Symbolic Transducers

Microsoft Research Technical Report MSR-TR-2011-3
January 21, 2011

Nikolaj Bjørner and Margus Veanes
Microsoft Research, Redmond, WA 98052
{nbjorner,margus}@microsoft.com

***Abstract*—Symbolic Finite Transducers, or SFTs, is a representation of finite transducers that annotates transitions with logical formulae to denote sets of concrete transitions. This representation has practical advantages in applications for web security analysis, as it provides ways to succinctly represent web sanitizers that operate on large alphabets. More importantly, the representation is also conducive for efficient analysis using state-of-the-art theorem proving techniques. Besides introducing SFTs we provide algorithms for various closure properties including composition and domain restriction. A central result is that equivalence of SFTs is decidable when there is a fixed bound on how many different values that can be generated for arbitrary inputs. In practice, we use a semi-decision algorithm, encoded axiomatically, for non-equivalence of arbitrary SFTs. We show that several of the main results lift to a more expressive version of SFTs with Registers, SFTRs. They admit a fixed set of registers that can be referenced in the logical formulae, updated by input characters, or used to generate output.**

I. Introduction

State machines, such as automata and transducers typically use finite alphabets. This is both helpful when formulating the main algorithms and it is realistic when considering applications from text processing. Furthermore, implementations can apply compression algorithms on the transition functions when the alphabet is large. In symbolic analysis of automata, however, there are practical advantages to formulating transitions directly as formulae, and sometimes use an abstract, possibly infinite, alphabet. We are here interested in finite transducers that arise from applications such as web sanitizers.

Our main contributions are:

- In Section III we show how several properties and algorithms known from finite transducers lift to SFTs even when the alphabets are abstract. Main algorithms include composition and domain restriction.
- Equivalence of SFTs is shown to be decidable when there is a fixed bound on how many different values that can be generated for arbitrary inputs. The result, established in III-B4, is technical and of independent interest.
- Section IV introduces SFTs with registers and lifts algorithms to this more expressive class.
- We present a principled axiomatic encoding of a semi-decision algorithm for non-equivalence checking in Section V. It applies to arbitrary SFTRs and represents the method used in practice.

```
1:   static string EncodeHtmlX(string strInput)
2:   {
3:     if (strInput == null) return null;
4:     if (strInput.Length == 0) return string.Empty;
5:     StringBuilder b = new StringBuilder();
6:     foreach (char c in strInput)
7:       if (((‘a’ <= c) && (c <= ‘z’)) || (c == ‘,’) ||
8:           ((‘A’ <= c) && (c <= ‘Z’)) || (c == ‘ ’) ||
9:           ((‘0’ <= c) && (c <= ‘9’)) || (c == ‘.’) ||
10:          (c == ‘-’) || (c == ‘_’) || (c == ‘;’))
11:        b.Append(c);
12:      else {
13:        b.Append(string.Format("&#x{0:X}",(int)c));
14:        b.Append(";");
15:      }
16:    return b.ToString();
17:  }
```

Fig. 1. `AntiXSS` html sanitizer with hexadecimal formatting.

A. Examples and an Application to Web Sanitizers

We here illustrate the use of SFT analysis on web security analysis using a running example. We later develop the necessary algorithms for this analysis. *Cross site scripting* (*XSS*) *attacks* are a major concern in web applications, and happen as a result of untrusted data leaking across web sites. Part of data may be interpreted as code (e.g. JavaScript) by a browser, that may end up being executed in the browser of another user. The first line of defense against XSS attacks is the use of *sanitizers* in web servers, that escape or remove potentially harmful strings. Although sanitizers are typically small programs, in the order of tens of lines of code, writing them correctly is difficult. The work in [1] introduces a domain specific language BEK based on SFTs for writing and analyzing sanitizers. The axiomatic approach to SFTs, introduced in the current paper, has been implemented as part of the underlying algorithmic support in BEK and uses the SMT solver Z3 [2].

***Example** 1 (Sanitizer Program):* A typical sanitizer, shown in Fig. 1, is a version of a sanitizer from a public Microsoft AntiXSS library. ■

We represent the sanitizer program as a symbolic finite transducer. It uses transduction functions.

***Example** 2 (Transduction Functions):* In most modern programming languages, *strings* correspond to character sequences where characters use Unicode (UTF16) encoding. Assume that there is a sort BV_k, for $k \geq 1$, and that $\mathcal{U}^{\mathrm{BV}_k}$ is the domain of k-bit bit-vectors. The elements of $\mathcal{U}^{\mathrm{BV}_k}$ correspond to k-bit binary encodings of nonnegative integers from 0 to $2^k - 1$. A natural representation of Unicode characters

for symbolic analysis is as elements in $\mathcal{U}^{\mathrm{BV}_{16}}$. Assume the following operations, where $k = 16$:

$$\begin{array}{rl} <: & \mathrm{BV}_k \times \mathrm{BV}_k \to \mathrm{BOOL}, \\ \pi_m^n: & \mathrm{BV}_k \to \mathrm{BV}_k, \text{ for } 0 \le m < n \le k, \\ \oplus: & \mathrm{BV}_k \times \mathrm{BV}_k \to \mathrm{BV}_k, \end{array}$$

where $<$ corresponds to the underlying integer order and matches the lexicographic order over characters; π_m^n projects bits m through $n-1$ and pads the result with $k-n+m$ zeros; $\oplus$ is addition modulo 2^k. Then

$$\mathbf{h}_j(c) \stackrel{\text{def}}{=} \mathit{Ite}(9 < \pi_{4j}^{4j+4}(c),\ \pi_{4j}^{4j+4}(c) \oplus 55,\ \pi_{4j}^{4j+4}(c) \oplus 48)$$

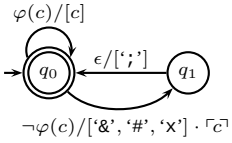
extracts the j'th 4-bit unit of c, $0 \le j \le 3$, and maps it to its hexadecimal representation (‘0’,‘1’,...,‘9’,‘A’,...,‘F’). ■

The transduction function allows defining a minimal symbolic transducer.

***Example** 3 (Transducer Guards):* The SFT below represents a so-called “string sanitizer”, where certain characters c in the input string, not satisfying the condition

$$\begin{array}{rl} \varphi(c): & (\text‘a’ \le c \wedge c \le \text‘z’) \vee (\text‘A’ \le c \wedge c \le \text‘Z’) \vee \\ & (\text‘0’ \le c \wedge c \le \text‘9’) \vee c = \text‘ ’ \vee c = \text‘.’ \vee \\ & c = \text‘,’ \vee c = \text‘-’ \vee c = \text‘_’ \vee c = \text‘;’ \end{array}$$

are in the output string replaced by their hexadecimal representation:



where $\ulcorner c \urcorner$ is the (up-to) four-character encoding of c:

$$\begin{array}{rl} \ulcorner c \urcorner \stackrel{\text{def}}{=} & \mathit{Ite}(\mathbf{h}_3(c) \neq \text‘0’, [\mathbf{h}_3(c), \mathbf{h}_2(c), \mathbf{h}_1(c), \mathbf{h}_0(c)], \\ & \mathit{Ite}(\mathbf{h}_2(c) \neq \text‘0’, [\mathbf{h}_2(c), \mathbf{h}_1(c), \mathbf{h}_0(c)], \\ & \mathit{Ite}(\mathbf{h}_1(c) \neq \text‘0’, [\mathbf{h}_1(c), \mathbf{h}_0(c)], [\mathbf{h}_0(c)]))) \end{array}$$

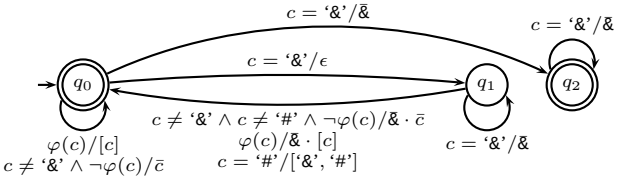
with $\mathbf{h}_j$'s as defined in Example 2. It is also straight-forward to rewrite the conditions into four transitions with simple guards and a fixed number of outputs each. ■

The main application of SFTs in the context of BEK is to formally verify key security properties of sanitizers. The two most important properties are *idempotence* (to determine if applying the same sanitizer twice matters) and *commutativity* (to determine if the order of applying different sanitizers matters). Since sanitizers are functions that take arbitrary input strings the corresponding SFTs are consequently single-valued and *total*, i.e., produce one output string for each input string. Equivalence checking of two corresponding SFTs A and B reduces therefore to partial equivalence checking, that we write as $A \cong B$.

***Example** 4 (Non-idempotent Sanitizers):* The SFT corresponding to Fig. 1, say A, is the one given in Example 3. The string formatting operation on line 13 corresponds to the transitions from state q_0 to q_1 in A. The question of idempotence of the sanitizer is the problem of deciding $A \circ A \cong A$. Fig. 3 defines a semi-decision procedure that finds a shortest-input counter-example, if one exists. This sanitizer is not idempotent, our implementation of the analysis algorithms finds a witness in less than a second. ■

In general, it is highly nontrivial to write sanitizers that are idempotent. For example, for the sanitizer in Fig. 1 the main problem is recognizing patterns that have been introduced in the output involving the characters ‘&’ and ‘#’ in order not to reencode those characters, for example to avoid double encodings such as `EncodeHtmlX("&")="&#x26;"` and `EncodeHtmlX("&#x26;") = "&#x26;&#x23;x26;"`.

***Example** 5 (Idempotent Sanitizers):* Suppose that the substring `"&#"` in the input must remain unchanged in the output. The following transducer captures this requirement:



where $\bar{c}$ stands for $[\text‘&’, \text‘#’, \text‘x’] \cdot \ulcorner c \urcorner \cdot [\text‘;’]$ and φ and ψ_i are defined in Example 3. Note that $\varphi(\text‘&’)$, $\varphi(\text‘#’)$ are false. In this case the analysis takes also less than a second and shows that the modified SFT *is idempotent*. However, the pattern `"&#"` is not precise enough, ideally, one should consider patterns that arise only as a result of santization, e.g. `"&#x"` is more precise and yields a similar but more complex extension of the original SFT. ■

Handcoding of sanitizers and SFTs such as the one in Example 5, is notoriously difficult are error-prone. In general, construction of SFTs from sanitizers can be automatized and the construction of special purpose SFTs that preserve certain patterns can be used together with SFT algorithms presented above to automatically construct and analyze properties of SFTs such as the one in Example 5.

***Example** 6 (Composition):* The following general technique can be used to aid the above construction. Consider an SFT $A_W^{\sigma/\mathrm{TUPLE}\langle \mathrm{BOOL}, \sigma \rangle}$ that, given a finite set W of “pattern” words in $\mathcal{U}^\sigma$, replaces each pattern $[a_0, a_1, \ldots, a_n]$ in the input with the word $[\langle \mathbf{true}, a_0 \rangle, \langle \mathbf{true}, a_1 \rangle, \ldots, \langle \mathbf{true}, a_n \rangle]$ and replaces any other character a in the input by $\langle \mathbf{false}, a \rangle$. Second, assume an SFT $A_f^{\mathrm{TUPLE}\langle \mathrm{BOOL}, \sigma \rangle / \sigma}$ that maps each character $\langle \mathbf{true}, a \rangle$ in the input to a and applies a transformation $f(a)$ to any character $\langle \mathbf{false}, a \rangle$. Then the composition $A_W \circ A_f$ is an SFT that performs the desired transformation f on input characters unless they occur in the context of W. The sort $\mathrm{TUPLE}\langle \sigma_0, \sigma_1 \rangle$ is a *tuple sort* and is associated with the usual operations (tuple constructor and projection functions). ■

***Example** 7 (Registers):* The main difficulty in encoding the transducer in Example 5 is having to introduce states that remember previous patters of characters that should not be replaced. By using registers, this encoding can be simplified and can be much more succinct. In order not to apply sanitization to characters occurring in a set of patterns W, use registers for remembering previous input characters. Suppose a single register r is needed (initially $r = \epsilon$) to remember the previous input character. Before outputting a transformation

$f(c)$ of the current character c, check if the pattern $r \cdot c$ occurs in W, in which case output $r \cdot c$ and set $r = \epsilon$, or if c extends a pattern, in which case let $r = c$ and output ϵ, or if c breaks a pattern, in which case output $f(r) \cdot f(c)$ and set $r = \epsilon$. ■

II. Preliminaries

Our work is based on classical automata theory, classical logic and model theory. We use terminology that is consistent with [3], [4], [5].

A. Finite Transducers and Automata

We recall the definition of a finite transducer [5]. Intuitively, a finite transducer is a generalization of a Mealy machine that may omit inputs and outputs and may be nondeterministic. We use ϵ as a special symbol denoting the empty word.

***Definition** 1:* A *finite transducer* (*FT*) A is defined as a six-tuple $(Q, q^0, F, I, O, \delta)$, where Q is a finite set of *states*, $q^0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, I is the *input alphabet*, O is the *output alphabet*, and δ is a finite *transition function* from $Q \times (I \cup \{\epsilon\})$ to $2^{Q \times O^*}$.

There exist several alternative definitions of FTs. By using the *standard form theorem* of FTs [5, Theorem 2.17], Definition 1 is easily seen to be equivalent to those definitions.

We indicate a component of an FT A by using A as a subscript. We often use the technically more convenient view of δ_A as a set of transitions Δ_A and write $p \xrightarrow{a/v}_A q$ for $(q, v) \in \delta_A(p, a)$. We omit the subscript A when it is clear from the context.

$$\begin{aligned} \Delta_A &\stackrel{\text{def}}{=} \Delta^{\epsilon}_A \cup \Delta^{\bar{\epsilon}}_A \\ \Delta^{\bar{\epsilon}}_A &\stackrel{\text{def}}{=} \{p \xrightarrow{a/v} q \mid (q, v) \in \delta_A(p, a), a \in I_A\} \\ \Delta^{\epsilon}_A &\stackrel{\text{def}}{=} \{p \xrightarrow{\epsilon/v} q \mid (q, v) \in \delta_A(p, \epsilon)\} \end{aligned}$$

Given a set V of elements, we write $v = [v_0, \ldots, v_{n-1}]$, for $v \in V^*$. For $v, w \in V^*$, $v \cdot w$ denotes the concatenation of v with w. (Both $[]$ and ϵ denote the empty sequence.)

Given $q_i \xrightarrow{u_i/v_i}_A q_{i+1}$ for $i < n$ we write $q_0 \overset{u/v}{\rightsquigarrow}_A q_n$ where $u = u_0 \cdot u_1 \cdot \ldots \cdot u_{n-1}$ and $v = v_0 \cdot v_1 \cdot \ldots \cdot v_{n-1}$. We write also $q \overset{\epsilon/\epsilon}{\rightsquigarrow}_A q$.

***Definition** 2:* An FT A induces the *transduction*,

$$T_A(u) \stackrel{\text{def}}{=} \{v \mid \exists q \in F_A \, (q^0_A \overset{u/v}{\rightsquigarrow} q)\}.$$

Two FTs A and B are *equivalent* if $T_A = T_B$.

We define $\mathbf{d}(A)$ as the underlying nondeterministic finite automaton with epsilon moves (ϵNFA) that is obtained from the FT A by eliminating outputs on all transitions. We write $L(B)$ for the language accepted by an ϵNFA B.

***Definition** 3:* An FT A is *finite-valued* if there exists k such that for all $u \in I^*_A$, $|T_A(u)| \leq k$; A is *single-valued* if for all $u \in I^*_A$, $|T_A(u)| \leq 1$.

***Definition** 4:* An FT A is a *generalized sequential machine* or *GSM* if $\Delta^{\epsilon}_A = \emptyset$. We say A is *input-ϵ-free*.

Definition 4 is consistent with [5], [6]. However, the definition of a GSM is not standardized in the literature. Some sources define GSMs without a dedicated set of final states [7].

***Definition** 5:* An FT A is *deterministic* if $\mathbf{d}(A)$ is deterministic.

There exist single-valued FTs for which there exists no equivalent deterministic FT (e.g., an FT that removes all input symbols after the *last occurrence* of a given symbol.) Conversely, determinism does not imply single-valuedness, since several transitions with same input but distinct outputs may collapse into single transitions in $\mathbf{d}(A)$. Other definitions of deterministic FTs (allowing input-ϵ) are used by some authors [6]. Definition 5 is consistent with [5].

B. Background Structure and Models

We work modulo a *background* structure $\mathcal{U}$ over a language $\Gamma_{\mathcal{U}}$ that is multi-sorted. We also write $\mathcal{U}$ for the universe (domain) of $\mathcal{U}$. For each sort σ, $\mathcal{U}^{\sigma}$ denotes a nonempty sub-domain of $\mathcal{U}$. There is a Boolean sort BOOL, $\mathcal{U}^{\text{BOOL}} = \{\mathbf{true}, \mathbf{false}\}$, and the standard logical connectives are assumed to be part of the background. *Terms* are defined by induction as usual and are assumed to be well-sorted. Function symbols with range sort BOOL are called relation symbols. Boolean terms are called formulas or predicates. A term without free variables is *closed*.

An *uninterpreted function symbol of arity* $n \geq 1$ is a function symbol $f \notin \Gamma_{\mathcal{U}}$ with a *domain sort* $\sigma_1 \times \cdots \times \sigma_n$ and a *range sort* σ. An *interpretation for* f is a function from $\mathcal{U}^{\sigma_1} \times \cdots \times \mathcal{U}^{\sigma_n}$ to $\mathcal{U}^{\sigma}$. An *uninterpreted constant* is a constant $c \notin \Gamma_{\mathcal{U}}$ of some sort σ. An *interpretation for* c is an element of $\mathcal{U}^{\sigma}$. By convention, a constant is also called a *function symbol of arity* 0.

We write $\Sigma(t)$ for the set of all uninterpreted function symbols that occur in a term t. Given a set of uninterpreted function symbols Σ, t is a *term over* Σ, or a *Σ-term* if $\Sigma(t) \subseteq \Sigma$. We say *$\Sigma$-model* for an expansion of $\mathcal{U}$ to $\Gamma_{\mathcal{U}} \cup \Sigma$. The interpretation of a closed Σ-term t in a Σ-model M, is denoted by t^M and is defined by induction as usual. There is a background function (symbol) Ite:BOOL $\times \sigma \times \sigma \to \sigma$ for each sort σ and

$$Ite(\varphi, t, f)^M = \text{if } \varphi^M \text{ then } t^M \text{ else } f^M$$

Let φ be a closed Σ-formula. A Σ-model M *satisfies* φ or φ *is true in* M or $M \vDash \varphi$, if $\varphi^M = \mathbf{true}$; φ is *satisfiable* if it has a model, denoted by $IsSat(\varphi)$; φ is *true* if $\varphi^M = \mathbf{true}$ for all Σ-models M.

For each sort σ let c_σ stand for a *default fixed uninterpreted constant* of sort σ. We omit the sort σ when it is clear from the context. Let $\mathcal{T}^{\sigma}(\Sigma)$ denote the set of all closed terms of sort σ only using uninterpreted symbols from Σ, $\mathcal{T}^{\sigma}$ stands for $\mathcal{T}^{\sigma}(\Sigma)$ where Σ is an infinite set of ininterpreted constants of some fixed sort. Unless stated otherwise, we assume that $\mathcal{T}^{\sigma}$ is quantifier free, closed under substitutions, Boolean operations, and equality. $\mathcal{F}$ stands for $\mathcal{T}^{\text{BOOL}}$.

III. Symbolic Transducers

Symbolic automata provide a representation of automata where several transitions from a given source state to a given target state may be combined into a single transition

with a symbolic label denoting multiple concrete labels. This representation naturally separates the finite state graph from the character representation.

***Definition** 6:* A *Symbolic Finite Transducer (SFT) A over Γ with input sort ι and output sort o,* or $A^{\iota/o}_{\Gamma}$, is a six-tuple $(Q, q^0, F, \iota, o, \Delta)$, where Q is a finite set of *states*, $q^0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, ι is the *input sort*, o is the *output sort*, and $\Delta = \Delta^{\bar{\epsilon}} \cup \Delta^{\epsilon}$,

$$\begin{aligned} \Delta^{\bar{\epsilon}}: &\quad Q \times \mathcal{F}(c_\iota) \times (\mathcal{T}^o(c_\iota))^* \times Q \\ \Delta^{\epsilon}: &\quad Q \times \{\epsilon\} \times (\mathcal{T}^o)^* \times Q \end{aligned}$$

is a finite *symbolic transition relation*.

A single transition $(p, \varphi, \mathbf{u}, q) \in \Delta_A$ is also denoted by $p \xrightarrow{\varphi/\mathbf{u}}_A q$ or $p \xrightarrow{\varphi/\mathbf{u}} q$ when A is clear from the context; φ is called the *input condition* or *guard* of the transition and $\mathbf{u}$ is called the *output sequence* of the transition. Let I_A denote the set of non-epsilon input conditions in Δ_A. Let O_A denote the set of output terms in Δ_A.

The definition of a *symbolic finite automaton* (*SFA*) is the special case of an SFT whose outputs are empty. A transition of an SFA A^{ι} is denoted by $p \xrightarrow{\varphi} q$ where $\varphi \in I_A \cup \{\epsilon\}$.

We lift the interpretation of terms to apply to sequences of terms. Given $\mathbf{u} = [u_i]_{i<n} \in (\mathcal{T}^\gamma(\Sigma))^*$, for $n \geq 0$, and a Σ-model M, $\mathbf{u}^M \stackrel{\text{def}}{=} [u_i^M]_{i<n} \in (\mathcal{U}^\gamma)^*$.

***Definition** 7:* An SFT $A^{\iota/o}$ denotes the *concrete* FT

$$\begin{aligned} [\![A]\!] &\stackrel{\text{def}}{=} (Q_A, q^0_A, F_A, \mathcal{U}^\iota, \mathcal{U}^o, \Delta^{\bar{\epsilon}} \cup \Delta^{\epsilon}), \text{ where} \\ \Delta^{\bar{\epsilon}} &= \{p \xrightarrow{c_\iota^M/\mathbf{u}^M} q \mid p \xrightarrow{\varphi/\mathbf{u}} q \in \Delta^{\bar{\epsilon}}_A,\ M \vDash \varphi\}, \\ \Delta^{\epsilon} &= \{p \xrightarrow{\epsilon/\mathbf{u}^{\mathcal{U}}} q \mid p \xrightarrow{\epsilon/\mathbf{u}} q \in \Delta^{\epsilon}_A\}, \end{aligned}$$

where M ranges over $\{c_\iota\}$-models. Let $T_A \stackrel{\text{def}}{=} T_{[\![A]\!]}$.

***Example** 8:* Consider the SFT A in Example 3. Then $|\Delta^{\bar{\epsilon}}_{[\![A]\!]}| = 2^{16}$. For example, $[\![A]\!]$ has the following transitions:

$$q_0 \xrightarrow{\text{'b'}/[\text{'b'}]} q_0, \quad q_0 \xrightarrow{\text{'ö'}/[\text{'\&'},\text{'\#'},\text{'x'},\text{'F'},\text{'6'}]} q_1 \xrightarrow{\epsilon/[\text{';'}]} q_0$$

So $T_A(\text{“böb”}) = \{\text{“b\&\#xF6;b”}\}$. ■

The following basic property of SFTs is important in the context of algorithm design for SFTs.

***Definition** 8:* An SFT A is *clean* if $IsSat(\varphi)$ for $\varphi \in I_A$.

Other properties of SFTs are defined in terms of their denotations as FTs: SFT A is *deterministic*, resp. *single-valued*, *input-ϵ-free*, if $[\![A]\!]$ is deterministic, resp. single-valued, input-ϵ-free. The following proposition follows from Definitions 5 and 7.

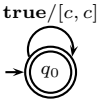
***Proposition** 1: A is deterministic if and only if A is input-ϵ-free and for all $p \xrightarrow{\varphi/\mathbf{u}} q, p \xrightarrow{\psi/\mathbf{v}} r \in \Delta_A$, if $q \neq r$ then $\varphi \wedge \psi$ is unsatisfiable.*

A stronger notion of *input-output determinism* that disallows different outputs for the same input can be characterized similarly and is a useful special case in practical applications. For example the SFT in Example 3 is input-output deterministic. If, in addition, the output sequence is required to be a singleton sequence, this leads to a special case of SFTs that can be called *symbolic Mealy automata*.

A. Alphabets of SFTs

In order to base the definitions of SFTs on classical formal language theory, the concrete alphabets $\mathcal{U}^\iota$ and $\mathcal{U}^o$ need to be *finite*. For example, in Example 3, $|\mathcal{U}^{\text{BV16}}| = 2^{16}$. However, for the symbolic representation the main concern is decidability and complexity of the character theory, rather than *finiteness* of the underlying domain. This point becomes more transparent when we discuss algorithms for SFTs. When considering an input or output sort whose domain is *infinite*, e.g. integers, all algorithms on SFTs remain intact, while SFTs are in this case strictly more expressive than FTs.

***Example** 9:* Consider the sort INT for integers and the following SFT $A^{\text{INT}/\text{INT}}$:



The image of T_A is essentially $\{[n, n] \mid n \in \mathcal{U}^{\text{INT}}\}^*$ that is not accepted by any SFA, since infinitely many states are required, contrary to the image of a finite transduction (also called *rational transduction*) that is a regular language. ■

Example 9 is an instance of the general case when $A^{\iota/o}$ is a clean SFT where both $\mathcal{U}^\iota$ and $\mathcal{U}^o$ are infinite, A has a transition whose output sequence contains c_ι in other than the first output term and denotes infinitely many concrete transitions. In this case the image of T_A cannot be recognized using a finite number of states.

B. Algorithms for SFTs

We here examine several algorithms for SFTs. Following the application to web sanitizers described in Section I-A, the main algorithms of interest are composition and equivalence checking. Our result on equivalence checking establishes that equivalence checking is decidable under fairly mild assumptions, but establishing this requires a technical argument.

Cleaning of an SFT A is elimination of all transitions in $\Delta^{\bar{\epsilon}}_A$ whose guard is unsatisfiable. Thus, cleaning is linear in the number of transitions modulo complexity of satisfiability of formulas in I_A. *In general we assume that SFTs are clean.*

1) Standard algorithms: Many classical automata algorithms, such as the following, are directly applicable to SFTs (and SFAs).

- *Epsilon elimination*: elimination of all $p \xrightarrow{\epsilon/\epsilon} q$ (*epsilon moves*), while linear in the number of states may increase the number of transitions quadratically.
- *Epsilon-loop elimination*: elimination of all paths $p \overset{\epsilon/\epsilon}{\rightsquigarrow} p$ is linear in the number of states and transitions.
- *Unreachable state elimination*: elimination of all states not reachable from the initial state.
- *Deadend elimination*: elimination of all non-initial states from which no final state is reachable.

Both, unreachable state elimination and deadend elimination are linear in the number of states and transitions. All those algorithms are independent of the labels annotating the transitions and work in exactly the same way for automata as well as transducers. Note that epsilon elimination does not imply

complete elimination of Δ^ϵ, which is in general not possible because the class of transductions induced by input-ϵ-free FTs is a proper subclass of all rational transductions. Epsilon-loop elimination is not as widely known, but practically useful in the context of symbolic analysis [8].

2) Composition: We lift the definition of transductions to sets as usual $T(V) \stackrel{\text{def}}{=} \bigcup_{v \in V} T(v)$. Given two transductions T_1 and T_2, the *composition* of T_1 and T_2 is the transduction:

$$T_1 \circ T_2(v) \stackrel{\text{def}}{=} T_2(T_1(v)).$$

A fundamental property of SFTs that makes them attractive for symbolic analysis, is *closure under symbolic composition*.

In [1] it is shown that for any two input-ϵ-free SFTs $A^{\iota/\sigma}$ and $B^{\sigma/o}$, one can effectively construct an input-ϵ-free SFT $A \circ B^{\iota/o}$ such that $T_{A \circ B} = T_A \circ T_B$. The extension of the algorithm to all SFTs is as follows. First, assume that epsilon-transitions have been eliminated, and if there was a transition

$$p \xrightarrow{\epsilon/[u_1,\ldots,u_n]} q, \text{ for some } n \geq 2,$$

new (nonfinal) states $p_1, \ldots, p_{n-1}$, were added and the transition was replaced by the new transitions

$$p \xrightarrow{\epsilon/[u_1]} p_1 \xrightarrow{\epsilon/[u_2]} \cdots \xrightarrow{\epsilon/[u_n]} q$$

Since c_ι does not occur in any u_i, the transformation is well-defined and equivalence preserving.

The composition algorithm [1] is a DFS algorithm where $Q_{A \circ B}$ is constructed as a reachable subset of $Q_A \times Q_B$, starting from (q_A^0, q_B^0). During creation of composed transitions from a reached state (p_1, p_2) the following transitions are added, for example, suppose

$$p_1 \xrightarrow{\varphi/[u_1,u_2]}_A q_1, \quad p_2 \xrightarrow{\psi_1/[v_1]}_B \_ \xrightarrow{\psi_2/[v_2]}_B q_2$$

then, let $\theta_i = \{c \mapsto u_i\}$ for $i = 1, 2$,

$$(p_1, p_2) \xrightarrow{\varphi \wedge \psi_1\theta_1 \wedge \psi_2\theta_2/[v_1\theta_1, v_2\theta_2]}_{A \circ B} (q_1, q_2)$$

The following additional cases are added to the algorithm:

- if $p_2 \xrightarrow{\epsilon/\mathbf{v}}_B q_2$ then $(p_1, p_2) \xrightarrow{\epsilon/\mathbf{v}}_{A \circ B} (p_1, q_2)$,
- if $p_1 \xrightarrow{\epsilon/[u]}_A q_1$, $p_2 \xrightarrow{\varphi/\mathbf{v}}_B q_2$, let $\theta = \{c_\sigma \mapsto u\}$, if $\varphi\theta$ is true then $(p_1, p_2) \xrightarrow{\epsilon/\mathbf{v}\theta}_{A \circ B} (q_1, q_2)$.

There is a subtle point about this extension. While the algorithm in [1] only depends on decidability of satisfiability for keeping $A \circ B$ *clean*, the extension requires evaluating $\varphi\theta$ above for maintaining *correct semantics* of $A \circ B$.

Note also that the transition $(p_1, p_2) \xrightarrow{\epsilon/\mathbf{v}\theta}_{A \circ B} (q_1, q_2)$ is an epsilon transition when $\mathbf{v} = \epsilon$, i.e., epsilon-transitions may reoccur in $A \circ B$ although eliminated from A and B.

***Example** 10:* Consider the following SFTs $A^{\iota/\sigma}$ and $B^{\sigma/o}$ where $a \in \mathcal{U}^\sigma$ and φ is some $\{c_\sigma\}$-formula:

Then $\Delta_{A \circ B} = \{(p_1, p_2) \xrightarrow{\epsilon/\epsilon} (q_1, q_2)\}$ if $\varphi\{c_\sigma \mapsto a\}$ is true, $\Delta_{A \circ B} = \emptyset$, otherwise. ■

$A^{\iota/o} \restriction D^\iota \stackrel{\text{def}}{=}$
1: **let** $Q = \{(q_A^0, q_D^0)\}$
2: **let** $\Delta = \emptyset$
3: **let** S be a stack with initial element (q_A^0, q_D^0)
4: **while** S is nonempty
5: **pop** (p, q) **from** S
6: **foreach** $(p, \varphi, \mathbf{u}, r) \in \Delta_A$ and $(q, \psi, s) \in \Delta_D$
7: **if** $\varphi = \epsilon$ **then**
8: **add** $((p, q), \epsilon, \mathbf{u}, (r, q))$ **to** Δ
9: **if** $(r, q) \notin Q$ **then add** (r, q) **to** Q and **push** (r, q) **to** S
10: **if** $\psi = \epsilon$ **then**
11: **add** $((p, q), \epsilon, \epsilon, (p, s))$ **to** Δ
12: **if** $(p, s) \notin Q$ **then add** (p, s) **to** Q and **push** (p, s) **to** S
13: **if** $\varphi \neq \epsilon$ and $\psi \neq \epsilon$ and $IsSat(\varphi \wedge \psi)$ **then**
14: **add** $((p, q), \varphi \wedge \psi, \mathbf{u}, (r, s))$ **to** Δ
15: **if** $(r, s) \notin Q$ **then add** (r, s) **to** Q and **push** (r, s) **to** S
16: **let** $B = (Q, (q_A^0, q_D^0), Q \cap (F_A \times F_D), \iota, o, \Delta)$
17: eliminate deadends from B
18: **return** B

Fig. 2. Domain restriction algorithm, $A^{\iota/o}$ is an SFT and D^ι is an SFA.

We obtain the following generalization of [1, Theorem 1] by using the above extension.

***Theorem** 1: Let $A_\Gamma^{\iota/\sigma}$ and $B_\Gamma^{\sigma/o}$ be SFTs. Then there is an SFT $A \circ B_\Gamma^{\iota/o}$ such that $T_{A \circ B} = T_A \circ T_B$. If $\mathcal{F}$ is decidable then $A \circ B$ can be constructed effectively.*

3) Domain restriction: Domain restriction is an operation that restricts the input sequences accepted by an SFT $A^{\iota/o}$ with respect to an SFA D^ι. The algorithm for the domain restriction $A \restriction D$ is shown in Fig. 2. The following proposition follows from definitions.

***Proposition** 2: Let $A_\Gamma^{\iota/o}$ be an SFT and D_Γ^ι an SFA. Then $T_{A \restriction D} = T_A \restriction L(D)$. Construction of $A \restriction D$ is effective.*

Note that the satisfiability checking performed in the algorithm (in line 13) ensures that $A \restriction D$ is clean and uses decidability of $\mathcal{F}(c_\iota)$. However, if satisfiability checking is omitted, $T_{A \restriction D}$ is unchanged.

4) Equivalence: Equivalence checking of FTs is undecidable in general [9], and is undecidable already for GSMs. The special case of equivalence checking of single-valued input-ϵ-free SFTs over decidable character background is shown to be decidable in [1]. This result is substantially generalized here (Theorem 2) to *finite-valued* SFTs. This result generalizes also the decidability of equivalence of finite-valued FTs [10], [6], [11]. We use several lemmas to prove Theorem 2.

***Proposition** 3: Let A be a finite-valued SFT such that $T_A(\epsilon) = \emptyset$. There is an input-$\epsilon$-free SFT that is effectively equivalent to A.*

Proof: First, assume that A is clean, has no epsilon-loops, no deadends, and no unreachable states. Second, note that A cannot have *input-epsilon loops* $p \overset{\epsilon/\mathbf{u}}{\rightsquigarrow} p$, $\mathbf{u} \neq \epsilon$, because A is finite-valued.

Let $\Delta_A(p)$ denote the set of all transitions in Δ_A starting from p. Similarly for Δ_A^ϵ and $\Delta_A^{\bar{\epsilon}}$.

The idea is to transform A repeatedly, each time decreasing the number of states p, such that $\Delta_A^\epsilon(p) \neq \emptyset$, while preserving equivalence. The following transformation is repeated until $\Delta_A^\epsilon(p) = \emptyset$ for $p \in Q_A \setminus \{q_A^0\}$.

1) Choose a non-initial state q such that $\Delta_A^\epsilon(q) \neq \emptyset$.

2) For each transition $p \xrightarrow{\varphi/\mathbf{u}} q$ in A add the new transitions

$$\{p \xrightarrow{\varphi/\mathbf{u}\cdot\mathbf{v}} r \mid q \xrightarrow{\epsilon/\mathbf{v}} r \in \Delta^{\epsilon}_A(q)\}$$

to A. Note that $r \neq q$ and if $\varphi = \epsilon$ then $p \neq q$. Also, the semantics of $\mathbf{v}$ is not affected because $\Sigma(\mathbf{v}) = \emptyset$.

3) Remove the transitions $\Delta^{\epsilon}_A(q)$ from A.

Equivalence of the transformed A to the original one follows by using absence of input-epsilon loops and that $q \neq q^0_A$. Eliminate all deadends that were created.

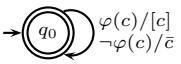
Finally, transitions in $\Delta^{\epsilon}_A(q^0_A)$ are eliminated one by one as follows. Fix $q^0_A \xrightarrow{\epsilon/\mathbf{u}} p$. Since $T_A(\epsilon) = \emptyset$ we know that $p \notin F_A$ and since p is not a deadend $\Delta_A(p) \neq \emptyset$. We know also that $q^0_A \neq p$. Replace the transition $q^0_A \xrightarrow{\epsilon/\mathbf{u}} p$ by

$$\{q^0_A \xrightarrow{\varphi/\mathbf{u}\cdot\mathbf{v}} r \mid p \xrightarrow{\varphi/\mathbf{v}} r \in \Delta_A(p)\}$$

Repeat the step until $\Delta^{\epsilon}_A(q^0_A) = \emptyset$. ■

Note that if A in Proposition 3 is not clean, then more transitions may be added during the transformations but whose guards remain unsatisfiable and the statement remains correct. If A is clean then the transformed SFT is also clean, since guards are not modified.

***Example** 11:* Consider the SFT in Example 3. Input-epsilon elimination yields the following equivalent SFT:



where $\bar{c}$ stands for $[\text{'\&'}, \text{'\#'}, \text{'x'}] \cdot \ulcorner c \urcorner \cdot [\text{';'}]$. ■

We say that a state of an SFT is *relevant* if it is reachable from the initial state and not a deadend.

***Lemma** 1: Let A be a finite-valued SFT. For all u, v, w, and relevant $p \in Q_A$, if $p \overset{u/v}{\rightsquigarrow}_A p$ and $p \overset{u/w}{\rightsquigarrow}_A p$ then $v = w$.*

Proof: Suppose there exist u, v, w, and p such that $p \overset{u/v}{\rightsquigarrow}_A p$ and $p \overset{u/w}{\rightsquigarrow}_A p$ and $v \neq w$. Then for any k, by the pumping lemma, there exist u_1, u_2, v_1 and v_2 and m such that $T_A(u_1 \cdot u^m \cdot u_2) \geq k$, contradicting finite-valuedness of A. ■

***Definition** 9:* An SFT A is a *component* if it is strongly connected and $F_A = \{q^0_A\}$, q^0_A is called the *anchor* of A. An SFT A is a *sequence (of components)* if it consists of disjoint components A_i for $0 \leq i \leq n$ such that $q^0_A = q^0_{A_0}$, $F_A = F_{A_n}$, and there is a single transition $q^0_{A_i} \rightarrow q^0_{A_{i+1}}$ for $0 \leq i < n$.

***Definition** 10:* The *union* of a set $\mathbf{A}$ of SFTs is an SFT with a new initial state and epsilon moves to the initial states of SFTs in $\mathbf{A}$.

***Definition** 11:* An SFT is in *sequence normal form* (*SNF*) if it is a union of pairwise disjoint sequences.

***Lemma** 2: All SFTs have an effectively equivalent SNF.*

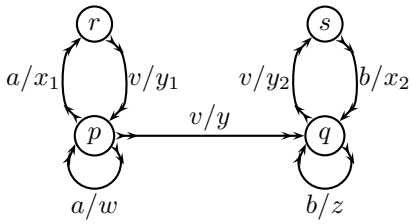
Proof: Let A be an SFT. The construction is standard: the sequences are constructed by considering all loop-free paths from the initial state of A to some final state, possibly creating extra states if a strongly connected component of A is entered and exited through different states. ■

The following is a key lemma used in the proof of decidability of equivalence of finite-valued SFTs below.

***Lemma** 3: Every finite-valued SFT has an effectively equivalent SNF with single-valued sequences.*

Proof: By using Lemma 2 we assume, without loss of generality, that the SFT is a single sequence. Moreover, by using Proposition 3, we assume that the SFT is input-ϵ-free.

We apply the following algorithm to transform the SFT into a set of single-valued sequences. First, note that if the SFT is a single component then it is already single-valued by Lemma 1. Next, we describe the algorithm for the case when the SFT has the form $A\alpha B$, where A and B are two components with anchors p and q and α is a nonempty path $p \rightsquigarrow q$. The case when either A or B have no transitions follows also from Lemma 1. So assume that both A and B contain nonempty paths $p \rightsquigarrow p$ and $q \rightsquigarrow q$. Different outputs may arise by ambiguous parses of an input sequence u through $A\alpha B$ that must allow paths:



and u has the form $a^m \cdot a \cdot v \cdot v \cdot b \cdot b^n$ causing the conflict $x_1 \cdot y_1 \cdot y \cdot z \neq w \cdot y \cdot y_2 \cdot x_2$ in the output. We can rule out the case when $a = b = \epsilon$ or else there exist either unboundedly many different outputs for v^k, by increasing k, contradicting finite-valuedness, or just a single output, independent of the parse, e.g. when $y_1 = y_2 = \epsilon$. So assume $a \neq \epsilon$ (the case $b \neq \epsilon$ is symmetrical). The idea is to resolve the conflict by replacing AvB with $(A \setminus \{av, v\}^*)vB$, $AavvB$ and $AavB$.

In order to detect and resolve such conflicts symbolically, extract the sequence $\bar{\varphi}$ of guards on the path α and search for the corresponding symbolic paths in A and B by checking satisfiability of the corresponding guard sequences for which there exist different output sequences. The maximum length of the paths corresponding to a and b that need to be considered is $|Q_A||Q_B|$.

For example, let $\alpha = p \xrightarrow{\psi/t} q$. And suppose there exist transitions $p \xrightarrow{\varphi_1/u_1} p \xrightarrow{\varphi'_1/u'_1} r \xrightarrow{\psi_1/t_1} p$ in A and transitions $q \xrightarrow{\varphi_2/u_2} q \xrightarrow{\psi_2/t_2} s \xrightarrow{\varphi'_2/u'_2} q$ in B. Let $\theta_i = \{c \mapsto c_i\}$ where c_i is fresh. Assume the following formula is satisfiable:

$$\varphi_1\theta_1 \wedge \psi\theta_2 \wedge \psi_2\theta_3 \wedge \varphi'_2\theta_4 \wedge \varphi'_1\theta_1 \wedge \psi_1\theta_2 \wedge \psi\theta_3 \wedge \varphi_2\theta_4$$
$$\wedge u_1\theta_1 \cdot t\theta_2 \cdot t_2\theta_3 \cdot u'_2\theta_4 \neq u'_1\theta_1 \cdot t_1\theta_2 \cdot t\theta_3 \cdot u_2\theta_4$$

Then there exist u with different outputs. Construct the SFA D for the guard sequences $\{[\varphi_1 \wedge \varphi'_1], [\varphi_1 \wedge \varphi'_1, \psi_1]\}^*$, in particular accepting $\{a, a \cdot v\}^*$ as above. Let $\bar{D}$ be the complement of D. Let $A' = A|\bar{D}$ (thus removing the conflicts from A). Let α_1 be the path $p \xrightarrow{\varphi_1 \wedge \varphi'_1/u_1} p_1 \xrightarrow{\psi/t} q$ and let α_2 be the path $p \xrightarrow{\varphi_1 \wedge \varphi'_1/u'_1} r_2 \xrightarrow{\psi_1/t_1} p_2 \xrightarrow{\psi/t} q$. Now replace $A\alpha B$ with the SFTs $A'\alpha B$, $A\alpha_1 B$ and $A\alpha_2 B$. Note that $A'\alpha B$ is now single-valued and can be transformed to SNF. It follows that the union of the new sequences is equivalent to $A\alpha B$. Repeat

the transformation on $A\alpha_1 B$ and $A\alpha_2 B$. Termination follows from that both have fewer nonequivalent conflicts remaining and that the length of paths α causing conflicts is effectively bounded by the size of the original SFT. The proof can be generalized to the case of sequences of arbitrary length. ■

Below we make use of the following pumping lemma about equations over finite sequences of elements.

***Lemma** 4: For all $u_1, u_2, v_1, v_2, w_1, w_2, z_1, z_2$, if*

$$\begin{aligned} u_1 \cdot u_2 &= v_1 \cdot v_2, \\ u_1 \cdot w_1 \cdot u_2 &= v_1 \cdot z_1 \cdot v_2, \\ u_1 \cdot w_2 \cdot u_2 &= v_1 \cdot z_2 \cdot v_2 \end{aligned}$$

then $u_1 \cdot w_1 \cdot w_2 \cdot u_2 = v_1 \cdot z_1 \cdot z_2 \cdot v_2$.

We also need the following generalization of [1, Theorem 2] where an algorithm is given for deciding equivalence of single-valued input-ϵ-free SFTs. The algorithm in [1] does not, however, generalize to the case of simultaneous equivalence between multiple SFTs, that is needed below. For a single-valued SFT A write $A(u) = v$ when $T_A(u) = \{v\}$.

***Lemma** 5: Let $A^{\iota/o}, B_1^{\iota/o}, \ldots, B_k^{\iota/o}$ be input-ϵ-free single-valued SFTs for some $k \geq 1$ then the problem $\exists x \, (\bigwedge_{i=1}^{k} T_A(x) \neq T_{B_i}(x))$ is decidable if $\mathcal{F}$ is decidable.*

Proof: Case $k = 1$ is [1, Theorem 2]. We prove the case for $k = 2$. Generalization to $k > 2$ is technically more involved but straightforward. Let $B = B_1$, $C = B_2$. We only need to consider inputs in $L = L(\mathbf{d}(A)) \cap L(\mathbf{d}(B)) \cap L(\mathbf{d}(C))$. For example, if $u \in L(\mathbf{d}(A)) \setminus L(\mathbf{d}(B))$ and $u \in L(\mathbf{d}(C))$ then $T_A(u) \neq T_B(u)$ and the problem reduces to equivalence of $A{\restriction}\overline{\mathbf{d}(B)}$ and C, where the construction of $A{\restriction}\overline{\mathbf{d}(B)}$ is effective. The other cases are similar.

For the case L construct the product $D = A \times B \times C$ that has states $Q_A \times Q_B \times Q_C$ and 3-output-transitions

$$\begin{aligned} &(p, q, r) \xrightarrow{\varphi \wedge \psi_1 \wedge \psi_2 / (u,v,w)} (p', q', r'), \\ &\text{for} \quad p \xrightarrow{\varphi/u}_A p', \quad q \xrightarrow{\psi_1/v}_B q', \quad r \xrightarrow{\psi_2/w}_C r' \end{aligned}$$

such that $IsSat(\varphi \wedge \psi_1 \wedge \psi_2)$. Note that $L(\mathbf{d}(D)) = L$. The unreachable states and the deadends are eliminated from D.

$$D(u) \stackrel{\text{def}}{=} (A(u), B(u), C(u))$$

Let $p_0 = q_A^0$, $q_0 = q_B^0$ and $r_0 = q_C^0$. We write s_0 for (p_0, q_0, r_0) and s_f for some $(p_f, q_f, r_f) \in F_A \times F_B \times F_C$.

Given $u \in L$ and $D(u) = (\mathbf{a}, \mathbf{b}, \mathbf{c})$, there are two (possibly overlapping) cases for a *B-conflict* $\mathbf{a} \neq \mathbf{b}$ (symmetrically for a *C-conflict* $\mathbf{a} \neq \mathbf{c}$):

1) there is a *B-length-conflict*: $|\mathbf{a}| \neq |\mathbf{b}|$, or
2) there is a *B-character-conflict*: for some i, $\mathbf{a}[i] \neq \mathbf{b}[i]$.

We say that a state $s \in Q_D$ is a *B-length-conflict-state* if there exists a *simple* loop (a loop without nested loops) $s \overset{u/(v,w,\_)}{\rightsquigarrow} s$ such that $|v| \neq |w|$. The statements below make implicit use of the assumption that D contains no unreachable states and no deadends.

(*) There are two ways how a B-length-conflict can arise.

1.a) There exists a B-length-conflict state s in D.

1.b) There exists a loopfree path $s_0 \overset{u/(v,w,\_)}{\rightsquigarrow} s_f$ such that $|v| \neq |w|$.

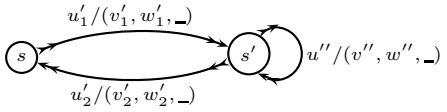
Proof of (*): We show that cases 1.a and 1.b are exhaustive. Consider any $u \in L$ such that $D(u) = (v, w, \_)$ and $|v| \neq |w|$ and suppose 1.b is false. Then there must exist $u_1, u', u_2, v_1, v', v_2, w_1, w', w_2$ such that

$$u = u_1 \cdot u' \cdot u_2, \; v = v_1 \cdot v' \cdot v_2, \; w = w_1 \cdot w' \cdot w_2,$$

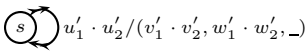
and a loop $s \overset{u'/(v',w',\_)}{\rightsquigarrow} s$ where $|v'| \neq |w'|$, or else $|v| = |w|$ since 1.b is false. Now suppose the loop is not simple, then there exist $u_1', u'', u_2', v_1', v'', v_2', w_1', w'', w_2'$ such that

$$u' = u_1' \cdot u'' \cdot u_2', \; v' = v_1' \cdot v'' \cdot v_2', \; w' = w_1' \cdot w'' \cdot w_2',$$

and a state s',



If $|v''| = |w''|$ then $|v_1' \cdot v_2'| \neq |w_1' \cdot w_2'|$ and

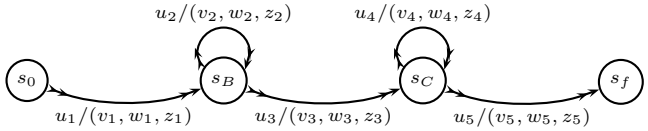


and repeat the argument for the shorter path if it is not simple. Otherwise, if $|v''| \neq |w''|$ and the loop through s' is not simple apply the argument for s'. □

Note that the problems of deciding 1.a and 1.b are decidable. In order to decide if a state s is a B-length-conflict-state consider all the possible simple loops $s \rightsquigarrow s$: for each such path check if the outputs lengths for A and B are different. There are finitely many such paths. Similarly for 1.a.

Next, we proceed by case analysis, showing that we can effectively decide all the different combinations of possible B-conflicts and C-conflicts that can arise. We write B.1.a for the case when there exists a B-length-conflict-state, similarly for the other cases.

Case (B.1.a, C.1.a): Check if there exist s_A and s_B such that s_B is a B-length-conflict-state and s_C is a C-length-conflict state and $s_B \rightsquigarrow s_C$. Then there exists a path

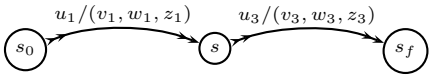


where $|v_2| \neq |w_2|$ and $|v_4| \neq |z_4|$. It follows that there exist m and n such that

$$\begin{aligned} |v_1 \cdot v_2^m \cdot v_3 \cdot v_4^n \cdot v_5| &\neq |w_1 \cdot w_2^m \cdot w_3 \cdot w_4^n \cdot w_5| \\ |v_1 \cdot v_2^m \cdot v_3 \cdot v_4^n \cdot v_5| &\neq |z_1 \cdot z_2^m \cdot z_3 \cdot z_4^n \cdot z_5| \end{aligned}$$

Thus there exists $u = u_1 \cdot u_2^m \cdot u_3 \cdot u_4^n \cdot u_5 \in L$ such that $D(u)$ is a B-conflict and a C-conflict. There are finitely many such combinations. The case $s_C \rightsquigarrow s_B$ is symmetrical. No other simultaneous combinations of (B.1.a, C.1.a) are possible.

Case (B.1.a, C.1.b): Check if there exists a B-length-conflict-state s and a looopfree path $s_0 \rightsquigarrow s \rightsquigarrow s_f$ that causes a C-length conflict, i.e., there exists a path



such that $|v_1 \cdot v_3| \neq |z_1 \cdot z_3|$. There exists u_2 such that $s \overset{u_2/(v_2,w_2,z_2)}{\rightsquigarrow} s$ where $|v_2| \neq |w_2|$. Thus, there exists m such that

$$\begin{array}{rcl} |v_1 \cdot v_2^m \cdot v_3| & \neq & |w_1 \cdot w_2^m \cdot w_3| \\ |v_1 \cdot v_2^m \cdot v_3| & \neq & |z_1 \cdot z_2^m \cdot z_3| \end{array}$$

Thus there exists $u = u_1 \cdot u_2^m \cdot u_3 \in L$ such that $D(u)$ is a B-conflict and a C-conflict. There are finitely many such combinations. No other simultaenous combinations of (B.1.a, C.1.b) are possible. The case (B.1.b, C.1.a) is symmetrical.

Case (B.1.b, C.1.b): Check if there exists a loopfree path $s_0 \rightsquigarrow s_f$ that causes both a B-length-conflict and and a C-length-conflict. Then there exists u such that $D(u)$ is a B-conflict and a C-conflict. There are finitely many such paths and no other simultaneous occurrences of (B.1.b, C.1.b) are possible.

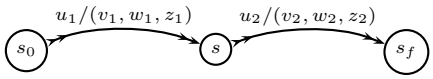
Case (B.2, C.1): Assume, by previous cases, that (B.1, C.1) is not possible. Let ℓ be the length of the longest possible output from either A, B or C on any loopfree path. Clearly, ℓ can be computed effectively. Suppose there exists a C-length-conflict-state s. Consider all paths

$$\rho_m : \quad s_0 \rightsquigarrow (s \rightsquigarrow s)^m \rightsquigarrow s_f, \quad m \leq 2\ell$$

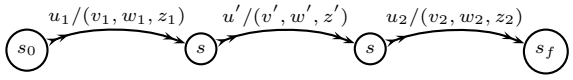
Since B.1.a is not possible, we know that for all loops $s \rightsquigarrow s$ the A-output and the B-output have the same length. For each ρ_m check if a simultaneous B-character-conflict and C-length-conflict exists.

If no such simultaneous conflicts exist it follows from the following argument that no such simultaneous conflicts exist in any longer paths. We may assume that all such loops have nonempty A (and thus B) outputs, since empty outputs neither cause nor remove any character conflicts.

- Suppose some ρ_m, $\ell \leq m < 2\ell$, contains a B-character conflict. Then, by choice of ℓ and since all the A and B-outputs are nonempty, there exist u_i, v_i, w_i, z_i, $1 \leq i \leq 2$, such that

  and either the character conflict occurs in the prefixes of v_1, w_1 or in the suffixes of v_2, w_2 (i.e., the conflict is not in the overlap). Thus, the B-charater-conflict remains in

  for any $s \overset{u'/(v',w',z')}{\rightsquigarrow} s$, where $|v'| = |w'|$ and $|v'| \neq |z'|$. We now have a contradiction, because either ρ_m or ρ_{m+1} must cause a simultaneous C-length-conflict, i.e., either $|v_1 \cdot v_2| \neq |z_1 \cdot z_2|$ or $|v_1 \cdot v' \cdot v_2| \neq |z_1 \cdot z' \cdot z_2|$.

- Thus, in particular, ρ_ℓ and $\rho_{\ell+1}$ do not cause any B-character-conflicts. It now follows from Lemma 4 that for all $m \geq \ell$, in ρ_m the outputs of A and B will be equal.

There are finitely many symbolic paths in D that correspond to the concrete ρ_m's above. For each such path construct a formula in $\mathcal{F}$ that is satisfiable iff a B-character-conflict exists. For example, for a symbolic path

$$s_0 \xrightarrow{\varphi_1/(v_1,w_1,\_)} s \xrightarrow{\varphi_2/(v_2,w_2,\_)} s_f,$$

given substitution $\theta_i = \{c_\iota \mapsto c_i\}$ where c_i is a fresh uninterpreted constant the formula is:

$$\varphi_1\theta_1 \wedge \varphi_2\theta_2 \wedge v_1\theta_1 \cdot v_2\theta_2 \neq w_1\theta_1 \cdot w_2\theta_2$$

The case C.1.b is covered by considering all loopfree paths. It follows that the case (B.2, C.1) is decidable. The case (B.1, C.2) is symmetrical.

Case (B.2, C.2): Assume, by previous cases, that (B.1, C.1), (B.1, C.2) and (C.1, B.2) are not possible. In other words, for all loops, the lengths of outputs from A, B and C are equal and the total lengths of outputs are equal. Let ℓ be as above and consider all paths:

$$\rho_m : \quad s_0 \rightsquigarrow (s \rightsquigarrow s)^m \rightsquigarrow s_f, \quad m \leq 2\ell$$

where $s \rightsquigarrow s$ is a simple loop and check if a simultaneous character conflict exists. As above, all such paths correspond to finitely many symbolic paths in D. For example, for a symbolic path

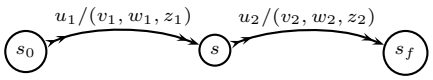
$$s_0 \xrightarrow{\varphi_1/(v_1,w_1,z_1)} s \xrightarrow{\varphi_2/(v_2,w_2,z_2)} s_f,$$

given substitution $\theta_i = \{c_\iota \mapsto c_i\}$ where c_i is a fresh uninterpreted constant the formula is:

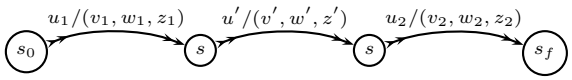
$$\varphi_1\theta_1 \wedge \varphi_2\theta_2 \wedge v_1\theta_1 \cdot v_2\theta_2 \neq w_1\theta_1 \cdot w_2\theta_2 \wedge v_1\theta_1 \cdot v_2\theta_2 \neq z_1\theta_1 \cdot z_2\theta_2$$

the following argument shows that, if no such simultaneous character conflicts are detected then there are none in any longer paths. The argument is similar to the case (B.2, C.1). As above, we may assume that all such loops have nonempty A (and thus B and C) outputs, since empty outputs neither cause nor remove any character conflicts.

- Suppose for some ρ_m, ρ_n, $\ell \leq m < n < 2\ell$, ρ_m causes a B-character-conflict and ρ_n causes a C-character-conflict. Then, by choice of ℓ and since all the A and B-outputs are nonempty, there exist u_i, v_i, w_i, z_i, $1 \leq i \leq 2$, such that

  and either both character conflicts occur in the prefixes of (v_1, w_1, z_1) or in the suffixes of (v_2, w_2, z_2), or one occurs in the prefixes of (v_1, w_1, z_1) and the other in the suffixes of (v_2, w_2, z_2) (i.e., the conflicts are not in the overlap). Then both conflicts remain in

for any $s \overset{u'/(v',w',z')}{\rightsquigarrow} s$, where $|v'| = |w'| = |z'|$. In particular both conflicts would be in some $\rho_{2\ell}$, contradicting the assumption.

- Thus, in particular, ρ_ℓ and $\rho_{\ell+1}$ contains, without loss of generality, no B-character conflicts. It now follows from Lemma 4 that for all $m \geq \ell$, in ρ_m the outputs of A and B will be equal.

One can show that the above cases are exhaustive. Decidability follows for $k = 2$. ∎

Note that the proof above uses arbitrarily many uninterpreted constants of sort ι, i.e., it assumes decidability of $\mathcal{F}$ while the proof of the case for $k = 1$ needs only two distinct constants of sort ι and needs decidability of $\mathcal{F}(\{c : \iota, d : \iota\})$

***Theorem** 2: Equivalence of finite-valued SFTs is decidable provided that $\mathcal{F}$ is decidable.*

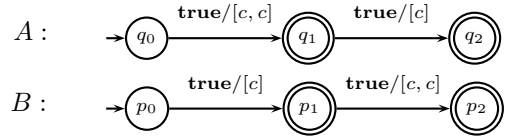
Proof: Let A and B be finite-valued SFTs. Assume $D = L(\mathbf{d}(A)) = L(\mathbf{d}(B))$, or else A and B are not equivalent. By using Lemma 3 assume A and B are on SNF containing single-valued SFTs. Assume, without loss of generality that A and B do not accept the empty string and that all component sequences in A and B are input-ϵ-free. To decide $A \cong B$, we check that for all $v \in D$, $T_A(v) \subseteq T_B(v)$ and $T_B(v) \subseteq T_A(v)$. Conversely, $A \ncong B$ iff either (1) or (2) holds for some $v \in D$:

1) for some A_1 in A and all B_1 in B, $T_{A_1}(v) \neq T_{B_1}(v)$.
2) for some B_1 in B and all A_1 in A, $T_{A_1}(v) \neq T_{B_1}(v)$.

Decidability of (1) and (2) follows now from Lemma 5. ∎

Although we can check equivalence, there is no direct way to generate witnesses when equivalence does not hold (even for the single-valued case). For this, intersection of SFTs would be useful for separating the common answers in the single-valued case Intersection of two SFTs $A^{\iota/o}$ and $B^{\iota/o}$, if one exists, is an SFT $A \cap B$ such that $T_{A\cap B} = T_A \cap T_B$. However, SFTs are not closed under intersection, already in the case of input-ϵ-free single-valued SFTs as illustrated by the following example.

***Example** 12:* Consider SFTs $A^{\sigma/\sigma}$ and $B^{\sigma/\sigma}$:



Then $T_A \cap T_B = \{[c, c] \mapsto [c, c, c] \mid c \in \mathcal{U}^\sigma\}$ is not expressible as an SFT when $\mathcal{U}^\sigma$ is infinite, and is very large when $|\mathcal{U}^\sigma|$ is large. ∎

IV. Extension with registers

In this section we introduce the extension of SFTs with registers, and generalize some of the results for SFTs. By a *register* we mean an uninterpreted constant.

***Definition** 12:* A *Symbolic Finite Transducer with n Registers (SFTR^n)* is an extension of an SFT $A^{\iota/o}$ with n distinct *registers* $r_i : \sigma_i$, $r_i \neq c_\iota$, for $0 \leq i < n$. Let $R = \{r_i\}_{i<n}$. Transition labels in $\Delta_A^{\bar{\epsilon}}$ may, in addition to c_ι, contain registers. Each transition is associated with an *update* that is a substitution ρ for R such that $r_i\rho \in \mathcal{T}^{\sigma_i}(R \cup \{c_\sigma\})$. Input-$\epsilon$-transitions form a separate set of Δ_A^{ϵ} of transitions whose labels do not contain c_σ. The *initial update* R^0 is an R-model. We write $q \xrightarrow{\varphi/v;\rho} p$ for the symbolic transition of A where ρ is the associated update.

The *concrete transducer* $[\![A]\!]$ of A is defined similarly to the case of SFTs, except that the number of states in $[\![A]\!]$ is infinite in general. The states of $[\![A]\!]$ are all pairs (q, M) where M is an R_A-model. The initial state is (q_A^0, R_A^0). If (q, M) is a state, $q \xrightarrow{\varphi/v;\rho}_A p$, and there exists $N = M \cup \{c_\sigma \mapsto a\}$ such that $N \vDash \varphi$ then

$$(q, M) \xrightarrow{a/v^N}_{[\![A]\!]} (p, \{r \mapsto r\rho^N \mid r \in R_A\}).$$

The definition of T_A is lifted from the finite case.

One can show that SFTRs are closed under composition. The algorithm is a DFS algorithm similar to the case of SFTs. Let A be a SFTR^n and let B be an SFTR^m such that $R_A \cap R_B = \emptyset$. Then $A \circ B$ is an SFTR^{n+m} that is defined as follows.

$$\begin{aligned} Q_{A\circ B} &= Q_A \times Q_B, \\ F_{A\circ B} &= F_A \times F_B, \\ R_{A\circ B} &= R_A \cup R_B, \\ R^0_{A\circ B} &= R^0_A \cup R^0_B. \end{aligned}$$

Let $(p_1, p_2) \in Q_{A\circ B}$. The following transitions are added from (p_1, p_2), for example, suppose

$$p_1 \xrightarrow{\varphi/[u_1,u_2];\rho}_A q_1, \quad p_2 \xrightarrow{\psi_1/\mathbf{v_1};\rho_1}_B p' \xrightarrow{\psi_2/\mathbf{v_2};\rho_2}_B q_2$$

Let $\theta_1 = \{c \mapsto u_1\}$ be the substitution for the input character read from p_2 and let $\theta_2 = \{c \mapsto u_2\} \cup \rho_1\theta_1$ be the substitution for the input character read from p' as well as the composed register update after the update from p_2. Then the composed transition is

$$(p_1, p_2) \xrightarrow{\varphi\wedge\psi_1\theta_1\wedge\psi_2\theta_2/\mathbf{v_1}\theta_1\cdot\mathbf{v_2}\theta_2;\rho\cup\rho_2\theta_2}_{A\circ B} (q_1, q_2)$$

The generalization to arbitrary lengths of outputs from p_1 is obvious. The case when there is an input-ϵ transition from p_1 is similar, in this case we get input-ϵ transitions in the composition whose guards involve registers only.

***Theorem** 3: Given SFTR^m $A^{\iota/\sigma}$ and SFTR^n $B^{\sigma/o}$ there is an SFTR^{m+n} $A \circ B^{\iota/o}$ such that $T_{A\circ B} = T_A \circ T_B$. The construction of $A \circ B$ is effective.*

Although general reachability at the level $[\![A]\!]$ is undecidable in this case, *cleaning* of SFTRs is local as it requires only satisfiability checks of the guards.

A practically very useful fragment of SFTRs is the case when each output term and each update term is either a concrete value, an input character, or a register. We say the SFTR has *simple updates*. For example, there is a whole class of sanitizers that only filter input characters and either keep, remove, replace, or append them with other characters, but do not perform bit-level transformations on the character itself. It is easy to see that this fragment is closed under composition as well, since it is preserved by substitutions.

SFTRs are strictly more expressive than SFTs with simple updates as illustrated by the following example.

***Example** 13:* Consider $T = \{[c,c] \mapsto [c,c,c] \mid c \in \mathcal{U}^\sigma\}$ from Example 12. Let $A^{\sigma/\sigma}$ be an SFTR with a register $r:\sigma$ and transitions

$$q_A^0 \xrightarrow{c=c/[c];r\mapsto c} q_1 \xrightarrow{c=r/[c,c];r\mapsto r} q^{\text{final}}$$

Then $T_A = T$. ∎

For SFTRs with simple updates we can omit guard formulas from input-ϵ transitions. For SFTRs with simple updates, such guards would only depend on the registers, and updates to registers on input-ϵ transitions can only depend on the current registers as well. We can therefore consider all ϵ paths existing input-non-ϵ transition and fold the guards from the ϵ paths into a corresponding number of versions of the input-non-ϵ transition.

Moreover, we extend the result of decidability of single-valued SFTs to the case when all update terms are simple.

***Theorem** 4: Equivalence of single-valued SFTRs with simple updates is decidable provided that $\mathcal{F}$ is decidable.*

Proof: Notice first that we establish the theorem for *single-valued* SFTRs, not general *finite-valued* SFTRs, so we can apply an argument that uses ideas from [1]. Let A and B be two single-valued SFTRs that we wish to check equivalence for. We will perform a depth first exploration of the product of A and B starting from their initial states. The states examined during the product construction are of the form $(p,q,\mathbf{u},\mathbf{v},\varphi,\psi)$, where $p \in Q_A, q \in Q_B$; either $\mathbf{u}$ or $\mathbf{v}$ is empty and accumulates the output generated by A (B) that extends the common output sequence of B (A); and φ resp. ψ are the conjunction of guards from A resp. B accumulated on the trace leading to the current state. The use of guard formulas is specific to SFTRs where the transition can depend on the values in registers. The idea used in [1], that does not require tracking guard formulas, is to explore the product until either (1) a state is visited with different outputs; or (2) the same pair $(p,q,\mathbf{u},\mathbf{v})$ is revisited with a longer output for $\mathbf{u}$ ($\mathbf{v}$); or (3) the full product has been explored without hitting cases (1) or (2). Equivalence of A and B then amounts to reaching condition (3).

With SFTRs, transitions depend on the contents of registers and these have to be tracked during a product exploration. So the states examined during the product construction contain φ and ψ with the path conditions. There is an infinite number of such traces of course, but let us consider isomorphic sets of such guards. Two sets of instantiated guards are isomorphic if there is a renaming of the characters (that are treated as fresh variables) that equates the two sets. Such sets are equi-satisfiable and we don't need to consider more than one such equivalence class on a path. The number of isomorphism classes for SFTRs with simple updates is finite, so the states examined during the product construction is also finite.

Also note that we can extend the definition of register update to allow updating a register with a function of the input character as long as the domain and range of the function is finite. ∎

Equivalence for general SFTRs is not decidable since they allow registers to be updated by arbitrary functions.

V. A Semi-Decision Procedure for Non-Equivalence

In the following we develop a theory for SFTs that uses the theory of algebraic datatypes [12]. In particular, it provides a semi-decision algorithm for the counterexample generation problem. In practice it uses state-of-the-art SMT solvers that support algebraic datatypes in the background, such as Z3 [2]. The approach is close in spirit to similar encodings for symbolic regular automata. A distinguishing feature of this encoding is that it applies to general SFTRs that are free of ϵ-loops. We provide the encoding for SFTs. The extension to SFTRs is straight-forward. The detailed encoding for SFTs can be found in the appendix A; we here summarize the highlights.

We assume a sort $\text{LIST}\langle\alpha\rangle$ of lists over sort α and decision procedures for the quantifier-free theory of lists. Let us introduce a theory that is used to characterize the transition relation of an SFT A. First, we define a *non-epsilon step formula*, where θ is the substitution $\{c_\iota \mapsto hd(x)\}$ and x and y are list variables,

$$\begin{aligned}&Step_{(p,\varphi,[u_0,\ldots,u_{n-1}],q)}(x,y) \stackrel{\text{def}}{=}\\ &x \neq \varepsilon \wedge \varphi\theta \wedge \textstyle\bigwedge_{k<n}(tl^{(k)}(y) \neq \varepsilon \wedge hd(tl^{(k)}(y)) = u_k\theta)\end{aligned}$$

stating that the first element in x satisfies the condition φ and that the k'th element of y is equal to $u_k\theta$ for $k < n$. We define an *input-epsilon step formula* similarly.

***Definition** 13:* Let

$$\Sigma_A \stackrel{\text{def}}{=} \{Acc_p : \text{LIST}\langle\iota\rangle \times \text{LIST}\langle o\rangle \to \text{BOOL}\}_{p\in Q_A}$$

be a set of fixed uninterpreted relation symbols. Let $x : \text{LIST}\langle\iota\rangle$ and $y : \text{LIST}\langle o\rangle$ be variables. For all $p \in Q_A$ define:

$$\begin{aligned}ax_p \stackrel{\text{def}}{=} \forall x\, y\, \big(&Acc_p(x,y) \Leftrightarrow \big((p{\in}F_A \wedge x = \varepsilon \wedge y = \varepsilon) \vee\\ &\textstyle\bigvee_{(p,\varphi,u,q)\in\Delta_A}(Step_{(p,\varphi,u,q)}(x,y) \wedge\\ &\qquad Acc_q(tl^{(|\varphi|)}(x), tl^{(|u|)}(y)))\big)\big)\end{aligned}$$

Let $Acc_A \stackrel{\text{def}}{=} Acc_{q_A^0}$, $Th(A) \stackrel{\text{def}}{=} \{ax_p \mid p \in Q_A\}$.

The main property of the encoding can be summarized as follows.

***Theorem** 5: $Th(A)$ is consistent. Moreover, if A is ϵ-loop-free, then there is a unique Σ_A-model $M_A \vDash Th(A)$ such that $w \in T_A(v)$ if and only if $M_A \vDash Acc_A(v,w)$.*

The axioms can be used in the context of theorem provers that produce ground instantiations: for each length ℓ, there is a finite set of instances of $Th(A)$, such that $M_A, w \vDash Acc_A(v,w) \wedge |v| \leq \ell$ iff $w \in T_A(v), |v|^M \leq \ell$. The high-level procedure is illustrated in Fig. 3. The reason why negation of the acceptors predicates works in this procedure is that the transducers are epsilon-loop-free, implying that the axioms are well-founded. (Similar transformation would in general not work for arbitrary transition systems.)

The encoding of SFTRs is a straight-forward generalization: it requires the predicates Acc_A to take additional arguments corresponding to registers, and it requires the transition relations to encode updates and outputs from registers. Theorem 5 remains intact for this generalization.

Witness$^{\not\supseteq}(A^{\iota/o}, B^{\iota/o}) \stackrel{\text{def}}{=}$
1: **let** $\Sigma_A \cup \Sigma_B$ be new uninterpreted function symbols
2: **assert** $Th(A) \cup Th(B)$ as auxiliary axioms
3: **let** $y : \text{LIST}\langle o\rangle$ be a new uninterpreted constant
4: **let** $x = \varepsilon : \text{LIST}\langle \iota\rangle$
5: **loop**
6: **if** exists $M \vDash Acc_A(x, y) \wedge \neg Acc_B(x, y)$, or
7: $M \vDash Acc_B(x, y) \wedge \neg Acc_A(x, y)$ **then**
8: **return** (x^M, y^M)
9: **else**
10: **let** $x := [c \,|\, x]$ where $c : \iota$ is a new uninterpreted constant

Fig. 3. For input-ϵ-free SFTs A and B such that $A \not\supseteq B$, generates a witness (v, c) for $c \in T_A(v) \setminus T_B(v) \cup T_B(v) \setminus T_A(v)$.

VI. Related Work

Relationships between logics and automata have been investigated for over half a century, a comprehensive study is presented in [13]. In recent years there has been considerable interest in automata over infinite languages, starting with the work on *finite memory automata* [14], also called *register automata*, where, besides a finite number of states, the automaton has a finite number of registers taking values in an infinite domain, a register automaton has thus infinitely many extended states, called configurations, consisting of a finite state component together with a value assignment for the registers. Similar extensions of finite automata, called *extended finite state machines*, have been studied and used in the context of protocol testing [15]. Finite words over an infinite alphabet are usually called *data words* in the literature.

Besides register automata, various other automata models over data words have been studied in the literature [16], such as *pebble automata* [17] that use markers to annotate positions in a data word, and *data automata* [18] that are two-way automata used to establish (un)decidability results on two-variable logic with equality over data words. Several characterizations of logics with respect to different models of data word automata are studied in [19]. This line of work focuses on fundamental questions about definability, decidability, complexity, and expressiveness on classes of automata on one hand and fragments of logic on the other hand.

A different line of work on automata with infinite alphabets introduces *lattice automata* [20] that are finite state automata whose transitions are labeled by elements of an atomic lattice. The motivation for the work comes from verification of symbolic communicating machines. Unlike register automata, pebble automata or data automata, lattice automata provide a way to incorporate a particular interpretation for data through the operations. An advantage of the approach is that labels can be over-approximated to equivalence classes and mapped to a finite alphabet leading to classical finite automata.

Finite state automata with arbitrary predicates over symbols, called *predicate-augmented finite state recognizers*, or *symbolic finite automata* (SFAs) in the current paper, were first used in the context of natural language processing [21], where predicates provide a natural way to express phonological generalizations such as fricative and nasal that are more common in computational phonology than individual phonemes. It is pointed out in [21] that the finiteness assumption of the alphabet is not needed. To the contrary, an infinite or unbounded alphabet is often needed for robustness of parsing algorithms. A similar extension of regular languages is used in [22]. Compared to lattice automata, SFAs are defined modulo a given character theory and avoid the problem that an element of a lattice may not have a complement.

While the work in [21] views symbolic automata as a "fairly trivial" extension of finite automata, the more fundamental question is how classical automata algorithms, that use alphabet symbols explicitly, can be extended to the symbolic case. In particular symbolic complementation by a combinatorial optimization problem called *minterm generation* [23] leads to significant speedups compared to state-of-the-art automata algorithm implementations.

The work in [21] introduces also an extension to finite state transducers called a *predicate-augmented finite state transducer* (pfst). A pfst has two kinds of transitions: 1) $p \xrightarrow{\varphi/\psi} q$ where φ and ψ are character predicates or ϵ, or 2) $p \xrightarrow{c/c} q$. In the first case the symbolic transition corresponds to all concrete transitions $p \xrightarrow{a/b} q$ such that $\varphi(a)$ and $\psi(b)$ are true, the second case corresponds to *identity* transitions $p \xrightarrow{a/a} q$ for all characters a. A pfst is not expressive enough for describing an SFT. Besides identities, it is not possible to establish functional dependencies from input to output that are needed for example to encode sanitizers such as the one in Fig. 1. Also, the input and output alphabets are the same, thus it is not possible to describe transducers with different input and output sorts, which is important for various SFT constructions such as the one mentioned in Section I-A.

Besides the work on BEK [1], finite state transducers have been used for dynamic and static analysis to validate sanitization functions in web applications in [24], by an over-approximation of the strings accepted by the sanitizer using static analysis of existing PHP code. Other security analysis of PHP code, e.g., SQL injection attacks, use string analyzers to obtain over-approximations (in form of context free grammars) of the HTML output by a server [25], [26], [27].

Our work is complementary to previous efforts in extending SMT solvers to understand the theory of strings. HAMPI [28] and Kaluza [29] extend the STP solver to handle equations over strings and equations with multiple variables. The work in [30] shows how to solve subset constraints on regular languages. In contrast, we show how to combine any of these solvers with SFTs whose edges can take symbolic values in the theories understood by the solver. Axiomatization of SFAs using a background of lists was initially introduced in [31] and is used to provide integrated support for REGEX-constraints in parameterized unit testing of .NET code [32], [33], and to provide support for LIKE-expressions in SQL query analysis [34]. Axiomatization of SFAs was extended to symbolic PDAs in [8]. The axiomatic theory of SFTs presented here is to our knowledge new.

VII. Conclusion

We have introduced and studied Symbolic Finite Transducers and Symbolic Finite Transducers with Registers. The main algorithms of interest for applications, such as web sanitizers, are composition and equivalence checking. Equivalence checking is in general undecidable and we revert to a semi-decision procedure presented in Section V. However, the cause for undecidability is subtle, and this paper identifies a boundary based on whether the transducer is finite-valued (and satisfiability of guard formulas is decidable). SFTRs add extra expressibility, but under the simple update assumption, also maintain decidability for single-valued SFTRs. We conjecture that equivalence of SFTRs for finite-valued SFTRs with simple updates is also decidable.

The symbolic representation of transducers is both convenient for applications and allows for succinct representations. Basic automata algorithms lift in many cases in a straightforward way to this representation, and it allows leveraging state-of-the-art theorem proving technology for analyzing the automata.

References

[1] P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes, "Bek: Modeling imperative string operations with symbolic transducers," Microsoft Research, MSR-TR-2010-154, November 2010, subm. PLDI'11.

[2] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *TACAS'08*, ser. LNCS. Springer, 2008.

[3] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[4] W. Hodges, *Model theory*. Cambridge Univ. Press, 1995.

[5] S. Yu, "Regular languages," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Springer, 1997, vol. 1, pp. 41–110.

[6] A. Demers, C. Keleman, and B. Reusch, "On some decidable properties of finite state translations," *Acta Informatica*, vol. 17, pp. 349–364, 1982.

[7] M. A. Harrison, *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[8] M. Veanes, N. Bjørner, and L. de Moura, "Symbolic automata constraint solving," in *LPAR-17*, ser. LNCS, vol. 6397, 2010, pp. 640–654.

[9] O. Ibarra, "The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications," *SIAM Journal on Computing*, vol. 4, pp. 524–532, 1978.

[10] M. P. Schützenberger, "Sur les relations rationnelles," in *GI Conference on Automata Theory and Formal Languages*, ser. LNCS, vol. 33, 1975, pp. 209–213.

[11] K. Culic and J. Karhumäki, "The Equivalence Problem for Single-Valued Two-Way Transducers (on NPDT0L Languages) is Decidable," *SIAM Journal on Computing*, vol. 16, no. 2, pp. 221–230, 1987.

[12] A. Mal'cev, *The Metamathematics of Algebraic Systems*. North-Holland, 1971.

[13] W. Thomas, "Languages, automata, and logic," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Springer, 1997, vol. 3, pp. 389–455.

[14] M. Kaminski and N. Francez, "Finite-memory automata," in *31st Annual Symposium on Foundations of Computer Science (FOCS 1990)*, vol. 2. IEEE, 1990, pp. 683–688.

[15] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines – a survey," in *Proceedings of the IEEE*, vol. 84, no. 8, Berlin, Aug 1996, pp. 1090–1123.

[16] L. Segoufin, "Automata and logics for words and trees over an infinite alphabet," in *CSL*, ser. LNCS, Z. Ésik, Ed., vol. 4207, 2006, pp. 41–57.

[17] F. Neven, T. Schwentick, and V. Vianu, "Finite state machines for strings over infinite alphabets," *ACM Trans. CL*, vol. 5, pp. 403–435, 2004.

[18] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David, "Two-variable logic on words with data," in *LICS*. IEEE, 06, pp. 7–16.

[19] M. Benedikt, C. Ley, and G. Puppis, "Automata vs. logics on data words," in *CSL*, ser. LNCS, vol. 6247. Springer, 2010, pp. 110–124.

[20] T. L. Gall and B. Jeannet, "Lattice automata: A representation for languages on infinite alphabets, and some applications to verification," in *SAS 2007*, ser. LNCS, vol. 4634, 2007, pp. 52–68.

[21] G. V. Noord and D. Gerdemann, "Finite state transducers with predicates and identities," *Grammars*, vol. 4, no. 3, pp. 263–286, 2001.

[22] D. Perrin, "Finite automata," in *Handbook of Theoretical Computer Science. Volume B: Formal Models and Semantics*, J. van Leeuwen, Ed. Elsevier and the MIT Press, 1990, pp. 1–57.

[23] P. Hooimeijer and M. Veanes, "An evaluation of automata algorithms for string analysis," in *VMCAI'11*, ser. LNCS. Springer, 2011.

[24] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Saner: Composing static and dynamic analysis to validate sanitization in web applications," in *SP*, 2008.

[25] Y. Minamide, "Static approximation of dynamically generated web pages," in *WWW '05: Proceedings of the 14th International Conference on the World Wide Web*, 2005, pp. 432–441.

[26] G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," in *PLDI*. ACM, 2007, pp. 32–41.

[27] G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su, "Dynamic test input generation for web applications," in *ISSTA*, 2008.

[28] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst, "HAMPI: a solver for string constraints," in *ISSTA*, 09, pp. 105–116.

[29] P. Saxena, D. Akhawe, S. Hanna, S. McCamant, F. Mao, and D. Song, "A symbolic execution framework for javascript," in *SP*, 2010.

[30] P. Hooimeijer and W. Weimer, "A decision procedure for subset constraints over regular languages," in *PLDI*. ACM, 2009, pp. 188–198.

[31] M. Veanes, P. d. Halleux, and N. Tillmann, "Rex: Symbolic regular expression explorer," in *ICST*. IEEE, 2010, pp. 498–507.

[32] Pex, http://research.microsoft.com/projects/pex.

[33] N. Tillmann and J. de Halleux, "Pex - white box test generation for .NET," in *TAP*, ser. LNCS, vol. 4966. Springer, 2008, pp. 134–153.

[34] M. Veanes, N. Tillmann, and J. de Halleux, "Qex: Symbolic SQL query explorer," in *LPAR-16*, ser. LNAI, E. Clarke and A. Voronkov, Eds., vol. 6355. Springer, 2010, pp. 425–446.

APPENDIX

A. Axioms for SFTs

We develop an axiomatic approach for reasoning about SFTs. The background is assumed to include *lists*, as a special case of algebraic data types. For each element sort σ there is a list sort $\text{LIST}\langle\sigma\rangle$. The following background functions are assumed for elements of sort $\text{LIST}\langle\sigma\rangle$:

$$\begin{array}{rl} hd: & \text{LIST}\langle\sigma\rangle \to \sigma \\ tl: & \text{LIST}\langle\sigma\rangle \to \text{LIST}\langle\sigma\rangle \\ [\_|\_]: & \sigma \times \text{LIST}\langle\sigma\rangle \to \text{LIST}\langle\sigma\rangle \\ \varepsilon: & \text{LIST}\langle\sigma\rangle \end{array}$$

The standard list axioms are assumed to hold for these functions. In particular, for all $e:\sigma$ and $l:\text{LIST}\langle\sigma\rangle$, $hd([e\,|\,l]) = e$, $tl([e\mid l]) = l$, and $[e\mid l] \neq \varepsilon$. We write $[e_0,\ldots,e_n]\cdot l$, or $[e_0,\ldots,e_n]$ when $l=\varepsilon$, as a shorthand for $[e_0\,|\cdots[e_n\,|\,l]]$. Given a unary function symbol f, term t, and $n \geq 0$, we write $f^{(0)}(t)$ for t and $f^{(n+1)}(t)$ for $f(f^{(n)}(t))$. For example $tl^{(2)}([e_0,e_1,e_2,e_3])$ denotes the list $[e_2,e_3]$.

We adapt the notion of IDs and step relations from [3] to SFTs. An ID is an *Instantaneous Description* of a possible state of an SFT together with an input word (still to be read) and output word (still to be written) starting from that state. The formal definition is as follows. Words in $(\mathcal{U}^\sigma)^*$ are encoded by lists in $\mathcal{U}^{\text{LIST}\langle\sigma\rangle}$. In the following let $A^{\iota/o}$ be an SFT.

***Definition** 14:* An *ID* of A is a triple (v,q,w) where $v \in \mathcal{U}^{\text{LIST}\langle\iota\rangle}$, $q \in Q_A$, and $w \in \mathcal{U}^{\text{LIST}\langle o\rangle}$. The *step relation* of A is the binary relation $\vdash_A$ over IDs:

$$\begin{array}{rcl} ([a\,|\,v],p,u\cdot w) \vdash_A (v,q,w) & \Leftrightarrow & (p,a,u,q) \in \Delta^{\bar\epsilon}_{[\![A]\!]} \\ (v,p,u\cdot w) \vdash_A (v,q,w) & \Leftrightarrow & (p,\epsilon,u,q) \in \Delta^{\epsilon}_{[\![A]\!]} \end{array}$$

We write $\vdash_t$ for $\vdash_A$ restricted to a single symbolic transition $t \in \Delta_A$. The following proposition follows from definitions, where $\vdash_A^*$ denotes the reflexive and transitive closure of $\vdash_A$.

***Proposition** 4:* $T_A(v) = \{w \mid \exists q \in F_A((v,q_A^0,w) \vdash_A^* (\varepsilon,q,\varepsilon))\}$.

The following lemma is used below. For uniformity, we view guards of transitions as either singleton sequences containing a formula or empty sequences. Thus, the length of a guard of a transition is 0 if the transition is an input-epsilon transition, the length is 1, otherwise.

***Lemma** 6: Let $t=(p,\varphi,u,q)\in\Delta_A$, $v,v'\in\mathcal{U}^{\text{LIST}\langle\iota\rangle}$, and $w,w'\in\mathcal{U}^{\text{LIST}\langle o\rangle}$. The following statements are equivalent*

- $\mathcal{U} \vDash Step_t(v,w)$ *and* $v' = tl^{(|\varphi|)}(v)$ *and* $w' = tl^{(|u|)}(w)$
- $(v,p,w) \vdash_t (v',q,w')$.

Proof: By using list axioms and Definition 14. ∎

The *correctness criterion* that we want $Th(A)$ to fulfill is $IsSat(\bigwedge Th(A) \wedge Acc_A(v,w))$ if an only if $w \in T_A(v)$. To this end, we need to consider SFTs whose step relation is well-founded.

***Theorem** 6: $Th(A)$ is satisfiable. Moreover, if $\vdash_A$ is well-founded, then there is a unique Σ_A-model $M_A \vDash Th(A)$ such that $w \in T_A(v)$ if and only if $M_A \vDash Acc_A(v,w)$.*

Proof: To show satisfiability of $Th(A)$ define the Σ_A-model M by $Acc_p^M(v,w)$ iff $\exists f \in F_A((v,p,w) \vdash_A^* (\varepsilon,f,\varepsilon))$. It follows from definitions that $M \vDash \psi$ for all $\psi \in Th(A)$.

Next, assume $\vdash_A$ is well-founded. Let $Q = Q_A$. Since Q is finite it follows that there exists a well-ordering $>_Q$ over Q such that if $p >_Q q$ then there exists no epsilon-path $q \overset{\epsilon/\epsilon}{\rightsquigarrow}_A p$, or, equivalently, that $(\varepsilon,q,\varepsilon) \not\vdash_A^* (\varepsilon,p,\varepsilon)$. Define the lexicographic order $\succ$ over $\mathcal{U}^{\text{LIST}\langle\iota\rangle} \times \mathcal{U}^{\text{LIST}\langle o\rangle} \times Q$ as follows:

$$\begin{array}{rl} (v,w,q) \succ (v',w',q') \overset{\text{def}}{=} & |v| > |v'| \vee \\ & (|v| = |v'| \wedge |w| > |w'|) \vee \\ & (|v| = |v'| \wedge |w| = |w'| \wedge q >_Q q') \end{array}$$

Next, we prove ($\star$) by induction over $\succ$.

($\star$) For all $p \in Q$, $v \in \mathcal{U}^{\text{LIST}\langle\iota\rangle}$, $w \in \mathcal{U}^{\text{LIST}\langle o\rangle}$, and all $M \vDash Th(A)$:

$$\exists f \in F_A((v,p,w) \vdash_A^* (\epsilon,f,\epsilon)) \Leftrightarrow M \vDash Acc_p(v,w).$$

To establish the base case, let (v,w,p) be minimal with respect to $\succ$. Thus, $v=w=\varepsilon$ and there are no epsilon moves from p. Hence, by $M \vDash ax_p$,

$$\begin{array}{rcl} M \vDash Acc_p(\varepsilon,\varepsilon) & \Leftrightarrow & (p \in F_A \wedge \varepsilon=\varepsilon \wedge \varepsilon=\varepsilon) \vee M \vDash \psi \\ & \Leftrightarrow & p \in F_A \\ & \Leftrightarrow & \exists f \in F_A((\varepsilon,p,\varepsilon) \vdash_A^* (\varepsilon,f,\varepsilon)) \end{array}$$

where $M \nvDash \psi$ since each step formula from p contains a trivially false conjunct $\varepsilon \neq \varepsilon$ because there are no epsilon moves from p.

The case is also clear for $v=w=\varepsilon$ and $p \in F_A$, because then it does not make a difference if $M \vDash \psi$ or $M \nvDash \psi$.

Now consider any (v,w,p) such that $p \in F_A \wedge v=\varepsilon \wedge w=\varepsilon$ is false and (v,w,p) is not minimal with respect to $\succ$. Hence, by $M \vDash ax_p$,

$$\begin{array}{rcl} M \vDash Acc_p(v,w) & \Leftrightarrow & \bigvee_{t\in\Delta_A} (\exists v'\,w'\,((v,p,w) \vdash_t (v',q,w') \wedge \\ & & \qquad M \vDash Acc_q(v',w'))) \\ & \Leftrightarrow & \bigvee_{t\in\Delta_A} (\exists v'\,w'\,((v,p,w) \vdash_t (v',q,w') \wedge \\ & & \qquad \exists f \in F_A((v',q,w') \vdash_A^* (\varepsilon,f,\varepsilon))) \\ & \Leftrightarrow & \exists f \in F_A((v,p,w) \vdash_A^* (\varepsilon,f,\varepsilon))) \end{array}$$

where the first equivalence follows from Lemma 6 and Definition 13. The second equivalence follows from the IH, since $(v,w,p) \succ (v',w',q)$. The third equivalence holds by definition of $\vdash_A$. ($\star$) follows by the induction principle.

The rest follows from ($\star$) by letting $p = q_A^0$ and by using Proposition 4. Finally, M_A is unique by definition of T_A. ∎

Theorem 6 fails if we omit the condition that $\vdash_A$ is well-founded.

***Example** 14:* Consider A with a single (nonfinal) state q and a single transition (q,ϵ,ϵ,q). Then $Th(A) = \{\forall x\,y\,(Acc_q(x,y) \Leftrightarrow Acc_q(x,y))\}$. So all Σ_A-models satisfy $Th(A)$ while $T_A(v) = \emptyset$ for all v. ∎

The following proposition provides a simple condition over the structure of A that is equivalent to $\vdash_A$ being well-founded; the proposition reflects the role of $>_Q$ in the proof of Theorem 6.

***Proposition** 5: $\vdash_A$ is well-founded $\Leftrightarrow$ A is epsilon-loop-free.*

The practical significance of the proposition is that there is an efficient algorithm that given A constructs an equivalent epsilon-loop-free SFT from A. While full epsilon move elimination may cause quadratic increase in the number of symbolic transitions (by eliminating *sharing*), epsilon-loop elimination does not increase the number of symbolic transitions.

Define the *ϵ-loop closure* as the intersection of the epsilon closure and the the inverse epsilon closure of a state:

$$\widetilde{p} \stackrel{\text{def}}{=} \{q \in Q_A \mid p \stackrel{\epsilon/\epsilon}{\rightsquigarrow}_A q\} \cap \{q \in Q_A \mid q \stackrel{\epsilon/\epsilon}{\rightsquigarrow}_A p\}$$

and lift the definition to sets: $\widetilde{P} \stackrel{\text{def}}{=} \{\widetilde{p} \mid p \in P\}$.

***Definition** 15:* Let $\widetilde{A} \stackrel{\text{def}}{=} (\widetilde{Q_A}, \widetilde{q^0_A}, \widetilde{F_A}, \iota, o, \Delta)$ where $\Delta = \{(\widetilde{p}, \varphi, u, \widetilde{q}) \mid (p, \varphi, u, q) \in \Delta_A\} \setminus \{\widetilde{p} \xrightarrow{\epsilon/\epsilon} \widetilde{p} \mid p \in Q_A\}$.

The following proposition follows from Definition 15 and by using techniques similar to the proof of equivalence between nondeterministic finite automata and nondeterministic finite automata with epsilon-moves (see [3]).

***Proposition** 6: $\widetilde{A}$ is epsilon-loop-free and $T_A = T_{\widetilde{A}}$.*

The following corollary provides a foundation for expressing decision problems over SFTs as logical formulas and provides a basis for integration of $Th(\widetilde{A})$ in the context of state-of-the-art satisfiability modulo theories (SMT) solvers, as discussed in the following section.

***Corollary** 1: $M_{\widetilde{A}} \vDash Acc_{\widetilde{A}}(v, w) \Leftrightarrow w \in T_A(v)$.*

Proof: Use Theorem 6 and Propositions 5 and 6. ■

The following corollary shows how two SFT theories can be combined for non-equivalence checking.

***Corollary** 2:* Let $A^{\iota/o}$ and $B^{\iota/o}$ be epsilon-loop-free SFTs such that $\Sigma_A \cap \Sigma_B = \emptyset$. Let $\Sigma = \Sigma_A \cup \Sigma_B \cup \{x : \text{LIST}\langle\iota\rangle, y : \text{LIST}\langle o\rangle, z : \text{LIST}\langle o\rangle\}$. Then $Th(A) \cup Th(B)$ is satisfiable and the following statements are equivalent for Σ-models M such that $M \vDash Th(A) \cup Th(B)$.

1) $M \vDash Acc_A(x, y) \wedge \neg Acc_B(x, y)$
2) $y^M \in T_A(x^M)$ and $y^M \notin T_B(x^M)$.

Proof: By $\Sigma_A \cap \Sigma_B = \emptyset$ Theorem 6 and Proposition 5. ■

B. Implementation

SMT solvers establish satisfiability of formulas modulo theories. Two inputs are given:

- *Auxiliary Axioms*: a satisfiable set Ψ of universally quantified Σ_Ψ-axioms.
- *Goal formula*: a quantifier free $(\Sigma \cup \Sigma_\Psi)$-formula γ, where $\Sigma \cap \Sigma_\Psi = \emptyset$.

The case of an empty set of axioms is most common. Otherwise, axioms in Ψ are expanded using quantifier instantiation as explained below. In order to use Ψ it is required that Ψ is *well-formed* for γ. Different well-formdness requirements exist, the one we use is specific for SFT theories and is defined below. Under the well-formdness assumption, if $\gamma \wedge \bigwedge \Psi$ is satisfiable, the solver generates a partial model $M \vDash \gamma$ that can be extended to a model of Ψ. M is *partial* in the sense that the interpretation given for function symbols in Σ_Ψ is finite and is restricted to values t^M for all subterms t (of appropriate sort) that occur during model search. The following general property is used as a correctness assumption of SMT solvers.

***Proposition** 7: If $\gamma \wedge \bigwedge \Psi$ is satisfiable and Ψ is well-formed for γ then the SMT solver generates a partial model $M \vDash \gamma$ that can be extended to a model of Ψ.*

In the following we describe how $Th(A)$ is used as auxiliary axioms. First, we introduce an equivalence preserving transformation $Th'(A)$ of $Th(A)$. For each axiom $\forall x\, y\, (Acc_p(x, y) \Leftrightarrow \beta) \in Th(A)$, there are two axioms in $Th'(A)$ where the lhs of each axiom is declared as its *pattern*:

$$\forall y\, (Acc_p(\varepsilon, y) \Leftrightarrow \beta\{x \mapsto \varepsilon\}),$$
$$\forall e\, r\, y\, (Acc_p([e \,|\, r], y) \Leftrightarrow \beta\{x \mapsto [e \,|\, r]\}),$$

It is clear that $Th'(A)$ is equivalent to $Th(A)$. The particular choice of patterns reflects the use of $Th'(A)$ below.

Consider a goal formula γ containing a subformula φ that *matches* the pattern α of an axiom $\forall \mathbf{x}\, (\alpha \Leftrightarrow \beta)$, meaning that $\varphi = \alpha\theta$ for some substitution θ for $\mathbf{x}$. Then, the *unfolding of γ wrt α* is the formula γ' obtained from γ by replacing all occurrences of φ with $\beta\theta$, denoted here by $\gamma \vdash_\Psi \gamma'$. Unfolding clearly preserves equivalence wrt Ψ. A *Ψ-reduct of γ*, if one exists, is a formula γ' such that $\gamma \vdash^*_\Psi \gamma'$ and γ' cannot be unfolded.

***Definition** 16:* Ψ is *well-formed* for γ if $\vdash_\Psi$ is well-founded and any Ψ-reduct of γ is a Σ-formula.

In other words, well-formdness implies that repeated unfolding of γ is guaranteed to terminate with a formula that does not contain any symbols from Σ_Ψ.

1) SFT nonequivalence checking: Two SFTs $A^{\iota/o}_\Gamma$ and $B^{\iota/o}_\Gamma$ are equivalent if and only if the following conditions hold:

- A and B are *domain equivalent*: $L(\mathbf{d}(A)) = L(\mathbf{d}(B))$.
- A and B are *partial equivalent*, $A \cong B$: forall $v \in L(\mathbf{d}(A)) \cap L(\mathbf{d}(B))$, $T_A(v) = T_B(v)$.

Domain equivalence is decidable if $\mathcal{F}(\{c_\iota\})$ is decidable by using the difference algorithm for SFAs [8]. In Fig. 3 we provide an algorithm for generating a witness when $A \ncong B$ and A and B are input-ϵ-free. Using a different transformation of $Th(A)$ the algorithm can be extended to all SFTs. This algorithm provides a practically useful extension of Theorem 2 when A and B are single-valued and can be used as a semidecision procedure for $A \ncong B$, in general.

***Theorem** 7: Assume A and B are input-ϵ-free and $\mathcal{F}$ is decidable. If $A \ncong B$ then Witness$^{\ncong}(A, B)$ terminates and generates (v, c), s.t., $c \in T_A(v) \setminus T_B(v) \cup T_B(v) \setminus T_A(v)$. Moreover, v is shortest possible.*

Proof: Let γ be the formula in line 6 in Fig. 3. Let $\Psi = Th'(A) \cup Th'(B)$. Assume $A \ncong B$. Since A and B are input-ϵ-free, for every $\forall e\, r\, y\, (Acc_p([e \,|\, r], y) \Leftrightarrow \beta) \in \Psi$, the first argument of any Acc_q in β is r, and for every $\forall y\, (Acc_p(\varepsilon, y) \Leftrightarrow \beta) \in \Psi$, $\Sigma(\beta) \cap \Sigma_\Psi = \emptyset$. It follows that Ψ is well-formed for γ. Now use Proposition 7 and Corollary 2. It follows also from the algorithm that x^M is shortest. ■