

(c) Assume $q \in Q$ and $q' = \bullet$. If $IsSat(\varphi \curlywedge \varphi')$ then $\ell > \ell'$.

Intuitively, determinism means that no two rules may overlap. It follows from the definitions that if A is deterministic then A is single-valued. Both ESFTs in Example 1 are deterministic.

The *domain* of a function $\mathbf{f} : X \to 2^Y$ is $\mathscr{D}(\mathbf{f}) \stackrel{\text{def}}{=} \{x \in X \mid \mathbf{f}(x) \neq \emptyset\}$ and for an ESFT A, $\mathscr{D}(A) \stackrel{\text{def}}{=} \mathscr{D}(\mathscr{T}_A)$. When A is single-valued, and $u \in \mathscr{D}(A)$, we treat A as a partial function from Σ^* to Γ^* and write $A(u)$ for the value v such that $\mathscr{T}_A(u) = \{v\}$. For example, *Base64encode*(`"Foo"`) = `"Rm9v"` and *Base64decode*(`"QmFy"`) = `"Bar"`.

Cartesian ESFTs. We introduce a subclass of ESFTs that plays an important role in practice. A binary relation R over X is *Cartesian over* X if R is the Cartesian product $R_1 \times R_2$ of some $R_1, R_2 \subseteq X$. The definition is lifted to n-ary relations and σ^n-predicates for $n \geq 2$ in the obvious way. In order to decide if a satisfiable σ^n-predicate φ is Cartesian over σ, let $(a_0, \ldots, a_{n-1}) = \mathscr{W}(\varphi)$ and perform the following validity check:

$$IsCartesian(\varphi) \stackrel{\text{def}}{=} \forall \bar{x}\, (\varphi(\bar{x}) \Leftrightarrow \bigwedge_{i<n} \varphi(a_0, \ldots, a_{i-1}, x_i, a_{i+1}, \ldots, a_{n-1}))$$

In other words, a σ^n-predicate φ is Cartesian over σ if φ can be rewritten equivalently as a conjunction of n independent σ-predicates.

Definition 6. An ESFT (ESFA) is *Cartesian* if all its guards are Cartesian.

Both ESFTs in Example 1 are Cartesian. *Base64encode* trivially so, while the guards of all rules of *Base64decode* are conjunctions of independent unary predicates. In contrast, a predicate such as $\lambda(x_0, x_1).x_0 = x_1$ is not Cartesian.

Note that *IsCartesian*(φ) is decidable by using the decision procedure of the label theory. Namely, decide unsatisfiability of $\neg$*IsCartesian*(φ).

3 ESFAs and ESFTs Properties

We prove some basic properties of ESFAs and ESFTs and show how they drastically differ from SFAs and SFTs. While in the finite alphabet case adding finite lookahead does not add any expressiveness, in the symbolic setting most properties become undecidable. We first investigate basic ESFAs properties. Then we analyze ESFTs equivalence and propose a new one-equivalence algorithm for Cartesian ESFTs. Finally we present some preliminary results on ESFT composition.

3.1 ESFAs Properties

This section analyses standard language properties such as closures and decidability. We show how ESFAs have results similar to those of context free grammars rather than regular languages. We first show how checking whether the intersection of two ESFA definable language is empty is undecidable.

Theorem 1 (Domain Intersection). *Given two ESFAs A and B with lookahead 2 over quantifier free successor arithmetic and tuples, checking whether there exists an input accepted by both A and B is co-r.e.-complete.*

Proof. Recall that a Minsky machine has two registers r_1 and r_2 that can hold natural numbers and a program that is a finite sequence of instructions. Each instruction is one of the following: INC_i (increment r_i and continue with the next instruction); DEC_i (decrement r_i if $r_i > 0$ and continue with the next instruction); $JZ_i(j)$ (if $r_i = 0$ then jump to the j'th instruction else continue with the next instruction). The machine halts when the end of the program is reached. Let M be a Minsky machine with program P. Let $\sigma = \mathbb{N}^3$ represent the type of the snapshot or configuration $(\mathit{program\ counter}, r_1, r_2)$ of M.

Suppose $\pi_j : \sigma \to \mathbb{N}$ projects the j'th element of a k-tuple where $0 \leq j < k$. Construct ESFAs A and B over σ as follows. Let φ^{ini} be the σ-predicate $\lambda x.x = (0,0,0)$ stating that the program counter and both registers are 0. Let φ^{fin} be the final σ-predicate $\lambda x.\pi_0(x) = |P| \wedge \pi_1(x) \neq 0$.

Let φ^{step} be the σ^2-predicate $\lambda(x,x').\bigvee_{i<|P|}\varphi_i^{\mathrm{step}}$ where $\varphi_i^{\mathrm{step}}$ is the formula for the i'th instruction. If the i'th instruction is INC_1 then $\varphi_i^{\mathrm{step}}$ is

$$\pi_0(x) = i \wedge \pi_0(x') = i+1 \wedge \pi_1(x') = \pi_1(x) + 1 \wedge \pi_2(x') = \pi_2(x)$$

If the i'th instruction is $JZ_1(j)$ then φ_i^{step} is

$$\pi_0(x) = i \wedge \pi_0(x') = \mathit{Ite}(\pi_1(x) = 0, j, i+1) \wedge \pi_1(x') = \pi_1(x) \wedge \pi_2(x') = \pi_2(x)$$

Similarly for the other cases. Thus, φ^{step} encodes the valid step relation of M from current configuration x to the next configuration x'. Let

$$A = (\{p_0\}, p_0, \{p_0 \xrightarrow[2]{\varphi^{\mathrm{step}}} p_0, \quad p_0 \xrightarrow[0]{true} \bullet\}), and$$

$$B = (\{q_0, q_1\}, q_0, \{q_0 \xrightarrow[1]{\varphi^{\mathrm{ini}}} q_1, \quad q_1 \xrightarrow[2]{\varphi^{\mathrm{step}}} q_1, \quad q_1 \xrightarrow[1]{\varphi^{\mathrm{fin}}} \bullet\}).$$

So $\alpha \in \mathscr{D}(A) \cap \mathscr{D}(B)$ iff α is a valid computation of M, i.e., $\alpha[0]$ is the initial configuration, $\alpha[i+1]$ is a valid successor configuration of $\alpha[i]$ (this follows from A for all odd $i < |\alpha|$ and from B for all even $i < |\alpha|$), and $\alpha[|\alpha|-1]$ is a halting configuration.

It follows that $\mathscr{D}(A) \cap \mathscr{D}(B) \neq \emptyset$ iff M halts on input $(0,0)$ with a non-zero output in r_1. The latter is an r.e.-complete problem as an instance of Rice's theorem. ⊠

We first show that

Theorem 2 (ESFA Emptiness). *Given an ESFA A it is decidable to determine whether it accepts any input.*

Proof. Given A, we first where we remove all the transitions with unsatisfiable guards. Let's call the new ESFA A'. If A' has a path from the initial state to $\bullet$, then A is not empty. ⊠

The emptiness of a ESFA is a decidable problem while using a proof similar to that for CFGs we prove that checking whether an ESFA accepts every possible input is undecidable.

Theorem 3 (Universality). *Given an ESFA A over σ checking whether it accepts all the sequences in σ^* is undecidable.*

Proof. Let M be a Minsky machine with program P. Let $\sigma = \mathbb{N}^3$ represent the type of the snapshot or configuration (*program counter*, r_1, r_2) of M. Let φ^{ini}, φ^{fin} and φ^{step} be as in Theorem 1.

We construct an ESFA A_M that does not accept all the strings in σ^* iff M halts on input $(0, 0)$ with a non-zero output in r_1. The latter is an r.e.-complete problem as an instance of Rice's theorem.

Let

$$A = (\{p_0, p_1\}, p_0, \{p_0 \xrightarrow[1]{true} p_0, p_0 \xrightarrow[2]{\neg\varphi^{\text{step}}} p_1, \quad p_1 \xrightarrow[1]{true} p_1 p_1 \xrightarrow[0]{true} \bullet\})$$

$$B = (\{q_0, q_1\}, q_0, \{q_0 \xrightarrow[1]{\neg\varphi^{\text{ini}}} q_1, \quad q_1 \xrightarrow[1]{true} q_1, \quad q_1 \xrightarrow[0]{true} \bullet\})$$

$$C = (\{r_0\}, r_0, \{r_0 \xrightarrow[1]{true} r_0, \quad r_0 \xrightarrow[1]{\neg\varphi^{\text{fin}}} \bullet\})$$

$$D = (\{s_0\}, s_0, \{s_0 \xrightarrow[0]{true} \bullet\})$$

A accepts all the M configuration sequences in which one step is wrong, B all those that starts with the wrong initial state, C all those that end in the wrong configuration, and D the empty sequence. We define $A_M = A \cup B \cup C$ using Theorem 4. A_M does not accept all the inputs in σ^* iff M halts on input $(0, 0)$ with a non-zero output in r_1 (i.e. such sequence of configuration wouldn't be accepted by A_M). ⊠

As a consequence of Theorem 3 we have that ESFA equivalence is undecidable.

Corollary 1 (Equivalence). *Given two ESFAs A and B checking whether they accept the same languages is undecidable.*

Combining Theorems 2 and 1 we obtain the following.

Corollary 2 (Intersection). *ESFAs are not closed under intersection.*

Thanks to nondeterminism we can then show that ESFA definable languages are closed under union.

Theorem 4 (Union). *ESFAs are closed under union.*

Proof. Given two ESFAs $A_1 = (Q_1, q_0^1, R_1)$ and $A_2 = (Q_2, q_0^2, R_2)$ over a sort σ we construct an ESFA B over σ such that $\mathscr{D}(C) = \mathscr{D}(A) \cup \mathscr{D}(B)$. C will have states $Q = Q_1 \cup Q_2 \cup \{q_0\}$ and initial state q_0. The transition relation R of C is then defined as follows:

$$R = R1 \cup R2 \cup \{q_0 \xrightarrow[k]{\varphi} q \mid q_0^1 \xrightarrow[k]{\varphi} q \in R_1 \vee q_0^2 \xrightarrow[k]{\varphi} q \in R_2\}$$

. The proof of correctness is straightforward. ⊠

Combining Theorems 2 and 4 we obtain the following.

Corollary 3. *ESFAs are not closed under complement.*

We next show how nondeterministic ESFA are strictly more expressive than their deterministic counterpart.

Theorem 5. *Deterministic ESFAs are not closed under union.*

Proof. Consider A and B from the proof of Theorem 7. They are deterministic, but their union is not definable by a deterministic ESFA. ⊠

Corollary 4. *Nondeterministic ESFAs are strictly more expressive than deterministic ESFAs.*

Finally, Cartesian ESFAs capture exactly the class of SFA definable languages.

Theorem 6 (Cartesian ESFA iff SFA). *SFAs and Cartesian ESFA are equivalent in expressiveness.*

From Theorem 6 we have that Cartesian ESFAs enjoy all the properties of SFAs (regular languages) such as boolean closures and decidable equivalence.

3.2 Equivalence of ESFTs

While the general equivalence problem of $\mathscr{T}_A = \mathscr{T}_B$ is already undecidable for very restricted classes of finite state transducers [7], the problem is decidable for SFTs in the single-valued case. More generally, one-equality of transductions (defined next) is decidable for SFTs (over decidable label theories).

Definition 7. Functions $\mathbf{f}, \mathbf{g} : X \to 2^Y$ are *one-equal*, $\mathbf{f} \overset{1}{=} \mathbf{g}$, if forall $x \in X$, if $x \in \mathscr{D}(\mathbf{f}) \cap \mathscr{D}(\mathbf{g})$ then $|\mathbf{f}(x) \cup \mathbf{g}(x)| = 1$. Let

$$\mathbf{f} \uplus \mathbf{g}(x) \overset{\text{def}}{=} \begin{cases} \mathbf{f}(x) \cup \mathbf{g}(x), & \text{if } x \in \mathscr{D}(\mathbf{f}) \cap \mathscr{D}(\mathbf{g}); \\ \emptyset, & \text{otherwise.} \end{cases}$$

Proposition 1. $\mathbf{f} \overset{1}{=} \mathbf{g}$ *iff* $\mathbf{f} \uplus \mathbf{g}$ *is single-valued.*

Note that $\mathbf{f} \overset{1}{=} \mathbf{f}$ iff $\mathbf{f}$ is single-valued. Thus, one-equality is a more refined notion than single-valuedness, because an effective construction of $A \uplus B$ such that $\mathscr{T}_{A \uplus B} = \mathscr{T}_A \uplus \mathscr{T}_B$ may not always be feasible or even possible for some classes of transducers.

Definition 8. Functions $\mathbf{f}, \mathbf{g} : X \to 2^Y$ are *domain-equivalent* if $\mathscr{D}(\mathbf{f}) = \mathscr{D}(\mathbf{g})$.

Definitions 7 and 8 are lifted to (E)SFTs. For domain-equivalent single-valued transducers A and B, $A \overset{1}{=} B$ implies equivalence of A and B ($\mathscr{T}_A = \mathscr{T}_B$).

A natural question that arises is whether decidability of one-equality of SFTs generalizes to ESFTs. The answer is positive for the subclass of *Cartesian* ESFTs (that includes ESFTs in Example 1), but negative in general. We first show that one-equality of ESFTs over decidable label theories is undecidable in general.

Theorem 7 (One-Equality). *One-equality of ESFTs with lookahead 2 over quantifier free successor arithmetic and tuples is co-re-complete.*

Proof. We give a reduction from the Domain Intersection problem of Theorem 1. Let A_1 and A_2 be ESFAs with lookahead 2 over quantifier free successor arithmetic and tuples. We construct ESFTs A'_i, for $i \in \{1, 2\}$, as follows:

$$A'_i = (Q_{A_i}, q^0_{A_i}, \Delta_{A_i} \cup \{p \xrightarrow[k]{\varphi/[i]} \bullet \mid p \xrightarrow[k]{\varphi} \bullet \in F_{A_i}\})$$

So $\mathscr{T}_{A'_i}(t) = \{[i]\}$ if $t \in \mathscr{D}(A_i)$ and $\mathscr{T}_{A'_i}(t) = \emptyset$ otherwise. Let $\mathbf{f} = \mathscr{T}_{A'_1} \uplus \mathscr{T}_{A'_2}$. So

- $|\mathbf{f}(t)| = 0$ iff $t \notin \mathscr{D}(A_1) \cup \mathscr{D}(A_2)$;
- $|\mathbf{f}(t)| = 1$ iff $t \in \mathscr{D}(A_1) \cup \mathscr{D}(A_2)$ and $t \notin \mathscr{D}(A_1) \cap \mathscr{D}(A_2)$;
- $|\mathbf{f}(t)| = 2$ iff $t \in \mathscr{D}(A_1) \cap \mathscr{D}(A_2)$.

It follows that $A'_1 \overset{1}{=} A'_2$ iff (by Proposition 1) $\mathbf{f}$ is single-valued iff $\mathscr{D}(A_1) \cap \mathscr{D}(A_2) = \emptyset$. Now use Theorem 1. ⊠

The main decidability result of the paper is Theorem 8 that extends the corresponding result for SFTs [18, Theorem 1]. We use the following definitions. A transition, $p \xrightarrow[\ell]{\varphi/f} q$ where $\ell > 1$, φ is Cartesian and $\mathscr{W}(\varphi) = (a_1, \ldots, a_\ell)$, is represented, given $\varphi_i = \lambda x.\varphi(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_\ell)$, by the following path of *split* transitions,

$$p \xrightarrow[1]{\varphi_1/f} p_1 \xrightarrow[1]{\varphi_2/\bot} p_2 \cdots p_{\ell-1} \xrightarrow[1]{\varphi_\ell/\bot} q$$

where p_i for $1 \leq i < \ell$ are new *temporary* states, the output f is postponed until all input elements have been read. Let Δ^{s}_A denote such *split* view of Δ_A. Here we assume that all finalizers have lookahead zero, since we do not assume ESFTs here to be deterministic.

Example 2. It is trivial to transform any ESFT into an equivalent (possibly nondeterministic) form where all finalizers have zero lookahead. Consider the ESFT *Base64encode* in Example 1. In the last two finalizers, replace $\bullet$ with a new state p_1 and and add the new finalizer $p_1 \xrightarrow[0]{true/[]} \bullet$. ⊠

Definition 9. Let A and B be Cartesian ESFTs with same input and output types and zero-lookahead finalizers. The *product* of A and B is the following *product ESFT* $A{\times}B$. The *initial state* $q^0_{A\times B}$ of $A{\times}B$ is (q^0_A, q^0_B). The states and transitions of $A{\times}B$ are obtained as the least fixed point of

$$\left.\begin{array}{r}(p,q)\in Q_{A\times B}\\ p\xrightarrow[1]{\varphi/f}p'\in\varDelta^{\mathrm{s}}_A\\ q\xrightarrow[1]{\psi/g}q'\in\varDelta^{\mathrm{s}}_B\end{array}\right\}\overset{\mathit{IsSat}(\varphi\wedge\psi)}{\Longrightarrow}(p',q')\in Q_{A\times B},\quad(p,q)\xrightarrow[1]{\varphi\wedge\psi/(f,g)}(p',q')\in\varDelta_{A\times B}$$

Let $F_{A\times B}$ be the set of all rules $(p,q)\xrightarrow[0]{\mathit{true}/(v,w)}\bullet$ such that $p\xrightarrow[0]{\mathit{true}/v}\bullet\in F_A$, $q\xrightarrow[0]{\mathit{true}/w}\bullet\in F_B$, and $(p,q)\in Q_{A\times B}$. Finally, remove from $Q_{A\times B}$ (and $\varDelta_{A\times B}$) all deadends (non-initial states from which $\bullet$ is not reachable).

We lift the definition of transductions to product ESFTs. A pair-state $(p,q)\in Q_{A\times B}$ is *aligned* if all transitions from (p,q) have outputs (f,g) such that $f\neq\bot$ and $g\neq\bot$. The relation $\overset{/}{\twoheadrightarrow}_{A\times B}$ is defined analogously to ESFTs.

Lemma 1 (Product). *For all aligned* $(p,q)\in Q_{A\times B}$, $u\in\varSigma^*$, $v,w\in\varGamma^*$*:*

$$(p,q)\overset{u/(v,w)}{\twoheadrightarrow}_{A\times B}\bullet\quad\Leftrightarrow\quad p\overset{u/v}{\twoheadrightarrow}_A\bullet\wedge q\overset{u/w}{\twoheadrightarrow}_B\bullet.$$

We define also, for all $u\in\varSigma^*$, $\mathscr{T}_{A\times B}(u)\overset{\text{def}}{=}\{(v,w)\mid q^0_{A\times B}\overset{u/(v,w)}{\twoheadrightarrow}\bullet\}$ and $\mathscr{D}(A{\times}B)\overset{\text{def}}{=}\mathscr{D}(\mathscr{T}_{A\times B})$. Lemma 1 implies that $\mathscr{D}(A{\times}B)=\mathscr{D}(A)\cap\mathscr{D}(B)$ and $A\overset{1}{\not\equiv}B$ iff there exists u and $v\neq w$ such that $(v,w)\in\mathscr{T}_{A\times B}(u)$.

Next we prove an *alignment* lemma that allows us to either effectively eliminate all nonaligned pair-states from $A{\times}B$ without affecting $\mathscr{T}_{A\times B}$ or else establishes that $A\overset{1}{\not\equiv}B$. A product ESFT is *aligned* if all pair-states in it are aligned.

Lemma 2 (Alignment). *If $A\overset{1}{=}B$ then there exists an aligned product ESFT that is equivalent to $A{\times}B$. Moreover, there is an effective procedure that either constructs it or else proves that $A\overset{1}{\not\equiv}B$, if the label theory is decidable.*

Proof. The product $A{\times}B$ is incrementally transformed by eliminating nonaligned pair-states from it. Each iteration preserves equivalence. Using DFS, initialize the search *frontier* to be $\{q^0_{A\times B}\}$. Pick (and remove) a state (p,q) from the frontier and consider all transitions starting from it. The main two cases are the following:

1. If there are transitions from (p,q) where both the A-output f and the B-output g are $(\sigma^\ell\to\gamma)$-sequences with equal lookahead (say $\ell=2$):

$$(p,q)\xrightarrow[1]{\varphi/(f,g)}(p_1,q_1)\xrightarrow[1]{\psi/(\bot,\bot)}(p_2,q_2)$$

replace the path with the following combined transition with lookahead 2

$$(p, q) \xrightarrow[2]{\lambda(x_0,x_1).\varphi(x_0)\wedge\psi(x_1)/(f,g)} (p_2, q_2).$$

and add (p_2, q_2) to the frontier unless (p_2, q_2) has already been visited. Note that $(p_2, q_2) \in Q_A \times Q_B$ and thus (p_2, q_2) is aligned.

2. Assume there are transitions where the A-output f is a $(\sigma^k \to \gamma)$-sequence and the B-output g is a $(\sigma^\ell \to \gamma)$-sequence ($k \neq \ell$, say $k = 2$ and $\ell = 1$):

$$(p, q) \xrightarrow{\varphi/(f,g)} (p_1, q_1) \xrightarrow{\psi/(\bot,g_1)} (p_2, q_2)$$

So p_1 is temporary while q_1 is not.
Decide if f can be split into two independent $(\sigma \to \gamma)$-sequences f_1 and f_2 such that for all $a_1 \in [\![\varphi]\!]$ and $a_2 \in [\![\psi]\!]$, $[\![f]\!](a_1, a_2) = [\![f_1]\!](a_1) \cdot [\![f_2]\!](a_2)$. To do so, choose h_1 and h_2 such that $f = \lambda(x, y).h_1(x, y) \cdot h_2(x, y)$ (note that the total number of such choices is $|f| + 1$ where $|f|$ is the length of the output sequence), let $f_1 = \lambda x.h_1(x, \mathscr{W}(\psi))$, $f_2 = \lambda x.h_2(\mathscr{W}(\varphi), x)$ and check validity of the *split predicate*

$$\forall x\, y\, ((\varphi(x) \wedge \psi(y)) \Rightarrow f(x, y) = f_1(x) \cdot f_2(y))$$

If there exists a valid split predicate then pick such f_1 and f_2, and replace the above path with

$$(p, q) \xrightarrow{\varphi/(f_1,g)} (p_1', q_1') \xrightarrow{\psi/(f_2,g_1)} (p_2, q_2)$$

where (p_1', q_1') is a new *aligned* pair-state added to the frontier.
Suppose that splitting fails. We show that $A \overset{1}{\neq} B$, by way of contradiction. Assume $A \overset{1}{=} B$.
Since splitting fails, the following *dependency* predicates are satisfiable:

$$\begin{aligned} D1 &= \lambda(x, x', y).\varphi(x) \wedge \varphi(x') \wedge \psi(y) \wedge f(x, y) \neq f(x', y) \\ D2 &= \lambda(x, y, y').\varphi(x) \wedge \psi(y) \wedge \psi(y') \wedge f(x, y) \neq f(x, y') \end{aligned}$$

Let $(a_1, a_1', a_2) = \mathscr{W}(D1)$ and $(e_1, e_2, e_2') = \mathscr{W}(D2)$. Assume that $A \overset{1}{=} B$. We proceed by case analysis over $|f|$. We know that $|f| \geq 1$, or else splitting is trivial.

(a) Assume first that $|f| = 1$. Let

$$[b] = [\![f]\!](a_1, a_2),\ [b'] = [\![f]\!](a_1', a_2),\ [d] = [\![f]\!](e_1, e_2),\ [d'] = [\![f]\!](e_1, e_2').$$

Thus $b \neq b'$ and $d \neq d'$. Since (p, q) is aligned, and (p_1, q_1) is reachable and alive (by construction of $A \times B$, $\bullet$ is reachable from (p_1, q_1)), there exists $\alpha, \beta \in \Sigma^*$, $u_1, u_2, v_1, v_2, v_3, v_4 \in \Gamma^*$, such that, by *IsSat*(*D1*),

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[a_1,a_2]/[b]}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[a_1]/[\![g]\!](a_1)} q_1 \overset{[a_2]\cdot\beta/v_2}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1\cdot[b]\cdot u_2 = \\ v_1 \cdot [\![g]\!](a_1)\cdot v_2\end{array}$$

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[a_1',a_2]/[b']}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[a_1']/[\![g]\!](a_1')} q_1 \overset{[a_2]\cdot\beta/v_2}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1\cdot[b']\cdot u_2 = \\ v_1 \cdot [\![g]\!](a_1')\cdot v_2\end{array}$$

By $b \neq b'$, $|v_1| \leq |u_1| < |v_1 \cdot [\![g]\!](a_1)| = |v_1| + |g|$. Also, by *IsSat(D2)*,

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[e_1,e_2]/[d]}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[e_1]/[\![g]\!](e_1)} q_1 \overset{[e_2]\cdot\beta/v_3}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1\cdot[d]\cdot u_2 = \\ v_1 \cdot [\![g]\!](e_1)\cdot v_3\end{array}$$

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[e_1,e_2']/[d']}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[e_1]/[\![g]\!](e_1)} q_1 \overset{[e_2']\cdot\beta/v_4}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1\cdot[d']\cdot u_2 = \\ v_1 \cdot [\![g]\!](e_1)\cdot v_4\end{array}$$

By $d \neq d'$, $|v_1 \cdot [\![g]\!](e_1)| = |v_1| + |g| \leq |u_1|$. But $|u_1| < |v_1| + |g|$. ↯

(b) Assume that $f = \lambda(x,y).[f_1(x,y), f_2(x,y)]$ (the case for $|f| > 2$ is similar). Since f cannot be split, either $f_1(x,y)$ depends on y (modulo ψ) or $f_2(x,y)$ depends on x (modulo φ).

i. Suppose $f_1(x,y)$ does not depend on y. Then $f_2(x,y)$ must depend of *both* x and y or else f can be split. We can then choose values $a_1, a_1', e_1 \in [\![\varphi]\!]$ and $a_2, e_2, e_2' \in [\![\psi]\!]$ such that $[\![f_2]\!](a_1,a_2) \neq [\![f_2]\!](a_1',a_2)$ and $[\![f_2]\!](e_1,e_2) \neq [\![f_2]\!](e_1,e_2')$. A contradiction is reached similarly to the case of $|f| = 1$.

ii. The case when $f_2(x,y)$ does not depend on x is symmetrical to (i).

iii. Suppose $f_1(x,y)$ depends on y and $f_2(x,y)$ depends on x. Choose $e_1, a_1, a_1' \in [\![\varphi]\!]$ and $e_2, e_2', a_2 \in [\![\psi]\!]$ such that $[\![f_1]\!](e_1,e_2) \neq [\![f_1]\!](e_1,e_2')$ and $[\![f_2]\!](a_1,a_2) \neq [\![f_2]\!](a_1',a_2)$. Let

$$b_1 = [\![f_1]\!](e_1,e_2),\ b_1' = [\![f_1]\!](e_1,e_2'),\ b_2 = [\![f_2]\!](a_1,a_2),\ b_2' = [\![f_2]\!](a_1',a_2)$$

Since (p,q) is input-synchronized, and (p_1,q_1) is reachable and alive, there exists $\alpha, \beta \in \Sigma^*$, $u_1, u_2, v_1, v_2, v_3, v_4 \in \Gamma^*$, such that:

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[a_1,a_2]/[\_,b_2]}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[a_1]/[\![g]\!](a_1)} q_1 \overset{[a_2]\cdot\beta/v_2}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1\cdot[\_,b_2]\cdot u_2 = \\ v_1 \cdot [\![g]\!](a_1)\cdot v_2\end{array}$$

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[a_1',a_2]/[\_,b_2']}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[a_1']/[\![g]\!](a_1')} q_1 \overset{[a_2]\cdot\beta/v_2}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1\cdot[\_,b_2']\cdot u_2 = \\ v_1 \cdot [\![g]\!](a_1')\cdot v_2\end{array}$$

Since $b_2 \neq b_2'$ it must be that $|u_1|+1 < |v_1 \cdot [\![g]\!](a_1)| = |v_1|+|g|$ Also,

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[e_1,e_2]/[b_1,\_]}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[e_1]/[\![g]\!](e_1)} q_1 \overset{[e_2]\cdot\beta/v_3}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1 \cdot [b_1,\_] \cdot u_2 = \\ v_1 \cdot [\![g]\!](e_1) \cdot v_3 \end{array}$$

$$\left.\begin{array}{l} p_0 \overset{\alpha/u_1}{\twoheadrightarrow} p \overset{[e_1,e_2']/[b_1',\_]}{\twoheadrightarrow} p_2 \overset{\beta/u_2}{\twoheadrightarrow}_A \bullet \\ q_0 \overset{\alpha/v_1}{\twoheadrightarrow} q \xrightarrow{[e_1]/[\![g]\!](e_1)} q_1 \overset{[e_2']\cdot\beta/v_4}{\twoheadrightarrow}_B \bullet \end{array}\right\} \overset{(A \overset{1}{=} B)}{\Longrightarrow} \begin{array}{l} u_1 \cdot [b_1',\_] \cdot u_2 = \\ v_1 \cdot [\![g]\!](e_1) \cdot v_4 \end{array}$$

Thus, since $b_1 \neq b_1'$, we have $|v_1 \cdot [\![g]\!](e_1)| = |v_1| + |g| \leq |u_1|$. But $|u_1| < |v_1| + |g|$. ↯

The remaining cases are similar and effectively eliminate all nonaligned pair-states from $A{\times}B$ or else establish that $A \overset{1}{\neq} B$. ⊠

Assume $A{\times}B$ is aligned and let $\lceil A{\times}B \rceil$ be the following *product SFT* (product ESFT all of whose transitions have lookahead 1) over the input type σ^*. For each $p \xrightarrow[\ell]{\lambda\bar{x}.\varphi(x_0,x_1,\ldots,x_{\ell-1})/(f,g)} q$ in $\Delta_{A\times B}$ let y be a variable of sort σ^* and let φ_1 be the σ^*-predicate

$$\lambda y.\varphi(y[0], y[1], \ldots, y[\ell-1]) \wedge \mathit{tail}^{\ell}(y) = [] \bigwedge_{i<\ell} \mathit{tail}^{i}(y) \neq []$$

where $y[i]$ is the term that accesses the i'th head of y and $\mathit{tail}^i(y)$ is the term that accesses the i'th tail of y. Lift f to the $(\sigma^* \to \gamma)$-sequence $f_1 = \lambda y.f(y[0], y[1], \ldots, y[\ell-1])$ and lift g similarly to g_1. Add the rule $p \xrightarrow[1]{\varphi_1/(f_1,g_1)} q$ as a rule of $\lceil A{\times}B \rceil$. Thus, the domain type of $\mathscr{T}_{\lceil A\times B\rceil}$ is $(\Sigma^*)^*$ while the range type is $2^{\Gamma^*\times\Gamma^*}$. For $u = [u_0, u_1, \ldots, u_n] \in (\Sigma^*)^*$, let $\lfloor u \rfloor \overset{\text{def}}{=} u_0 \cdot u_1 \cdots u_n$ in Σ^*.

Lemma 3 (Grouping). *Assume $A{\times}B$ is aligned. For all $u \in \Sigma^*$ and $v, w \in \Gamma^*$: $(v, w) \in \mathscr{T}_{A\times B}(u)$ iff $\exists z(u = \lfloor z \rfloor \wedge (v, w) \in \mathscr{T}_{\lceil A\times B\rceil}(z))$.*

Proof. The type lifting does not affect the semantics of the label-theory specific transformations. ⊠

Note that, $[[a_1, a_2], [a_3]]$ and $[[a_1], [a_2, a_3]]$ may very well be distinct inputs of the lifted product, while both correspond to the same flattened input $[a_1, a_2, a_3]$ of the original product. Intuitively, the internal subsequences correspond to input alignment boundaries of the two ESFTs A and B.

So, in particular, grouping preserves the property of there existing an input u and outputs $v \neq w$ such that $(v, w) \in \mathscr{T}_{A\times B}(u)$. We use the following lemma that is extracted from the main result in [18, Proof of Theorem 1].

Lemma 4 (SFT One-Equality [18]). *Let C be a product SFT over a decidable label theory. The problem of deciding if there exist u and $v \neq w$ such that $(v, w) \in \mathscr{T}_C(u)$ is decidable.*

We can now prove the main decidability result of this paper.

Theorem 8 (Cartesian ESFT One-Equality). *One-equality of Cartesian ESFTs over decidable label theories is decidable.*

Proof. Let A and B be Cartesian ESFTs. Construct $A{\times}B$. By the Product lemma 1, $\mathscr{D}(A{\times}B) = \mathscr{D}(A) \cap \mathscr{D}(B)$ and $A \overset{1}{\not\equiv} B$ iff there exist u and $v \neq w$ such that $(v, w) \in \mathscr{T}_{A\times B}(u)$. By using the Alignment lemma 2, construct aligned product SFT C such that $\mathscr{T}_C = \mathscr{T}_{A\times B}$ or else determine that $A \overset{1}{\not\equiv} B$. Now lift C to $\lceil C\rceil$, and by using the Grouping lemma 3, $A \overset{1}{\not\equiv} B$ iff there exist u and $v \neq w$ such that $(v, w) \in \mathscr{T}_{\lceil C\rceil}(u)$. Finally, observe that adding the sequence operations for accessing the head and the tail of sequences in the lifting contruction do, by themselves, not affect decidability of the label theory, apply Lemma 4. ⊠

3.3 Composition of ESFTs

In this section we show some preliminary results (mainly negative) on ESFT composition.

Given $\mathbf{f}: X \to 2^Y$ and $\mathbf{x} \subseteq X$, $\mathbf{f}(\mathbf{x}) \overset{\text{def}}{=} \bigcup_{x\in\mathbf{x}} \mathbf{f}(x)$. Given $\mathbf{f}: X \to 2^Y$ and $\mathbf{g}: Y \to 2^Z$, $\mathbf{f} \circ \mathbf{g}(x) \overset{\text{def}}{=} \mathbf{g}(\mathbf{f}(x))$. This definition follows the convention in [5], i.e., $\circ$ applies first $\mathbf{f}$, then $\mathbf{g}$, contrary to how $\circ$ is used for standard function composition. The intuition is that $\mathbf{f}$ corresponds to the relation $R_\mathbf{f}: X \times Y$, $R_\mathbf{f} \overset{\text{def}}{=} \{(x, y) \mid y \in \mathbf{f}(x)\}$, so that $\mathbf{f} \circ \mathbf{g}$ corresponds to the binary relation composition $R_\mathbf{f} \circ R_\mathbf{g} \overset{\text{def}}{=} \{(x, z) \mid \exists y(R_\mathbf{f}(x, y) \land R_\mathbf{g}(y, z))\}$.

Definition 10. *A class of transducer C is closed under composition iff for every $\mathscr{T}_1$ and $\mathscr{T}_2$ that are C-definable $\mathscr{T}_1 \circ \mathscr{T}_2$ is also C-definable.*

We start by showing that both ESFTs and Cartesian ESFTs are not closed under composition.

Theorem 9. *ESFTs are not closed under composition.*

Proof Sketch: We show two Cartesian ESFTs whose composition cannot be expressed by any ESFT. Let A be following ESFT over $\mathbb{Z} \to \mathbb{Z}$

$$A = (\{q\}, q, \{q \xrightarrow[2]{\mathit{true}/[x_1,x_0]} q, q \xrightarrow[0]{\mathit{true}/[]} \bullet\}).$$

and B be following ESFT over $\mathbb{Z} \to \mathbb{Z}$

$$B = (\{q_0, q_1\}, q_0, \{q_0 \xrightarrow[1]{\mathit{true}/[x_0]} q_1, q_1 \xrightarrow[2]{\mathit{true}/[x_1,x_0]} q_1, q_1 \xrightarrow[1]{\mathit{true}/[x_0]} \bullet\})$$

The two transformations behave as in the following examples:

$$\mathscr{T}_A([a_0, a_1, a_2, a_3, a_4, a_5, a_6, \ldots]) = [a_1, a_0, a_3, a_2, a_5, a_4, a_7, \ldots]$$

$$\mathscr{T}_B([b_0, b_1, b_2, b_3, b_4, b_5, \ldots]) = [b_0, b_2, b_1, b_4, b_3, b_6, \ldots]$$

When we compose $\mathscr{T}_A$ and $\mathscr{T}_B$ we get the following transformation:

$$\mathscr{T}_{A\circ B}([a_0, a_1, a_2, a_3, a_4, a_5, a_6, \ldots]) = [a_1, a_3, a_0, a_5, a_2, a_7, \ldots]$$

Intuitively, looking at $\mathscr{T}_{A\circ B}$ we can see that no finite lookahead seems to suffice for this function. The following argument is illustrated by this figure:

$$\begin{tikzcd}
 & a_0 \arrow[ddr] & a_1 \arrow[ddl] & a_2 \arrow[ddr] & a_3 \arrow[ddl] & a_4 \arrow[ddr] & a_5 \arrow[ddl] & a_6 \arrow[ddr] & a_7 \arrow[ddl] & \cdots \\
A: & & & & & & & & & \\
 & a_1 \arrow[dd] & a_0 \arrow[ddr] & a_3 \arrow[ddl] & a_2 \arrow[ddr] & a_5 \arrow[ddl] & a_4 \arrow[ddr] & a_7 \arrow[ddl] & a_6 & \cdots \\
B: & & & & & & & & & \\
 & a_1 & a_3 & a_0 & a_5 & a_2 & a_7 & a_4 & \cdots &
\end{tikzcd}$$

⊠

Corollary 5. *Given two Cartesian ESFTs, it is not the case that their composition is ESFT definable.*

Corollary 6. *Cartesian ESFTs are not closed under composition.*

We now show that in general the composition of two ESFTs cannot be effectively computed.

Theorem 10 (Undecidability of Composition Computation). *Given two ESFTs with lookahead 2 over quantifier free successor arithmetic and tuples whose composition f is ESFT definable, it is undecidable to compute the ESFT corresponding to f.*

Proof Sketch: Given a Minsky machine M we construct two ESFTs A and B such that their composition C is ESFT definable and C is defined on some input. iff M halts on input $(0, 0)$ with a non-zero output in r_1 Consider the predicates defined in the proof of Theorem 1. Let

$$A = (\{p_0\}, p_0, \{p_0 \xrightarrow[2]{\varphi^{\text{step}}/[x_0,x_1]} p_0, \quad p_0 \xrightarrow[0]{\mathit{true}/\bullet} \}), \mathit{and}$$

$$B = (\{q_0, q_1\}, q_0, \{q_0 \xrightarrow[1]{\varphi^{\text{ini}}/[x_0]} q_1, \quad q_1 \xrightarrow[2]{\varphi^{\text{step}}/[x_0,x_1]} q_1, \quad q_1 \xrightarrow[1]{\varphi^{\text{fin}}/[x_0]} \bullet\}).$$

So $\alpha \in \mathscr{D}(A \circ B)$ iff α is a valid computation of M, i.e., $\alpha[0]$ is the initial configuration, $\alpha[i+1]$ is a valid successor configuration of $\alpha[i]$ (this follows from A for all odd $i < |\alpha|$ and from B for all even $i < |\alpha|$), and $\alpha[|\alpha| - 1]$ is a halting configuration. Since M is deterministic and we fix the initial configuration, we have that $\mathscr{D}(A \circ B) = \{\alpha\}$ iff there exists α, such that M halts on α or $\mathscr{D}(A \circ B) = \emptyset$ otherwise. In the first case we will have $\mathscr{T}_{A\circ B}(\alpha) = \alpha$ and undefined on any input different from α. In the second case $\mathscr{T}_{A\circ B}$ is always undefined. In both cases $\mathscr{T}_{A\circ B}$ is ESFT definable. Let's call C the ESFT that implements $\mathscr{T}_{A\circ B}$. Since emptyness of ESFT is a decidable problem, we can decide if M halts on input $(0, 0)$ with a non-zero output in r_1. The latter is an r.e.-complete problem. ⊠

4 Experiments and Applications

In this section we show how several practical applications can be modeled and verified using ESFAs and ESFTs. We first use ESFTs to prove the correctness of some real world string encoders and decoders. We then show how ESFAs and ESFTs can be useful in the context of deep packet inspection and network protocol transformations. Finally we propose ESFTs as a tool for the analysis of list manipulating programs. All our experiments are run using the tool Bek[4].

Analysis of String Encoders. A string encoder E transforms input strings in a given format A into output strings in a different format B. A decoder D inverts such transformation. The formats A and B usually use different alphabets (character sets). The first half of Table 1 shows examples of common string encoders/decoders and their respective lookahead sizes. $E \circ D$ ($D \circ E$) denotes the sequential compositions of the encoder with the decoder (decoder with the encoder). We compute such compositions using the semi-decision procedure of [3].

The correctness of Utf8 encoding was already investigated in [3] using a semi-decision procedure for 1-equality. We use the algorithm proposed in Section 3.2 to confirm such result and we prove the correctness of three new encoders: Base64, Base32 and Base16. The second half of Table 1 shows the running times of the analyses. The column $E \circ D \overset{1}{=} I$ ($D \circ E \overset{1}{=} I$) shows the cost of checking whether $E \circ D$ ($D \circ E$) is 1-equivalent to the identity transducer I. Composition times (typically 1-2 ms) are included in the measurements.

	Lookahead		Analysis (ms)	
	E	D	$E\circ D\overset{1}{=}I$	$D\circ E\overset{1}{=}I$
Utf8:	2	4	16	24
Base64:	3	5	53	19
Base32:	5	8	8	12
Base16:	1	2	2	1

Table 1. Analyzed encoders (E) and decoders (D), their lookaheads, and analysis times.

We want to stress that during our experiments we identified wrong implementations of the Utf8 encoder/decoder in which the algorithm of Section 3.2 correctly detected that one-equality fails, while the semi-decision procedure used in [3] did not terminate.

Deep Packet Inspection. Fast identification of network traffic patterns is of vital importance in network routing, firewall filtering and intrusion detection. This task is addressed with the name "deep packet inspection" (DPI) [17]. Due to performance constraints, DPI must be performed in a single pass over the input. The simplest approach is to use DFAs and NFAs to identify patterns. These representations are either not succinct or not streamable. Extended Finite Automata (XFA) [17] make use of registers to reduce the state space while preserving determinism and therefore deterministic ESFAs can be seen as a subclass of XFAs that are able to deal with finite lookahead. Deterministic ESFA can also represent the alphabet symbolically, which enables a new level of succinctness. We believe that deterministic ESFAs can help achieve further succinctness in particular problem instances. To support this hypothesis we observe that

[4] http://www.rise4fun.com/Bek.

several examples shown in [17, Figure 2,3] can be represented as deterministic ESFAs with few transitions. For example the language `^/\ncmd[^\n]{200}$` can be succinctly captured by a deterministic ESFA with one transition!

Network Protocol Conversions. Deep packet inspection can be naturally extended by adding data manipulation. As in the previous setting we are interested in deterministic ESFTs which can commit their output at every transition without seeing the rest of the input. Deterministic ESFTs can be used to compute logs of network traffic or translate headers of one protocol into another. As an example, a simple translation from an IPv4 header to an IPv6 header[5] can be easily implemented with a deterministic ESFT with less than 50 transitions. The same transformation using an SFT would require over 100000 transitions.

Verification of List Manipulating Programs. ESFTs can be used for verification of *list manipulating programs* as they naturally model sequential pattern matching. The ML guards `x1::x2::xs -> (x1+x2)::(f2 xs)` and `x1::x2::x3::xs -> (x1+x2+x3)::(f3 xs)`, respectively belonging to the functions $f_2, f_3 : \mathit{list}\ \mathit{int} \to \mathit{list}\ \mathit{int}$, can be naturally expressed as ESFT transitions. Therefore f_2 and f_3 can be modelled as ESFTs. We can then use the 1-equivalence algorithm of Section 3.2 to prove that $f_2(f_2(f_2\ l)) \overset{1}{=} f_3(f_3\ l)$ in less than 1 ms.

Modeling Tuple Alphabets. ESFTs also provide a natural way to extend SFTs to work with tuple alphabets without changing the underlying solver. In particular, the language BEK [8] is optimized for 16 bits characters. An ESFT of lookahead two, for example, will be able to model 32 bits characters without having to change the underlying solver.

5 Related Work

Symbolic finite transducers (SFTs) and Bek were originally introduced in [8] with a focus on security analysis of sanitizers. The formal foundations and the theoretical analysis of the underlying SFT algorithms, in particular, an algorithm for one-equality of SFTs, modulo a decidable background theory is studied in [18]. Symbolic Transducers (STs) that allow the use of registers are also defined in [18]. Full equivalence of finite state transducers is undecidable [7], and already so for very restricted fragments [9]. In the single-valued case, decidability was established in [15], and extended to the finite-valued case in [2, 19].

ESFTs were introduced in [3] as a succinct and more analyzable representation of a subclass of symbolic transducers (STs). The main result in [3] is a register elimination technique that provides a way to construct (product) ESFTs from (product) symbolic transducers (STs). While this technique provides a semi-decision procedure for one-equality checking (by using grouping, Lemma 3)

[5] More information at http://www.cs.washington.edu/research/networking/napt/

of non-Cartesian ESFTs, it does not provide a full decision procedure for one-equality of the Cartesian case. The procedure in [3] fails to decide *alignment* of ESFTs, that is the key lemma (Lemma 2) used in the main decidability result of Theorem 8, that is a proper extension of the decidability result of one-equality of SFTs [18, Theorem 1]. We also show that one-equality is undecidable in the non-Cartesian case, Theorem 7, that is in sharp contrast to the theory of classical automata, where the non-Cartesian case is irrelevant (from the point of view of decidability) due to the standard form [20, Theorem 2.17].

Extended Top-Down Tree Transducers [12] (ETTTs) are commonly used in natural language processing. ETTS also allow finite lookahead on transformation from trees to trees, but only support finite alphabets. The special case in which the input is a string (unary tree) is equivalent to ESFTs over finite alphabets. This paper focuses on ESFTs over any decidable theory. We leave as future work extending the model to tree transformations.

In recent years there has been considerable interest in automata using infinite alphabets [16], starting with the work on register automata [10]. Finite words over an infinite alphabet are often called data words. This line of work focuses on fundamental questions about decidability, complexity, and expressiveness on classes of automata on one hand and fragments of logic on the other hand.

Streaming transducers [1] provide another recent symbolic extension of finite transducers where the label theories are restricted to be total orders, in order to maintain decidability of equivalence. Streaming transducers are largely orthogonal to SFTs or the extension of ESFTs, as presented in the current paper. For example, streaming transducers do not allow arithmetic, but can reverse the input, which is not possible with ESFTs.

The correctness of UTF8 encoder and decoder was proven in [3] using two semi-decision procedures for equivalence and composition. In this paper we show that the composition of UTF8 encoder and decoder can be expressed as a Cartesian ESFT and can be formally analyzed with the one-equivalence algorithm introduced in this paper. We do the same for three encoders of which the correctness was not proven before: BASE64, BASE32, BASE16.

Extended Finite Automata (XFA) are introduced in [17] for network packet inspection. XFAs are a succinct representation of DFAs that use registers and allow programs over the registers. ESFAs are orthogonal to XFAs in two ways: 1) XFAs only support finite alphabets; and 2) XFAs aim at representing *most* DFAs succinctly, while ESFAs only capture the languages that use finite lookahead. We have not investigated the application of ESFAs to network packet inspection in detail, but we think that they can help achieving a further level of succinctness. History-based finite automata [11] are another extension of DFAs that have been introduced for encoding regular expressions in the context of network intrusion detection systems, they use a single register (bitvector) to keep track of history. The register is used together with the input character to determine enabledness of transitions.

We use the SMT solver Z3 [4] for incrementally solving label constraints that arise during the exploration algorithm. Similar applications of SMT techniques have been introduced in the context of symbolic execution of programs by using path conditions to represent under and over approximations of reachable states [6].

Our work is complementary to previous efforts in using SMT solvers to solve problems related to list transformations. Kaluza [14] extends the SMT solver to handle equations with multiple variables over strings.

6 Conclusion

We showed fundamental negative and positive results about several classical decision problems of ESFAs and ESFTs, establishing a sharp boundary between decidability (the Cartesian case with any decidable background) and undecidability (the non-Cartesian case with a background of successor arithmetic). While the main motivation came from typical *static analysis* problems using ESFTs, an equally important application of the Cartesian case is for efficient *code generation*. Namely, the conjucts of a Cartesian predicate can be compiled and normalized into separate unary predicates that may for example use BDDs for efficient and unique set representation when dealing with bitvectors, as in the context of strings coders. Identifying classes of ESFTs that are closed under composition, as well as extending ESFTs to trees are left as open problems.

References

1. R. Alur and P. Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL'11*, pages 599–610. ACM, 2011.
2. K. Culic and J. Karhumäki. The equivalence of finite-valued transducers (on HDTOL languages) is decidable. *Theoretical Computer Science*, 47:71–84, 1986.
3. L. D'Antoni and M. Veanes. Static analysis of string encoders and decoders. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI 2013*, volume 7737 of *LNCS*, pages 209–228. Springer, 2013.
4. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS'08*, LNCS. Springer, 2008.
5. Z. Fülöp and H. Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. EATCS. Springer, 1998.
6. P. Godefroid. Compositional dynamic test generation. In *POPL'07*, pages 47–54, 2007.
7. T. Griffiths. The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines. *J. ACM*, 15:409–413, 1968.
8. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Fast and precise sanitizer analysis with Bek. In *USENIX*, August 2011.
9. O. Ibarra. The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. *SIAM Journal on Computing*, 4:524–532, 1978.
10. M. Kaminski and N. Francez. Finite-memory automata. *TCS*, 134(2):329–363, 1994.

11. S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. In *ANCS 2007*, pages 155–164. ACM/IEEE, 2007.
12. A. Maletti, J. Graehl, M. Hopkins, and K. Knight. The power of extended top-down tree transducers. *SIAM J. Comput.*, 39(2):410–430, June 2009.
13. M. Mohri. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311, June 1997.
14. P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A symbolic execution framework for javascript. SP '10, pages 513–528, Washington, DC, USA, 2010. IEEE Computer Society.
15. M. P. Schützenberger. Sur les relations rationnelles. In *GI Conference on Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 209–213, 1975.
16. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, pages 41–57, 2006.
17. R. Smith, C. Estan, S. Jha, and S. Kong. Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. SIGCOMM '08, pages 207–218. ACM, 2008.
18. M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner. Symbolic finite state transducers: Algorithms and applications. In *POPL'12*, pages 137–150, 2012.
19. A. Weber. Decomposing finite-valued transducers and deciding their equivalence. *SIAM Journal on Computing*, 22(1):175–202, February 1993.
20. S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1997.