Form Methods Syst Des (2015) 47:93–119
DOI 10.1007/s10703-015-0233-4

Extended symbolic finite automata and transducers

Loris D'Antoni[1] · **Margus Veanes**[2]

Published online: 7 July 2015

Abstract Symbolic finite automata and transducers augment classic automata and transducers with symbolic alphabets represented as parametric theories. This extension enables to succinctly represent large and potentially infinite alphabets while preserving closure and decidability properties. Extended symbolic finite automata and transducers further extend these objects by allowing transitions to read consecutive input elements in a single step. In this paper we study the properties of these models. In contrast to the case of finite alphabets, we show how reading multiple symbols increases the expressiveness of the models, which causes some closure properties to stop holding and most decision problems to become undecidable. In particular we show how extended symbolic finite transducers are not closed under composition, and the equivalence problem is undecidable for both extended symbolic finite automata and transducers. We then introduce the subclass of Cartesian extended symbolic finite transducers in which guards are limited to conjunctions of unary predicates and we propose an equivalence algorithm for this subclass in the single-valued case. We also present a heuristic algorithm for composing extended symbolic finite transducers that works for many practical cases. Finally, we model real world programs with Cartesian extended symbolic finite transducers and use the proposed algorithms to prove their correctness.

Keywords Symbolic automata · Symbolic transducers · Equivalence · String encoders

✉ Loris D'Antoni
lorisdan@seas.upenn.edu

Margus Veanes
margus@microsoft.com

[1] University of Pennsylvania, 3330 Walnut St., Philadelphia, PA 19104, USA

[2] Microsoft Research, One Microsoft Way, Redmond, WA, USA

1 Introduction

Finite automata have proven to be an effective tool in a wide range of applications, including regular expression matching and network packet inspection [20]. Finite transducers extend finite automata with outputs and can model functions from strings to strings such as natural language transformations [16]. Due to their closure and decidability properties, these models are widely used in practice, however their classic formulations suffer from the following limitations:

1. the number of transitions "blows up" when dealing with large alphabets and the alphabet can't be infinite; and
2. transitions cannot express relations between symbols appearing at different positions in the input.

Both these limitations arise when trying to reason about programs such as string coders. These programs transform an input string in a given format into output strings in a different format. String coders typically operate over very large alphabets (2^{16} elements), and allow a single output symbol to depend on several input symbols appearing at different positions.

To solve the first problem symbolic finite automata/transducers [22] or SFAs/SFTs extend traditional automata and transducers by allowing transitions to be labelled with arbitrary predicates in a specified theory (e.g. Presburger arithmetic over the integers). When such a 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 [22], symbolic transducers or STs (SFTs with registers), are proposed in order to cope with the second problem above. STs can store values in registers and later compare such values to other symbols, and are therefore able to express relations between symbols appearing at different positions in the input. Unfortunately, this ability makes STs undecidable with respect to most analysis problems, even emptiness.

Extended symbolic finite automata/transducers or ESFAs/ESFTs extend SFAs/SFTs with the ability to read multiple *adjacent* input symbols in a single transition and combine their values in the output [5,6]. Since these models are more expressive than SFAs/SFTs, but less expressive than STs with registers, they are potential candidates for addressing the two problems we presented above while retaining decidability. In this paper we study their formal properties and show that unfortunately this increase expressiveness comes at a price. From the point of view of analysis, the key operations that are desired are *closure under Boolean operations* for ESFAs, and *composition* and *equivalence* for ESFTs.

We first show that ESFAs are not closed under intersection and that equivalence and universality of ESFAs are both undecidable problems. The emptiness problem remains decidable. In the case of ESFTs the situation is not brighter: ESFTs are not closed under composition and the equivalence problem is undecidable.

Given these negative results we focus our attention on the sub-class of Cartesian ESFTs in which the guards of the transitions are constrained to be conjunctions of unary predicates. The main result of this paper is an algorithm for checking equivalence of single-valued Cartesian ESFTs. This is a proper extension of the decidability result of equivalence of SFTs [22]. Given two ESFTs the algorithm works as follows: (1) It first expands each transition reading k symbols into k separate transitions taking advantage of the fact that guards are conjunctions of unary predicates; (2) it then computes the product of the two "expanded" ESFTs (this can be done since all guards are unary); and finally (3) it tries to align transitions so that the outputs of the two ESFTs are synchronized, and if this is not possible the two ESFTs are not equivalent.

We also propose a heuristic algorithm for composing ESFTs. The algorithm works by first converting ESFTs to STs, then composing the STs, and finally converting (when possible) the result back into an ESFT using an algorithm for eliminating registers.

We present four applications of our models in different areas. In our main experiment we use the proposed algorithms to prove the correctness of four real world string encoders which can be modeled using ESFTs. We then sketch how ESFAs and ESFTs can be used in the tasks of deep-packet inspection, transformation protocol headers, and can also be used to analyze list manipulating programs that use deep pattern matching.

Contributions In summary, we offer the following contributions:

1. A study of the closure and decidability properties of ESFAs (Sect. 3);
2. A study of the equivalence problem for ESFTs (Sect. 4) where we show that:
   - the equivalence of single-valued ESFTs is undecidable;
   - the equivalence of single-valued Cartesian ESFTs is decidable;
3. A study of the composition problem for ESFTs (Sect. 5) that includes:
   - a proof that ESFTs are not closed under composition;
   - a heuristic algorithm for composing ESFTs;
4. A concrete application of the presented algorithms to the task of proving the correctness of string encoders and decoders (Sect. 6).

We finally summarize previous work and conclude (Sects. 7 and 8).

2 Extended symbolic finite automata and transducers

2.1 Background

We assume a recursively enumerable (r.e.) *background universe* $\mathscr{U}$ with built-in function and relation symbols. Definitions below are given with $\mathscr{U}$ as an implicit parameter. 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. Our notational conventions are consistent with the definition of symbolic transducers [22]. The universe is multi-typed with $\mathscr{U}^{\tau}$ denoting the sub-universe of elements of type τ. We write Σ for $\mathscr{U}^{\sigma}$ and Γ for $\mathscr{U}^{\gamma}$.

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 $\mathscr{W}(\varphi) \in [\![\varphi]\!]$, where $[\![\varphi]\!] \subseteq \mathscr{U}^{\sigma}$ is the set of all values that satisfy φ; φ is *valid*, $\mathit{IsValid}(\varphi)$, when $[\![\varphi]\!] = \mathscr{U}^{\sigma}$.

2.2 ESFAs and ESFTs

We are studying in this paper an extension of SFTs with where transitions are allowed to consume more than one symbol, called *extended* SFTs or *ESFTs*. Originally, ESFTs were introduced in [6] 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 where all the 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 classic 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 $\mathbb{Z}$.

In the remainder of the section 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:

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

Intuitively, a rule with lookahead ℓ reads ℓ adjacent input symbols $s = [a_0, \ldots, a_{\ell-1}]$ and produces a sequence of output symbols f that is a function of the consumed input symbols.

Let $[\![R]\!] \stackrel{\text{def}}{=} \bigcup_{r \in R} [\![r]\!]$. We write $s_1 \times s_2$ for the concatenation of sequences s_1 and s_2.

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

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

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

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

The following example illustrates 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 the standard encoding BASE64, that is used to transfer binary data in textual format, e.g., in emails via the protocol 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* is an ESFT with one state and four rules:

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

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

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

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

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

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 in ASCII) is used as a padding character and it 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:

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

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

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

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, and sequences that do not correspond to any encoding are rejected.[1]

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 $\lambda(x_1, \ldots, x_{\max(m,n)}).\varphi(x_1, \ldots, x_m) \wedge \psi(x_1, \ldots, x_n)$. We define *equivalence of f and g modulo φ*, $f \equiv_\varphi g$, as: $\mathit{IsValid}(\lambda \bar{x}.(\varphi(\bar{x}) \Rightarrow f(\bar{x}) = g(\bar{x})))$.

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

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 $IsSat(\varphi \curlywedge \varphi')$ then $q = q'$, $\ell = \ell'$ and $f \equiv_{\varphi \curlywedge \varphi'} f'$.
(b) Assume $q = q' = \bullet$. If $IsSat(\varphi \curlywedge \varphi')$ and $\ell = \ell'$ then $f \equiv_{\varphi \curlywedge \varphi'} f'$.
(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, $Base64encode(\texttt{Foo+}) = \texttt{Rm9v+}$ and $Base64decode(\texttt{QmFy+}) = \texttt{Bar+}$.

2.3 Cartesian ESFAs and ESFTs

We introduce a subclass of ESFTs that plays an important role in this paper. 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* is 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 $IsCartesian(\varphi)$ is decidable by using the decision procedure of the label theory. Namely, decide unsatisfiability of $\neg IsCartesian(\varphi)$.

Cartesian ESFAs capture exactly the class of SFA definable languages.

Theorem 1 (Cartesian ESFA = SFA) *Cartesian ESFAs and SFAs are equivalent in expressiveness.*

Proof The $\Leftarrow$ direction is immediate. We prove the $\Rightarrow$ direction. Given a Cartesian ESFA $A = (Q, q_0, (\Delta, F))$ we construct and equivalent SFA A'. Without loss of generality we assume that every rule r in Δ has lookahead 2 and every finalizer has lookahead 0. For every rule $r = p \xrightarrow[2]{\varphi(x_1) \wedge \psi(x_2)} q$, the SFA A' has a three states q, p, q_r, and two rules $p \xrightarrow{\varphi(x_1)} r$, $r \xrightarrow{\psi(x_2)} q$. Finally, if A has a finalizer $q \xrightarrow[0]{true} \bullet$, the state q will be final in A'. □

As a consequence Cartesian ESFAs enjoy all the properties of SFAs (regular languages) such as boolean closures and decidability of equivalence.

2.4 Monadic ESFAs and ESFTs

We say that a (quantifier free) formula is in *monadic normal form* or *MNF* if it is a Boolean combination of unary formulas, where a *unary formula* is a formula with (at most) one free variable. A formula is *monadic* if it has an equivalent MNF. A natural problem that arises is, deciding whether a formula is monadic, and if so, constructing its MNF. For example, the formula $x < y$ over integers does not have an MNF while the formula $x < y \bmod 2$ has an MNF $(x < 0 \wedge y \bmod 2 = 0) \vee (x < 1 \wedge y \bmod 2 = 1)$ that is also a DNF. Another MNF of $x < y \bmod 2$ is $x < 0 \vee (x < 1 \wedge y \bmod 2 = 1)$ that is also a DNF but with semantically overlapping disjuncts.

Definition 7 An ESFT (ESFA) is *monadic* if all its guards are monadic.

It is shown in [21] that if the label theory is decidable and a formula is monadic then its MNF can be constructed effectively.

Theorem 2 *Monadic ESFTs and Cartesian ESFTs are effectively equivalent. Moreover, this holds also for the deterministic case.*

Proof The $\Leftarrow$ direction is immediate because Cartesian ESFTs are a special case of monadic ESFTs. For the direction $\Rightarrow$, we first apply the procedure *mondec* from [21] to each guard ϕ of the monadic ESFT to obtain an equivalent MNF of the guard, that we then rewrite into an equivalent DNF $\bigvee_{i<n} \phi_i$. Finally, we can replace each rule $p \xrightarrow[\ell]{\phi/f} q$ by the rules $p \xrightarrow[\ell]{\phi_i/f} q$, for $i < n$, where all ϕ_i are Cartesian. Determinism is clearly preserved, because all the new rules have identical outputs so the conditions (a) and (b) of Definition 5 are trivially fulfilled. □

3 Properties of extended symbolic finite automata

In this section we prove some basic properties of Extended Symbolic Finite Automata and show how they drastically differ from SFAs and have properties similar to those of context free grammars rather than regular languages.

First, we show how checking the emptiness of the intersection of two ESFA definable languages is an undecidable problem.

Theorem 3 (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 undecidable.*

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 (*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 $\lambda(x, x')$. $\bigvee_{i<|P|} \varphi_i^{\text{step}}$ where φ_i^{step} is the formula for the i'th instruction. If the i'th instruction is INC_1 then φ_i^{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

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

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

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

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

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 undecidable problem as an instance of Rice's theorem. □

Theorem 4 (Emptiness) *Given an ESFA A it is decidable to determine whether it accepts any input.*

Proof Given A, we first 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. □

Theorem 5 (Boolean properties) *ESFAs are closed under union but not closed under intersection and complement.*

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:

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

As a consequence of Theorems 3 and 4 we have that ESFA are not closed under intersection. Therefore, ESFAs cannot be closed under complement. □

While checking the emptiness of an ESFA is a decidable problem, it is not possible to decide whether an ESFA accepts every possible input. It follows that equivalence is also undecidable.

Theorem 6 (Universality and Equivalence) *Given an ESFA A over σ it is undecidable to check whether A accepts all the sequences in σ^*, and given two ESFAs A and B it is undecidable to check whether A and B accept the same languages.*

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 3.

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 undecidable problem as an instance of Rice's theorem.

Let

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

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

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

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

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 5. 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). The undecidability of equivalence follows. □

We finally show that longer a look-ahead adds expressiveness.

Theorem 7 *For every k there exists an ESFA with lookahead $k+1$ that cannot be represented by any ESFAs with lookahead k.*

Proof Consider the ESFA A over the theory of integers with one initial state q_0 and one final state q_1. The ESFA A has only one transitions between q_0 and q_1 of lookahead $k + 1$ with the following predicate $\psi(x_1, \cdots, x_{k+1}) = x_1 = x_2 = \cdots = x_{k+1}$. There doesn't exists an ESFA B with lookahead k equivalent to A. Let's assume B exists by way of contradiction. Since A only accepts strings of length $k + 1$, B can only have finitely many paths from its initial state to any final state. Let's assume w.l.o.g. that every path has length 2 and it has guards of the form $\varphi_1(x_1 \ldots x_l)$ and $\varphi_2(x_{l+1} \ldots x_{k+1})$. We now must have that $\psi(x_1, \ldots, x_{k+1}) \equiv \bigvee \varphi_1(x_1 \ldots x_l) \wedge \varphi_2(x_{l+1} \ldots x_{k+1})$. However the predicate ψ does not admit such a representation. □

4 Equivalence of extended symbolic finite transducers

While the general equivalence problem of $\mathscr{T}_A = \mathscr{T}_B$ is already undecidable for very restricted classes of finite state transducers [10], 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). In this section we first show that equivalence of ESFTs is in general undecidable, but it's decidable for Cartesian ESFTs.

Definition 8 Functions $\mathbf{f}, \mathbf{g}: X \to 2^Y$ are *one-equal*, $\mathbf{f} \overset{1}{=} \mathbf{g}$, if for all $x \in X$, 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 9 Functions $\mathbf{f}, \mathbf{g}: X \to 2^Y$ are *domain-equivalent* if $\mathscr{D}(\mathbf{f}) = \mathscr{D}(\mathbf{g})$.

Definitions 8 and 9 are lifted to (E)SFTs. For domain-equivalent single-valued transducers A and B, $A \overset{1}{=} B$ implies equivalence of A and B ($\mathscr{T}_A = \mathscr{T}_B$).

4.1 Equivalence of ESFTs is undecidable

We now show that one-equality of ESFTs over decidable label theories is undecidable in general.

Theorem 8 (One-equality) *One-equality of ESFTs with lookahead 2, over quantifier free successor arithmetic and tuples is undecidable.*

Proof We give a reduction from the domain intersection problem of Theorem 3. 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:

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

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 3. □

4.1.1 Equivalence symbolic finite transducers with look-back

In this section we briefly describe a model that is tightly related to ESFTs and for which equivalence is also undecidable. Symbolic finite transducers with look-back k (k-SLTs) [2] have a sliding window of size k that allows, in addition to the current input character, references of up to $k-1$ previous characters (using predicates of arity k). All the states of an SLTs are final and are associated with a constant output. In [2] it is wrongly claimed that equivalence of SLTs is decidable. We can prove using the same technique shown in the proof of Theorem 8 that one-equality is also undecidable for SLTs. We do not formally define SLTs here, but we briefly explain why the proof of Theorem 8 extends to this model. We let $\sigma = \mathbb{N}^3$ be the input sort and represent configurations of a Minsky machine in the same way we discussed in the proof of Theorem 3. Unlike ESFTs, SLTs do not consume k symbols at a time and can therefore read the same input character multiple times using look-back. We can construct an SLT A that when reading each character in the input uses the predicate φ^{step} defined in Theorem 8 to check whether the *current* configuration of the Minsky machine follows from the previous one. Finally, the state following

the accepting configuration outputs the constant a (every other state outputs ε). The SLT A outputs ε on every run that is not an accepting one for the Minsky machine and a on the accepting run (if it exists). We can now construct a simple SLT B with look-back 1 with one transition defined on the predicate $true$ that always outputs ε. The two SLTs A and B are one-equal iff the Minsky machine does not halt (i.e. the first SLT never outputs a).

4.2 Equivalence of Cartesian ESFTs is decidable

This is the main decidability result of the paper and it extends the corresponding result for SFTs [22, 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,

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

where p_i for $1 \leq i < \ell$ are new *temporary* states, and 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 add the new finalizer $p_1 \xrightarrow[0]{true/[]} \bullet$.

Definition 10 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

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

Let $F_{A\times B}$ be the set of all rules $(p,q) \xrightarrow[0]{true/(v,w)} \bullet$ such that $p \xrightarrow[0]{true/v} \bullet \in F_A, q \xrightarrow[0]{true/w} \bullet \in F_B$, and $(p,q) \in Q_{A\times B}$. Finally, remove from $Q_{A\times B}$ (and $\Delta_{A\times B}$) all dead ends (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 $\xrightarrow{/}_{A\times B}$ is defined analogously to ESFTs.

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

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

We define also, for all $u \in \Sigma^*$, $\mathscr{T}_{A\times B}(u) \overset{\text{def}}{=} \{(v,w) \mid q^0_{A\times B} \xrightarrow{u/(v,w)} \bullet\}$ 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 $A \overset{1}{\neq} B$ 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 non-aligned pair-states from $A \times B$ without affecting $\mathscr{T}_{A\times B}$ or else to establish that $A \overset{1}{\not=} B$. 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 $A \overset{1}{\not=} B$, if the label theory is decidable.*

Proof The product $A \times B$ is incrementally transformed by eliminating non aligned pair-states from it. Each iteration preserves equivalence. Using depth-first search, 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

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

   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 $f = \lambda(x, y) \cdot h_1(x, y) \cdot h_2(x, y)$ (note that the total number of such choices is $|f| + 1$ where $|f|$ is the length of the output sequence), let $f_1 = \lambda x \cdot h_1(x, \mathscr{W}(\psi))$, $f_2 = \lambda x \cdot h_2(\mathscr{W}(\varphi), x)$ 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 $A \overset{1}{\not=} B$, by way of contradiction. Assume $A \overset{1}{=} B$.
   Since splitting fails, the following *dependency* predicates are satisfiable:

$$D1 = \lambda(x, x', y) \cdot \varphi(x) \wedge \varphi(x') \wedge \psi(y) \wedge f(x, y) \neq f(x', y)$$
$$D2 = \lambda(x, y, y') \cdot \varphi(x) \wedge \psi(y) \wedge \psi(y') \wedge f(x, y) \neq f(x, y')$$

   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*),

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

$$\left.\begin{array}{l} p_0 \xrightarrow{\alpha/u_1} p \xrightarrow{[a'_1,a_2]/[b']} p_2 \xrightarrow{\beta/u_2}_A \bullet \\ q_0 \xrightarrow{\alpha/v_1} q \xrightarrow{[a'_1]/[\![g]\!](a'_1)} q_1 \xrightarrow{[a_2]\cdot\beta/v_2}_B \bullet \end{array}\right\} \begin{array}{l} (A \overset{1}{=} B)\ u_1 \cdot [b'] \cdot u_2 \\ \qquad\quad = v_1 \cdot [\![g]\!](a'_1) \cdot v_2 \end{array}$$

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

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

$$\left.\begin{array}{l} p_0 \xrightarrow{\alpha/u_1} p \xrightarrow{[e_1,e'_2]/[d']} p_2 \xrightarrow{\beta/u_2}_A \bullet \\ q_0 \xrightarrow{\alpha/v_1} q \xrightarrow{[e_1]/[\![g]\!](e_1)} q_1 \xrightarrow{[e'_2]\cdot\beta/v_4}_B \bullet \end{array}\right\} \begin{array}{l} (A \overset{1}{=} B)\ u_1 \cdot [d'] \cdot u_2 \\ \qquad\quad = v_1 \cdot [\![g]\!](e_1) \cdot v_4 \end{array}$$

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:

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

$$\left.\begin{array}{l} p_0 \xrightarrow{\alpha/u_1} p \xrightarrow{[a'_1,a_2]/[\_,b'_2]} p_2 \xrightarrow{\beta/u_2}_A \bullet \\ q_0 \xrightarrow{\alpha/v_1} q \xrightarrow{[a'_1]/[\![g]\!](a'_1)} q_1 \xrightarrow{[a_2]\cdot\beta/v_2}_B \bullet \end{array}\right\} \begin{array}{l} (A \overset{1}{=} B)\ u_1 \cdot [\_, b'_2] \cdot u_2 \\ \qquad\quad = v_1 \cdot [\![g]\!](a'_1) \cdot v_2 \end{array}$$

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

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

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

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 non-aligned pair-states from $A \times B$ or else establish that $A \overset{\flat}{\neq} B$. □

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 $p \xrightarrow[\ell]{\lambda \bar{x} \cdot \varphi(x_0, x_1, \ldots, x_{\ell-1})/(f,g)} q$ in $\Delta_{A \times B}$ let y be a variable of sort σ^* and let φ_1 be the σ^*-predicate

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

where $y[i]$ is the term that accesses the i'th head of y and $tail^i(y)$ 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 $\lfloor u \rfloor \overset{\text{def}}{=} u_0 \cdot u_1 \ldots u_n$ 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 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: there exists 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 [22, Proof of Theorem 1].

Lemma 4 (SFT one-equality [22]) *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 9 (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 $A \overset{\downarrow}{\neq} B$ 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 $A \overset{\downarrow}{\neq} B$. Now lift C to $\lceil C \rceil$, and by using the Grouping Lemma 3, $A \overset{\downarrow}{\neq} B$ 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 construction do, by themselves, not affect decidability of the label theory, apply Lemma 4. □

Since a Monadic ESFT can be effectively transformed into and equivalent Cartesian ESFT (Theorem 9) we get the following result.

Corollary 1 (Monadic ESFT one-equality) *One-equality of monadic ESFTs over decidable label theories is decidable.*

5 Composition of extended symbolic finite transducers

In this section we show few preliminary results on the problem of composing ESFTs. We first show that ESFTs and Cartesian ESFTs are not closed under composition. Moreover, even if the composition of two ESFTs is definable by another ESFT, it is undecidable to compute such an ESFT. Last, we give a semi-decision procedures for composing ESFTs.

5.1 ESFTs are not closed under 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 [9], 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 $R_{\mathbf{f}} \circ R_{\mathbf{g}} \overset{\text{def}}{=} \{(x, z) \mid \exists y (R_{\mathbf{f}}(x, y) \wedge R_{\mathbf{g}}(y, z))\}$.

Definition 11 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.

Theorem 10 *ESFTs are not closed under composition.*

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

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

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

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

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:

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

Intuitively, looking at $\mathscr{T}_{A\circ B}$ we can see that no finite lookahead seems to suffice for this function. Formally, for each a_i such that $i \geq 0$, $\mathscr{T}_{A\circ B}$ is the following function:

- if $i = 1$, a_i is output at position 0;
- if i is even and greater than 1, a_i is output at position $i - 2$;
- if i is equal to $k - 2$ where k is the length of the input, a_i is output at position $k - 1$;
- if i is odd and different from $k - 2$, a_i is output at position $i + 2$.

It is easy to see that the above transformation cannot be computed by any ESFT. Let's assume by contradiction that there exists an ESFT that computes $\mathscr{T}_{A\circ B}$. We consider the ESFT C with minimal lookahead (let's say n) that computes $\mathscr{T}_{A\circ B}$.

We now show that on an input of length greater than $n + 2$, C will misbehave. The first transition of C that will apply to the input will have a lookahead of size $l \leq n$. We now have three possibilities (the case $n = k - 2$ does not apply due to the length of the input):

$l = 1$: before outputting a_0 (at position 2) we need to output a_1 and a_3 which we have not read yet. Contradiction;
l is odd: position $l + 1$ is receiving a_{l-1} therefore C must output also the elements at position l. Position l should receive a_{l+2} which is not reachable with a lookahead of just l. Contradiction;
l is even and greater than 1: since $l > 1$, position l is receiving a_{l-2}. This means C is also outputting position $l - 1$. Position $l - 1$ should receive a_{l+1} which is not reachable with a lookahead of just l. Contradiction;

We now have that n cannot be the minimal lookahead which contradicts our initial hypothesis. Therefore $\mathscr{T}_{A\circ B}$ is not ESFT-definable. □

Corollary 2 *The composition of two Cartesian ESFTs is not always ESFT definable.*

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

Theorem 11 (Composition is not constructible) *Given two ESFTs with lookahead 2 over quantifier free successor arithmetic and tuples, A and B, such that composition $f = \mathscr{T}_{A\circ B}$ is ESFT definable, one cannot effectively construct an ESFT that defines the transformation f.*

Proof Given a Minsky machine M we construct two ESFTs A and B such that their composition $A \circ B$ is definable by an ESFT C such that:

- if M halts on input $(0, 0)$ with a non-zero output in r_1, C is defined exactly on the run of M, and
- otherwise C is the empty transducer that is undefined on any input.

The proof is analogous to that of Theorem 3 and we use the composition of ESFTs to simulate the intersection of two ESFAs. Consider the predicates defined in the proof of Theorem 3. Let

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

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

We have that $\alpha \in \mathscr{D}(A \circ B)$ iff α is a valid run 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 emptiness of ESFT is a decidable problem, we can decide if M halts on input $(0, 0)$ with a non-zero output in r_1. Since, the latter is an undecidable problem we have a contradiction and therefore the composition of two ESFTs cannot be computed. □

5.1.1 Composition of symbolic finite transducers with look-back

In this section we discuss how SLTs, the model we presented in Sect. 4.1.1, compares to ESFTs for what concern composition. We recall that symbolic finite transducers with look-back k (k-SLTs) [2] have a sliding window of size k that allows, in addition to the current input character, references of up to $k-1$ previous characters (using predicates of arity k). All the states of an SLTs are final and are associated with a constant output. In [2] it is wrongly claimed that SLTs are closed under composition. We briefly explain why this is the case using two SLTs over the sort $\sigma = \mathbb{N}$. Consider an SLT A that echoes the first element of the input, then deletes all the subsequent elements that are smaller or equal than 5, and finally outputs the first element that is greater than 5. For example on the input sequence $[1, 2, 4, 2, 5, 6]$, the SLT A outputs the sequence $[1, 6]$. We observe that on any input sequence of the form $a_1 \ldots a_n$ such that for every $1 < i \leq n$, $a_i \leq 5$, and $a_n > 5$, the SLT A outputs the sequence $a_1 a_n$. Next consider the SLT B that given a sequence $a_1 a_2$ outputs the sequence $a_2 a_1$ (this can be implemented by an SLT with look-back 2). On any input sequence of the form $a_1 \ldots a_n$ such that for every $1 < i \leq n$, $a_i \leq 5$, and $a_n > 5$, the function resulting by composing A with B should output the sequence $a_n a_1$. But this function can't be implemented using finite look-back. In particular, in order to output the symbol a_1 to the right of a_n, the symbol a_1 must be read by a transition that also reads the symbol a_n. But since n can be arbitrarily large, no finite look-back k would suffice.

5.2 A practical algorithm for composing ESFTs

In this section we present a sound algorithm for composing ESFTs that is not guaranteed to work in all cases, but works for many practical purposes [6]. Given two ESFTs, the algorithm first transforms them into Symbolic Transducers with registers [22] (STs), and by using the fact that STs are closed under composition, it computes their composition. The next step is a register elimination algorithm that tries to build an ESFT that is equivalent to the composed ST. This second step is sound but incomplete, and this is due to the fact that ESFTs are not closed under composition. Recall that the closure fails already for restricted classes of ESFTs (Corollary 2).

5.2.1 Symbolic transducers with registers

Registers provide a practical generalization of SFTs. SFTs with registers are called STs, since their state space (reachable by registers) may no longer be finite. An ST uses a *register* as a symbolic representation of states in addition to explicit (control) states. The rules of an ST are guarded commands with a symbolic input and output component that may use the register. By

using Cartesian product types, multiple registers are represented with a single (compound) register. Equivalence of STs is undecidable but STs are closed under composition [22].

Definition 12 A *Symbolic Transducer* or *ST* over $\sigma \to \gamma$ and register type τ is a tuple $A = (Q, q^0, \rho^0, R)$,

- Q is a finite set of *states*;
- $q^0 \in Q$ is the *initial state*;
- $\rho^0 \in \mathscr{U}^\tau$ is the *initial register value*;
- R is a finite set of *rules* $R = \Delta \cup F$;
- Δ is a set of *transitions* $r = (p, \varphi, o, u, q)$, also denoted $p \xrightarrow{\varphi/o;u} q$,
  - $p \in Q$ is the *start* state of r;
  - φ, the *guard* of r, is a $(\sigma \times \tau)$-predicate;
  - o, the *output* of r, is a finite sequence of $((\sigma \times \tau) \to \gamma)$-terms;
  - u, the *update* of r, is a $((\sigma \times \tau) \to \tau)$-term;
  - $q \in Q$ is the *end* state of r.
- F is a set of *final rules* $r = (p, \varphi, o)$, also denoted $p \xrightarrow{\varphi/o} \bullet$,
  - $p \in Q$ is the *start* state of r;
  - φ, the *guard* of r, is a τ-predicate;
  - o, the *output* of r, is a finite sequence of $(\tau \to \gamma)$-terms.

All ST rules in R have lookahead 1 and all final rules have lookahead 0. Longer lookaheads are not needed because registers can be used to record history, in particular they may be used to record previous input characters. A canonical way to do so is to let τ be σ^* that records previously seen characters, where initially $\rho^0 = []$, indicating that no input characters have been seen yet.

An ESFT transition

$$p \xrightarrow[3]{\lambda(x_0,x_1,x_2).\varphi(x_0,x_1,x_2)/\lambda(x_0,x_1,x_2).o(x_0,x_1,x_2)} q$$

can be encoded as the following set of ST rules where p_1 and p_2 are new states

$$p \xrightarrow{(\lambda(x,y).true)/[];\lambda(x,y).cons(x,nil)} p_1 \quad p_1 \xrightarrow{(\lambda(x,y).true)/[];\lambda(x,y).cons(x,y)} p_2$$
$$p_2 \xrightarrow{(\lambda(x,y).\varphi(y[1],y[0],x))/\lambda(x,y).o(y[1],y[0],x);\lambda(x,y).nil} q$$

Final rules are encoded similarly. The only difference is that q above is $\bullet$ and the register update is not used in the third rule. An ST rule $(p, \varphi, o, u, q) \in R$ denotes the following set of concrete transitions:

$$[\![(p, \varphi, o, u, q)]\!] \stackrel{\text{def}}{=} \{(p, s) \xrightarrow{a/[\![o]\!](a,s)} (q, [\![u]\!](a, s)) \mid (a, s) \in [\![\varphi]\!]\}$$

A final ST rule $(p, \varphi, o) \in F$ denotes the following set of concrete transitions:

$$[\![(p, \varphi, o)]\!] \stackrel{\text{def}}{=} \left\{(p, s) \xrightarrow{[]/[\![o]\!](s)} \bullet \mid s \in [\![\varphi]\!]\right\}$$

The reachability relation $p \xrightarrow{\boldsymbol{a}/\boldsymbol{b}}_A q$ for $\boldsymbol{a} \in \Sigma^*, \boldsymbol{b} \in \Gamma^*, p \in (Q \times \mathscr{U}^\tau), q \in (Q \times \mathscr{U}^\tau) \cup \{\bullet\}$ is defined analogously to ESFTs and $\mathscr{T}_A(\boldsymbol{a}) \stackrel{\text{def}}{=} \{\boldsymbol{b} \mid (q^0, \rho^0) \xrightarrow{\boldsymbol{a}/\boldsymbol{b}} \bullet\}$.

The following example illustrates a simplified case when an ESFT is turned into an ST by saving a single character in a register, thus $\tau = \sigma$ in this case. The resulting ST is then composed with itself. We abbreviate an ESFT rule $p \xrightarrow[k]{\lambda\bar{x}.\varphi(\bar{x})/\lambda\bar{x}.o(\bar{x})} q$, where $|\bar{x}| = k$, by $p \xrightarrow[k]{\varphi(\bar{x})/o(\bar{x})} q$, and an ST rule $p \xrightarrow{\lambda(x,y).\varphi(x,y)/\lambda(x,y).o(x,y);\lambda(x,y).u(x,y)} q$ by $p \xrightarrow{\varphi(x,y)/o(x,y);u(x,y)} q$.

Example 3 Let A be an ESFT with the single state q and the rules

$$q \xrightarrow[2]{true/[x_1,x_0,x_0]} q, \quad q \xrightarrow[0]{true/[]} \bullet.$$

For example, A transforms the input [0, 1, 2, 3] into the output [1, 0, 0, 3, 2, 2]. The corresponding ST of A, A^{st}, has the following transitions

$$q \xrightarrow{true/[];\, x} p, \quad p \xrightarrow{true/[x,y,y];\, 0} q, \quad q \xrightarrow{true/[]} \bullet,$$

where y refers to the register, x refers to the current input, p is a new state, and the initial register value is assumed to be 0. The first transition outputs nothing, and saves the current character in the register. The second transition outputs the current character followed by outputting the register twice in a row, and resets the register back to its initial value. Let us consider the self-composition $A^{\mathrm{st}} \circ A^{\mathrm{st}}$. The register of $A^{\mathrm{st}} \circ A^{\mathrm{st}}$ has the type $\sigma \times \sigma$ whose first component y_0 is the register of first instance of A and whose second component y_1 is the register of the second instance of A. The composed transitions are:

$$q_0 \xrightarrow{true/[];(x,y_1)} q_1, \quad q_1 \xrightarrow{true/[y_0,x,x];(0,y_0)} q_2,$$
$$q_2 \xrightarrow{true/[];(x,y_1)} q_3, \quad q_3 \xrightarrow{true/[x,y_1,y_1,y_0,y_0,y_0];(0,0)} q_0, \quad q_0 \xrightarrow{true/[]} \bullet$$

where q_0 is the initial state, initially register $y = (0, 0)$, i.e., $y_0 = y_1 = 0$. Only q_0 is a final state (has a finalizer with the empty output).

5.2.2 A register elimination algorithm

In this section we describe an algorithm for transforming a class of STs into ESFTs. The core idea that underlies the register elimination algorithm is a symbolic generalization of the classic state elimination algorithm for converting an NFA into a regular expression (see e.g. [26, Sect. 3.3]), that uses the notion of extended automata whose transitions are labelled by regular expressions. Here the labels of the ST are predicates over sequences of elements of fixed lookahead. Essentially the intermediate data structure of the algorithm is an "Extended ST". We often abbreviate a transition $p \xrightarrow{\lambda(x,y).\varphi(x,y)/\lambda(x,y).o(x,y);\lambda(x,y).u(x,y)} q$ by $p \xrightarrow{\varphi(x,y)/o(x,y);u(x,y)} q$.

Input: ST $A^{\sigma/\gamma;\tau}$.
Output: $\perp$ or an ESFT over $\sigma \to \gamma$ that is equivalent to A.

1. Lift A to the input type σ^*. Replace each transition $p \xrightarrow{\varphi(x,y)/o(x,y);u(x,y)} q$ with the following transition where *nil* is the empty list of type σ^*.

$$p \xrightarrow[(1)]{x \neq nil \wedge \varphi(head(x),y)/o(head(x),y);(head(x),y)} q$$

Intuitively, x must be a non-empty list, i.e., input *nil* is not allowed. In the rules we indicate, for clarity, that x is a list of length at least k by annotating the transition with subscript (k). Apply similar transformation to final rules.

2. Repeat the steps 2.a–2.c while there exists a state that does not have a self loop (a self loop is a transition whose start and end states are equal).

2.a Choose a state p that is not the state of any self loop and is not the initial state.

2.b For all transitions $p_1 \xrightarrow[(k)]{\varphi_1/o_1;u_1} p \xrightarrow[(\ell)]{\varphi_2/o_2;u_2} p_2$ in R:

$$\textbf{let } \varphi = \lambda(x,y).\varphi_1(x,y) \wedge \varphi_2(tail^k(x), u_1(x,y))$$
$$\textbf{let } o = \lambda(x,y).o_1(x,y) \cdot o_2(tail^k(x), u_1(x,y))$$
$$\textbf{let } u = \lambda(x,y).u_2(tail^k(x), u_1(x,y))$$
$$\textbf{if } IsSat(\varphi) \textbf{ then add } p_1 \xrightarrow[(k+\ell)]{\varphi/o;u} p_2 \text{ as a new rule.}$$

2.c Delete the state p.

3. If all guards and outputs do not depend on the register, remove the register from all the rules in the ST and return the resulting ESFT. Otherwise return $\perp$.

After the first step, the original ST accepts an input $[a_0, a_1, a_2]$ and produces output v iff the transformed ST accepts $[cons(a_0, \_), cons(a_1, \_), cons(a_2, \_)]$ and produces output v, where the tails _ are unconstrained and irrelevant. Step 2 further groups the inputs characters, e.g., to $[cons(a_0, cons(a_1, \_)), cons(a_2, \_)]$, etc, while maintaining this input/output property with respect to the original ST. Finally, in step 3, turning the ST into an ESFT, leads to elimination of the register as well as lowering of the character sort back to σ, and replacing each occurrence of $head(tail^k(x))$ with corresponding individual tuple element variable x_k. Soundness of the algorithm follows.

The algorithm omits several implementation aspects that have considerable effect on performance. One important choice is the order in which states are removed. In our implementation the states with lowest total number of incoming and outgoing rules are eliminated first. It is also important to perform the choices in an order that avoids unreachable state spaces. For example, the elimination of a state p in step 2 may imply that φ is unsatisfiable and consequently that p_2 is unreachable if the transition from p is the only transition leading to p_2. In this case, if p is reachable from the initial state, choosing p_2 before p in step 2 would be wasteful.

For the class of STs in which no register value is passed *through* a loop the algorithm always succeeds. Intuitively this capture the cases in which there are no symbolic dependencies between separate loop iterations. In other words, the register is only used through a fixed number of states, and reset after that. The following example illustrates a case for which the register elimination succeeds.

Example 4 Consider the ST $A^{st} \circ A^{st}$ from Example 3. We follow the steps of the algorithm and show how the composed ST can be transformed into an equivalent ESFT.

Step 1: We lift the input type to σ^* so that a lifted character is a list (sequence of type σ^*). We let $|x| > k$ abbreviate the formula $\bigwedge_{i=0}^{k} tail^k(x) \neq nil$. When $|x| > i$, we write x_i for the i'th element $head(tail^i(x))$ of x. Recall that y_0 is the first component of the register and y_1 is the second component. The lifted transitions of $A^{st} \circ A^{st}$ are:

$$q_0 \xrightarrow[(1)]{x\neq nil/[];(x_0,y_1)} q_1, \quad q_1 \xrightarrow[(1)]{x\neq nil/[y_0,x_0,x_0];(0,x_0)} q_2,$$
$$q_2 \xrightarrow[(1)]{x\neq nil/[];(x_0,y_1)} q_3, \quad q_3 \xrightarrow[(1)]{x\neq nil/[x_0,y_1,y_1,y_0,y_0,y_0];(0,0)} q_0, \quad q_0 \xrightarrow{true/[]} \bullet$$

Repeat Step 2: Choose $p = q_1$. Eliminate q_1 by merging the first two transitions. Here $k = \ell = 1$. Observe that x_0 in the second rule becomes $x_{0+k} = x_1$ and y_0 refers to the first sub-register update of the first transition, that is x_0. The merged transition is

$$q_0 \xrightarrow[(2)]{|x|>1/[x_0,x_1,x_1];(0,x_1)} q_2$$

Next, choose $p = q_2$. Eliminate q_2 similarly by replacing the new transition to q_2 and the original transition from q_2 by

$$q_0 \xrightarrow[(3)]{|x|>2/[x_0,x_1,x_1];(x_2,x_1)} q_3$$

Finally, choose $p = q_3$. Eliminate q_3 similarly by replacing the new transition to q_3 and the original transition from q_3 by

$$q_0 \xrightarrow[(4)]{|x|>3/[x_0,x_1,x_1,x_3,x_0,x_0,x_2,x_2,x_2];(0,0)} q_0$$

Step 3: It is now safe to remove the register because it is not being used any more in any guard or update. So the final ESFT, say AA, has the rules:

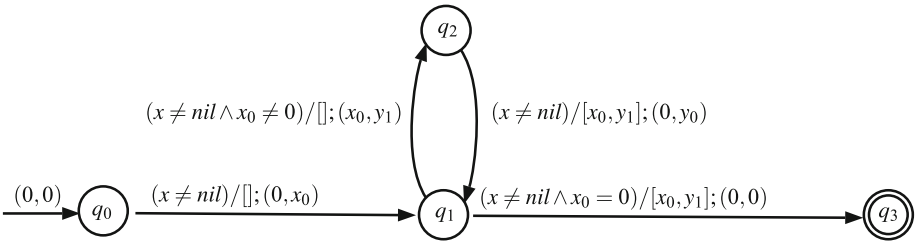
$$q_0 \xrightarrow[4]{true/[x_0,x_1,x_1,x_3,x_0,x_0,x_2,x_2,x_2]} q_0, \quad q_0 \xrightarrow[0]{true/[]} \bullet$$

where the lifting has been undone and here each variable x_i is of type σ. For example $AA([0, 1, 2, 3]) = [0, 1, 1, 3, 0, 0, 2, 2, 2]$.

We now discuss the cases when the algorithm fails. We first discuss the cases when it *should* fail, or else the algorithm would be unsound, and then identify two cases when it fails due to incompleteness (w.r.t. the class of STs that have an equivalent ESFT).

We know from Theorem 10 that already Cartesian ESFTs are not closed under composition. For example, if we take the ESFTs A and B from the proof of Theorem 10, first transform them into equivalent STs and then compose the STs, then the resulting ST cannot be transformed back into an ESFT. Although we know the algorithm will fail, it is nevertheless useful to see how this happens in the following example.

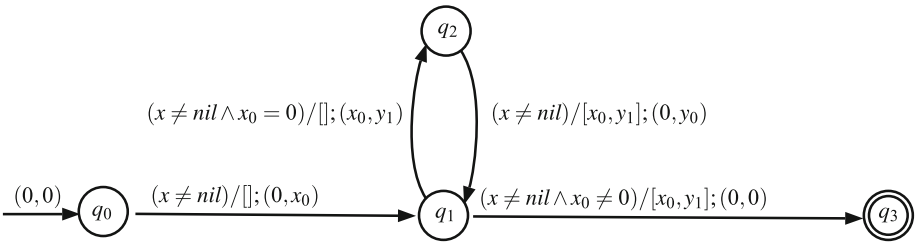
Example 5 Consider ESFTs A and B from the proof of Theorem 10. Modify B so that it is deterministic, by letting the guard on the self-loop on q_1 be $x_0 \neq 0$ and the guard on the finalizer from q_1 be $x_0 = 0$. The composition $B^{\text{st}} \circ A^{\text{st}}$, after lifting the input type, is then the following ST (recall that $B^{\text{st}} \circ A^{\text{st}}(w) = A^{\text{st}}(B^{\text{st}}(w))$):



If we apply the register elimination algorithm to this ST, it may first eliminate the state q_2, by creating the transition $q_1 \xrightarrow[(2)]{|x|>1 \wedge x_0 \neq 0/[x_1,y_1];(0,x_0)} q_1$. After this step the algorithm stops and returns $\perp$. □

Another reason why the algorithm fails for some inputs is due to Theorem 11, which states there are cases in which ESFTs can be composed, but their composition cannot be effectively constructed. In particular composing the two ESFTs of Theorem 11 requires *loop unrolling* in order to construct an equivalent ESFT. In this case the algorithm fails, as shown by the following example.

Example 6 Consider again the ESFTs A and B from the proof of Theorem 10. This time modify B so that the guard on the self-loop on q_1 is $x_0 = 0$ and the guard on the finalizer from q_1 is $x_0 \neq 0$. The composition $B^{\text{st}} \circ A^{\text{st}}$, after lifting the input type, is then the following ST, that is very similar to the one in Example 5:



If we apply the register elimination algorithm to this ST, it may again, first eliminate the state q_2, by creating the transition $q_1 \xrightarrow[(2)]{|x|>1 \wedge x_0=0/[x_1,y_1];(0,0)} q_1$. The algorithm is not able to detect that unrolling the loop once will remove the dependency on y_1 from the output (because y_1 will be fixed to 0 in all remaining iterations). The algorithm stops and returns $\perp$. □

The register elimination algorithm also returns undefined when the register is used in a way that does not affect the transducer's semantics. For example, if there is an output element $r - r$ where r is an integer valued register, then the value will always be 0. But the algorithm does not perform any *theory specific reasoning* and will therefore not detect such cases.

5.2.3 A practical composition algorithm

We now have all the ingredients necessary to try to compose ESFTs. The algorithm proceeds as follows. Given two ESFTs A and B:

1. compute two STs A' and B' equivalent to A and B respectively;
2. compute a ST $C' = A' \circ B'$;
3. run the register elimination algorithm on the ST C' and if it terminates output the ESFT C equivalent to $A \circ B$.

6 Experiments and potential 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 tools Bek[2] and Bex[3].

[2] Bek is available at: http://www.rise4fun.com/Bek.

[3] Bex is available at: http://www.rise4fun.com/Bex.

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

	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

I is the identity transducer

Analysis of string encoders A string encoder E transforms input strings in a given format into output strings in a different format. A decoder D inverts such a transformation. For coders E and D to be correct, the following equalities should hold: $E \circ D \overset{1}{=} I$ and $D \circ E \overset{1}{=} I$ (where I is the identity transducer).

We illustrated in Example 1 how the BASE64 encoder and decoder can be modeled using Cartesian ESFTs. Similarly, we can model BASE32, BASE16, and UTF8 coders. Using the equivalence and composition procedures presented in this paper we proved that the equality presented above hold for all these coders. Table 1 shows the corresponding running times. The first half of Table 1 shows the lookahead sizes of both encoders and decoders, while the second half shows the running times for checking correctness. Composition times (typically 1–2 ms) are included in the measurements.

Interestingly, during our experiments we identified wrong implementations of the UTF8 encoder/decoder for which the equivalence algorithm of Sect. 4 terminated, while the semi-decision procedure presented in [6] 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) [20]. 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) [20] 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. To support this hypothesis we observe that examples shown in [20, Figs. 2, 3] can be represented as deterministic ESFAs with few transitions. For example the language `^/\ncmd[^\n]{200}$` can be succinctly modeled as a deterministic ESFA with one transition! Moreover, the ability to compile ESFA to Symbolic Automata with registers 5.2.1 makes this model appealing for efficient deterministic left-to-right DPI.

Network Protocol Conversions Deep packet inspection can be naturally extended by adding data manipulation. In this setting, we are interested in deterministic ESFTs which can commit their output at every transition and without having to process 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 simplified translation from an IPv4 header to an IPv6 header[4] can be implemented with a deterministic ESFT with less than 50 transitions. However, the same transformation using an SFT would need to remember portions of the input packet and therefore require more than 100000 states and transitions.

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

Verification of List Manipulating Programs In [8] it was shown how SFTs can be used to verify pre and post conditions of list manipulating programs. However, SFTs can only model programs in which each node in the output list depends on at most one node in the input list. ESFTs can be used to mitigate this problem as they can be used to model sequential pattern matching. For example, the CAML guards `x1::x2::xs -> (x1+x2)::(f2 xs)` and `x1::x2::x3::xs -> (x1+x2+x3)::(f3 xs)`, can be naturally expressed as ESFT transitions. Let's consider two functions f_2 and f_3, both of type $list\ int \rightarrow list\ int$, that respectively contain the two guards defined above. These functions can be modeled as ESFTs. Using the one-equality algorithm of Sect. 4 and the composition algorithm of Sect. 5.2.3 we were able to prove that $\forall l.f_3(f_2\ l) \stackrel{\perp}{=} f_2(f_3\ l)$ 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, although the language Bek [11] is optimized for 16 bits characters, an ESFT of lookahead can be used to model 32 bits characters without having to change the underlying solver.

7 Related work

7.1 From SFA/SFT to ESFA/ESFT

The concept of automata with predicates instead of concrete symbols was first mentioned in [24] and was first discussed in [17] in the context of natural language processing. SFAs are further studied in [7] in the context of automata minimization.

Symbolic finite transducers (SFTs) were originally introduced in [11] 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 [22]. Symbolic transducers (STs) that allow the use of registers are also defined in [22]. Full equivalence of finite state transducers is undecidable [10], and already so for very restricted fragments [12]. In the single-valued case, decidability was established in [18], and extended to the finite-valued case in [3,25]. Symbolic finite transducers are extended to tree structures in [8].

ESFTs were introduced in [6] as a succinct and more analysable representation of a subclass of symbolic transducers (STs). The main result in [6] is the register elimination technique that provides a way to construct ESFTs from STs. The equivalence problem is then studied in [5] where it is shown to be decidable for Cartesian ESFTs, and undecidable in the general case. An algorithm for checking whether a predicate is monadic (finite disjunction of Cartesian predicates) is proposed in [21].

Register elimination is further studied in [23] where it is called grouping and is combined with explicit state-space exploration in order to extend the algorithm to a larger class of STs. In [23] the algorithm is used in a pipeline of techniques for transforming Bek programs into a parallelizable form.

7.2 Models over infinite alphabets

In recent years there has been considerable interest in automata that accept words over infinite alphabets [13,19]. In this line of work, symbols can only be compared using equality and arbitrary predicates are not allowed, making the proposed models incomparable to those analyzed in this paper. In our paper, we focus on proving negative and positive properties of

ESFAs and ESFTs over arbitrary decidable Boolean algebras. While we do not investigate specific theories, it would be interesting to understand whether the properties we discussed hold when considering an alphabet theory that only supports equality.

Symbolic visibly pushdown automata (SVPA) [4] operate over hierarchical words and can use binary predicates to relate symbols appearing at different positions in the input, while retaining decidable equivalence and Boolean closure properties. This is achieved by carefully restricting what symbols can be related. SVPAs and ESFAs are orthogonal in expressiveness and they operate over different structures (words vs nested words).

Symbolic finite transducers with look-back k (k-SLTs) [2] have a sliding window of size k that allows, in addition to the current input character, references of up to $k-1$ previous characters. SLTs use only final states, because it is unclear how to support non-final states in the context of learning. As we showed in this paper unlike what is claimed in [2], k-SLTs are *not* closed under composition, and equivalence of k-SLTs is *undecidable*.

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.

7.3 Models over finite alphabets

Extended finite automata (XFA) are introduced in [20] for network packet inspection. XFAs are a succinct representation of DFAs that uses registers to store and inspect values. History-based finite automata [14] are another extension of DFAs introduced in the context of network intrusion detection, that uses a single register (bit-vector) to keep track of the symbols read so far. In both models the register is used together with the input character to determine when a transition is enabled. Both these models focus on succinctness and the differ from ESFAs in two ways: (1) they only support finite alphabets; and (2) they can relate symbols at arbitrary positions, while ESFAs can only relate adjacent positions. We have not investigated the application of ESFAs to network packet inspection and network intrusion detection, but we think that ESFAs can help achieving a further level of succinctness in these domains.

Extended top-down tree transducers (ETTTs) [15] are commonly used in natural language processing. ETTTs 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.

8 Conclusion

We proved fundamental decidability results and closure properties for extended symbolic finite automata and transducers. First, we investigated the problem of deciding transducer equivalence and established a sharp boundary between decidability (the Cartesian case with any decidable background) and undecidability (the non-Cartesian case with a background of successor arithmetic). Second, although we showed that extended symbolic finite transducers are not closed under composition, we provided an incomplete but practically effective composition algorithm that we used to prove the correctness of real-world string encoders.

Future directions include identifying subclasses of these models that are effectively closed under composition, and extending the models to trees.

Acknowledgments Loris D'Antoni's research was supported by National Science Foundation Expeditions in Computing award CCF 1138996.

References

1. Alur R, Cerný P (2011) Streaming transducers for algorithmic verification of single-pass list-processing programs. In: POPL'11, ACM, pp 599–610
2. Botincan M, Babic D (2013) Sigma*: symbolic learning of input-output specifications. In: POPL'13, ACM, pp 443–456
3. Culic K, Karhumäki J (1986) The equivalence of finite-valued transducers (on HDTOL languages) is decidable. Theor Comput Sci 47:71–84
4. D'Antoni L, Alur R (2014) Symbolic visibly pushdown automata. In: Biere A, Bloem R (eds) Computer aided verification, vol 8559. Lecture notes in computer science. Springer, New York, pp 209–225. doi:10.1007/978-3-319-08867-9-14
5. D'Antoni L, Veanes M (2013) Equivalence of extended symbolic finite transducers. In: Proceedings of the 25th international conference on computer aided verification, CAV'13, Springer, Berlin, pp 624–639. doi:10.1007/978-3-642-39799-8_41
6. D'Antoni L, Veanes M (2013) Static analysis of string encoders and decoders. In: Giacobazzi R, Berdine J, Mastroeni I (eds) VMCAI, LNCS, vol 7737, Springer, pp 209–228
7. D'Antoni L, Veanes M (2014) Minimization of symbolic automata. In: Proceedings of the 41st ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL '14, ACM, New York, pp 541–553. doi:10.1145/2535838.2535849
8. D'Antoni L, Veanes M, Livshits B, Molnar D (2014) Fast: a transducer-based language for tree manipulation. In: Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, PLDI '14, ACM, New York, pp 384–394. doi:10.1145/2594291.2594309
9. Fülöp Z, Vogler H (1998) Syntax-directed semantics: formal models based on tree transducers. EATCS. Springer, New York
10. Griffiths T (1968) The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines. J ACM 15:409–413
11. Hooimeijer P, Livshits B, Molnar D, Saxena P, Veanes M (2011) Fast and precise sanitizer analysis with Bek. In: USENIX security, pp 1–16
12. Ibarra O (1978) The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. SIAM J Comput 4:524–532
13. Kaminski M, Francez N (1994) Finite-memory automata. TCS 134(2):329–363
14. Kumar S, Chandrasekaran B, Turner J, Varghese G (2007) Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. In: ANCS 2007, ACM/IEEE, pp 155–164
15. Maletti A, Graehl J, Hopkins M, Knight K (2009) The power of extended top-down tree transducers. SIAM J Comput 39(2):410–430. doi:10.1137/070699160
16. Mohri M (1997) Finite-state transducers in language and speech processing. Comput Linguist 23(2):269–311. http://dl.acm.org/citation.cfm?id=972695.972698
17. van Noord G, Gerdemann D (2001) Finite state transducers with predicates and identities. Grammars 4(3):263–286
18. Schützenberger MP (1975) Sur les relations rationnelles. In: GI conference on automata theory and formal languages. LNCS 33:209–213
19. Segoufin L, Segoufin L (2006) Automata and logics for words and trees over an infinite alphabet. CSL. Springer, Berlin Heidelberg, pp 41–57
20. Smith R, Estan C, Jha S, Kong S (2008) Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. In: SIGCOMM '08, ACM, pp 207–218
21. Veanes M, Bjørner N, Nachmanson L, Bereg S (2014) Monadic decomposition. In: CAV'14, LNCS, vol 8559, Springer, pp 628–645
22. Veanes M, Hooimeijer P, Livshits B, Molnar D, Bjorner N (2012) Symbolic finite state transducers: algorithms and applications. In: POPL'12, ACM, pp 137–150
23. Veanes M, Mytkowicz T, Molnar D, Livshits B (2015) Data-parallel string-manipulating programs. In: Proceedings of the 42nd ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL'15, ACM, pp 139–152

24. Watson BW (1996) Implementing and using finite automata toolkits. Nat Lang Eng 2(4):295–302. doi:10.1017/S135132499700154X
25. Weber A (1993) Decomposing finite-valued transducers and deciding their equivalence. SIAM J Comput 22(1):175–202
26. Yu S (1997) Regular languages. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages, vol 1. Springer, New York, pp 41–110