Equivalence of Finite-Valued Symbolic Finite Transducers

Margus Veanes and Nikolaj Bjørner

Microsoft Research
{margus,nbjorner}@microsoft.com

Abstract. Symbolic Finite Transducers, or SFTs, is a representation of finite transducers that annotates transitions with logical formulas to denote sets of concrete transitions. This representation has practical advantages in applications for web security analysis, where 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. Equivalence checking plays a central role in deciding properties of SFTs such as idempotence and commutativity. We show that equivalence of finite-valued SFTs is decidable, i.e., when the number of possible outputs for a given input is bounded by a constant.

1 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 predicates, and sometimes it is beneficial to use character types possibly even with an infinite domain, e.g., integers. We are interested in transducers that arise from applications such as web sanitizers and string encoders [1], that work over large alphabets like Unicode. The focus here is on the class of SFTs that, for a given input sequence, can output a finite number of possible outputs sequences.

A concrete example is an Html sanitizer that may either use a decimal or a hexadecimal encoding of characters codes, see Figure 1. Other typical parameters are, whether to use a "safe list" (of characters not to be encoded) or not, or whether to use shorthands such as `"&"` for encoding `"&"` and other common characters. Viewed as an SFT, a given input string such as `"="` may be encoded as `"&#61;"` or as `"&#x3D;"`, corresponding to the value of the second parameter of `EncodeHtml`, but there is an upper bound on how many different outputs an input sequence may be mapped into (two in this case), i.e., the underlying SFT is *finite-valued*. Our main result is that equivalence of SFTs is decidable in the finite-valued case. This is a nontrivial extension of decidability of SFT

```
1: static string EncodeHtml(string strInput, bool useDecimal = false)
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(useDecimal ? "&#{0}" : "&#x{0:X}",(int)c));
14:       b.Append(";");
15:     }
16:   return b.ToString();
17: }
```

Fig. 1. Html sanitizer with decimal or hexadecimal formatting.

equivalence in the *single-valued* case [2] and enables analysis scenarios that are, in general, not expressible with single-valued SFTs. SFTs do not have a notion of parameters other than the actual input. Instead, the use of parameters can be abstracted by considering finite-valued transducers.

2 Examples and an Application to Web Sanitizers

We here illustrate the use of SFT analysis on web security 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 [3]. We represent a sanitizer program as a symbolic finite transducer. It uses transduction functions.

Example 1 (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 $k = 16$:

$$\begin{aligned} <&: \mathrm{BV}_k \times \mathrm{BV}_k \to \mathrm{BOOL}, \\ \pi^n_m &: \mathrm{BV}_k \to \mathrm{BV}_k, \text{ for } 0 \leq m < n \leq k, \\ \oplus &: \mathrm{BV}_k \times \mathrm{BV}_k \to \mathrm{BV}_k, \end{aligned}$$

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

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

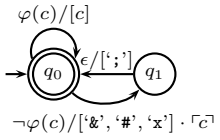
extracts the j'th nibble (half-byte) of c, $0 \leq j \leq 3$, and maps it to its hexadecimal representation (‘0’,‘1’,. . . ,‘9’,‘A’,. . . ,‘F’).

The transduction function allows defining a symbolic transducer.

Example 2 (Transducer Guards). The SFT below represents a so-called “string sanitizer”, where certain characters c in the input string, not satisfying the condition

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

are in the output string replaced by their hexadecimal representation:



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

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

with $\mathbf{h}_j$'s as defined in Example 1. It is also straight-forward to rewrite the conditions into four transitions with simple guards and a fixed number of outputs each.

The work in [1] introduces a domain specific language Bek based on SFTs for writing and analyzing sanitizers. The main application of SFTs in the context of Bek is to formally verify key security properties of sanitizers. Two examples of such 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 and (other optional parameters) the corresponding SFTs are consequently finite-valued and often *total*, i.e., produce at most some bounded number of output strings for each input string and accept all input strings.

3 Preliminaries

We recall the definition of a finite transducer [4]. 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 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 [4, 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.

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

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*,

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

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*[1] if $\Delta_A^{\epsilon} = \emptyset$. We say A is *input-ϵ-free*.

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 [5]. Definition 5 is consistent with [4].

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

3.1 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 $\mathit{Ite}(\varphi, t, f)^M =$ if φ^M then t^M 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 $\mathit{IsSat}(\varphi)$; 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 uninterpreted 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 $\mathcal{T}^{\text{BOOL}}$.

4 Symbolic Finite 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 $A_\Gamma^{\iota/o}$, 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 $\Delta = \Delta^{\bar{\epsilon}} \cup \Delta^\epsilon$,

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

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

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

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

Example 3. Consider the SFT A in Example 2. 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 $\mathit{IsSat}(\varphi)$ 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.

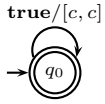
Proposition 1. *A is deterministic if and only if A is input-ϵ-free and for all $p \xrightarrow{\varphi/\mathbf{u}} q, p \xrightarrow{\psi/\mathbf{v}} r \in \Delta_A$, if $q \neq r$ then $\varphi \wedge \psi$ is unsatisfiable.*

4.1 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 2, $|\mathcal{U}^{\text{BV}_{16}}| = 2^{16}$. 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 4. Consider the sort INT for integers and the following SFT $A^{\text{INT}/\text{INT}}$:



The image of T_A is $\{[n,n] \mid n \in \mathcal{U}^{\text{INT}}\}^*$ 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 4 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.

5 Equivalence

Our main theorem is Theorem 1, it builds on Lemma 5 as our main technical result. The theorem generalizes the decidability of equivalence of single-valued SFTs [1]. The main reason why the technique for checking equivalence of single-valued SFTs does not generalize to checking equivalence of finite-valued SFTs is that the dependency from inputs to outputs does not remain *functional* in the finite-valued case. In the single-valued case one can detect inequivalence during an incremental product construction using local satisfiability checks, by essentially detecting *non-single-valuedness* of the product [1, Lemma 2]; this is nonsensical in the finite-valued case.

We use several lemmas to prove Theorem 1. The main ones are Lemma 4 and Lemma 5. Lemma 4 is used to transform the SFTs into a normal form that considerably simplifies the proof of Lemma 5. The main construction used in Lemma 5 is a product construction of the given SFTs. The product construction uses multiple outputs. The number of states in the product is bounded by the product of the number of states of the component SFTs. The key idea is to exhaustively detect *conflict-states* that represent product states at which point we know that there exists an input element that will at some point cause different outputs be yielded by the SFTs.

Proposition 2. *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 dead-ends, 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^{ϵ}_A and $\Delta^{\bar{\epsilon}}_A$.

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

1. Choose a non-initial state q such that $\Delta^{\epsilon}_A(q) \neq \emptyset$.
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 dead-ends that were created.

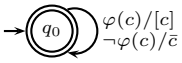
Finally, transitions in $\Delta^{\epsilon}_A(q^0_A)$ are eliminated one by one as follows. Fix $q^0_A \xrightarrow{\epsilon/\mathbf{u}} p$. Since $T_A(\epsilon) = \emptyset$ we know that $p \notin F_A$ and since p is not a dead-end $\Delta_A(p) \neq \emptyset$. We know also that $q^0_A \neq p$. Replace 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 2 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 5. Consider the SFT in Example 2. Input-epsilon elimination yields the following equivalent SFT:



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

We say that a state of an SFT is *relevant* if it is reachable from the initial state and not a dead-end. We use the following pumping lemma over word equations.

Lemma 1. *For all $u_1, u_2, v_1, v_2, w_1, w_2, z_1, z_2$: if $u_1 \cdot u_2 = v_1 \cdot v_2$, $u_1 \cdot w_1 \cdot u_2 = v_1 \cdot z_1 \cdot v_2$ and $u_1 \cdot w_2 \cdot u_2 = v_1 \cdot z_2 \cdot v_2$ then $u_1 \cdot w_1 \cdot w_2 \cdot u_2 = v_1 \cdot z_1 \cdot z_2 \cdot v_2$.*

Lemma 2. *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 Lemma 1, 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 $F_A = \{q_A^0\}$, q_A^0 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 $q_A^0 = q_{A_0}^0$, $F_A = F_{A_n}$, and there is a single transition $q_{A_i}^0 \to q_{A_{i+1}}^0$ 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 3. *All SFTs have an effectively equivalent SNF.*

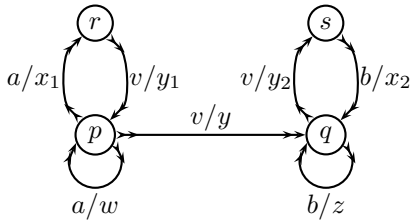
Proof. Let A be an SFT. The sequences are constructed by considering all loop-free paths from the initial state of A to some final state, possibly creating extra states if a strongly connected component of A is entered and exited through different states. □

The following lemma is used to simplify the proof of Lemma 5 by normalizing the representation of SFTs.

Lemma 4. *Every finite-valued SFT has an effectively equivalent SNF with single-valued sequences.*

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

$$\begin{aligned}&\varphi_1\theta_1 \wedge \psi\theta_2 \wedge \psi_2\theta_3 \wedge \varphi_2'\theta_4 \wedge \varphi_1'\theta_1 \wedge \psi_1\theta_2 \wedge \psi\theta_3 \wedge \varphi_2\theta_4\\ &\wedge u_1\theta_1 \cdot t\theta_2 \cdot t_2\theta_3 \cdot u_2'\theta_4 \neq u_1'\theta_1 \cdot t_1\theta_2 \cdot t\theta_3 \cdot u_2\theta_4\end{aligned}$$

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 $A' = A{\restriction}\bar{D}$ (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. □

The following lemma is our main technical result. Some details of the proof have been omitted but can be found in [7]. It generalizes the decidability of equivalence of single-valued SFTs [1]. 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 $\exists x \left(\bigwedge_{i=1}^k T_A(x) \neq T_{B_i}(x)\right)$ 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

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

such that $IsSat(\varphi\wedge\psi_1\wedge\psi_2)$. Note that $L(\mathbf{d}(D)) = L$. The unreachable states and the dead-ends 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 dead-ends.

(*) 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 loop-free path $s_0 \overset{u/(v,w,\_)}{\rightsquigarrow} s_f$ such that $|v| \neq |w|$.

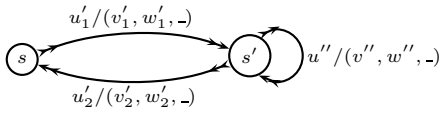
Proof of (*): We show that cases 1.a and 1.b are exhaustive. Consider any $u \in L$ such that $D(u) = (v, w, \_)$ and $|v| \neq |w|$ and suppose 1.b is false. Then there must exist $u_1, u', u_2, v_1, v', v_2, w_1, w', w_2$ such that

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

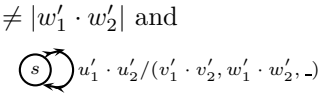
and a loop $s \overset{u'/(v',w',\_)}{\rightsquigarrow} s$ where $|v'| \neq |w'|$, or else $|v| = |w|$ since 1.b is false. Now suppose the loop is not simple.
Then there exist $u_1', u'', u_2', v_1', v'', v_2', w_1', w'', w_2'$ such that

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

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'. □

In the case of 1.a we have that for any path

$$s_0 \overset{u_1/(v_1,w_1,\_)}{\rightsquigarrow} s \overset{u_2/(v_2,w_2,\_)}{\rightsquigarrow} s_f$$

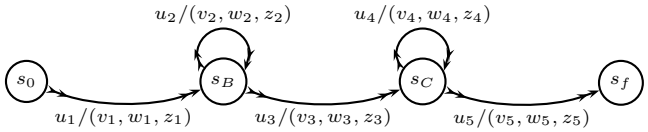
there exists u, v, w, $|v| \neq |w|$, and a large enough $m \geq 0$, such that, for all $n \geq m$,

$$\begin{aligned} &D(u_1 \cdot u^n \cdot u_2) = (v_1 \cdot v^n \cdot v_2, w_1 \cdot w^n \cdot w_2, \_), \\ &\quad |v_1 \cdot v^n \cdot v_2| \neq |w_1 \cdot w^n \cdot w_2| \end{aligned}$$

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_B and s_C 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

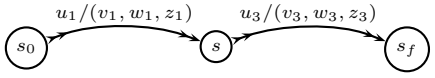


where $|v_2| \neq |w_2|$ and $|v_4| \neq |z_4|$. It follows that there exist m and n such that

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

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 loop-free 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

$$|v_1 \cdot v_2^m \cdot v_3| \neq |w_1 \cdot w_2^m \cdot w_3|, \qquad |v_1 \cdot v_2^m \cdot v_3| \neq |z_1 \cdot z_2^m \cdot z_3|$$

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 simultaneous

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 loop-free 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.

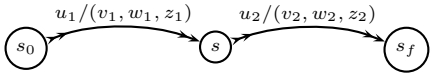
Case (B.2, C.1): Assume, by previous cases, that (B.1, C.1) is not possible. Let ℓ be the length of the longest possible output from either A, B or C on any loop-free path. Clearly, ℓ can be computed effectively. Suppose there exists a C-length-conflict-state s. Consider all paths

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

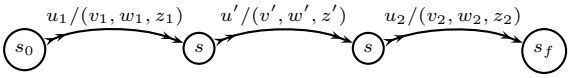
Since B.1.a is not possible, we know that for all loops $s \rightsquigarrow s$ the A-output and the B-output have the same length. For each ρ_m check if a simultaneous B-character-conflict and C-length-conflict exists.

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-character-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 1 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 loop-free 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) is proved in [7]. One can show that the above cases are exhaustive. Decidability follows for $k = 2$. □

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

Theorem 1. *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 4 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. □

6 Related Work

Equivalence checking of FTs is undecidable in general [8], and is undecidable already for GSMs. The special case of equivalence checking of single-valued SFTs over decidable character background is shown to be decidable in [2]. This result is substantially generalized here (Theorem 1) to *finite-valued* SFTs. This result generalizes also the decidability of equivalence of finite-valued FTs [9, 5, 10, 11]. Lemma 4 is a symbolic generalization of a decomposition technique studied in [11]. A fundamental simplifying assumption compared to SFTs is that the range of an FT is always regular. Equivalence of single-valued *extended* SFTs (SFTs with lookahead) is studied in [12], the motivation there is to analyze decoders, and it gives an orthogonal extension of decidability of equivalence of single-valued SFTs. 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 [3], 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 [13–15].

7 Conclusion

We studied equivalence of finite-valued Symbolic Finite Transducers. Although equivalence checking is in general undecidable 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). The symbolic representation of transducers is both convenient for applications and allows for succinct representations. Basic automata algorithms lift in many cases in a straight-forward way to this representation, and it allows leveraging state-of-the-art theorem proving technology for analyzing the automata. Our main motivation behind this work originates from analysis of sanitizers.

References

1. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes, "Fast and precise sanitizer analysis with Bek," in *USENIX Security*, 2011, pp. 1–16.
2. M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner, "Symbolic finite state transducers: Algorithms and applications," in *POPL'12*. ACM, 2012, pp. 137–150.
3. 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.
4. S. Yu, "Regular languages," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Springer, 1997, vol. 1, pp. 41–110.
5. A. Demers, C. Keleman, and B. Reusch, "On some decidable properties of finite state translations," *Acta Informatica*, vol. 17, pp. 349–364, 1982.
6. M. A. Harrison, *Introduction to Formal Language Theory*. Addison-Wesley, 1978.
7. N. Bjørner and M. Veanes, "Symbolic transducers," Microsoft Research, Technical Report MSR-TR-2011-3, 2011.
8. 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.
9. M. P. Schützenberger, "Sur les relations rationnelles," in *GI Conference on Automata Theory and Formal Languages*, ser. LNCS, vol. 33, 1975, pp. 209–213.
10. 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.
11. A. Weber, "Decomposing finite-valued transducers and deciding their equivalence," *SIAM J. Comput.*, vol. 22, no. 1, pp. 175–202, Feb. 1993.
12. L. D'Antoni and M. Veanes, "Equivalence of extended symbolic finite transducers," in *CAV'13*. Springer, 2013, pp. 624–639.
13. 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.
14. G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," in *PLDI*. ACM, 2007, pp. 32–41.
15. G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su, "Dynamic test input generation for web applications," in *ISSTA*, 2008.