Completeness in Approximate Transduction

Mila Dalla Preda[1], Roberto Giacobazzi[1&2], and Isabella Mastroeni[1]

[1]University of Verona and [2]IMDEA Software Institute
{mila.dallapreda, roberto.giacobazzi, isabella.mastroeni}@univr.it

Abstract. Symbolic finite automata (SFA) allow the representation of regular languages of strings over an infinite alphabet of symbols. Recently these automata have been studied in the context of abstract interpretation, showing their extreme flexibility in representing languages at different levels of abstraction. Therefore, SFAs can naturally approximate sets of strings by the language they recognise, providing a suitable abstract domain for the analysis of symbolic data structures. In this scenario, transducers model SFA transformations. We characterise the properties of transduction of SFAs that guarantee soundness and completeness of the abstract interpretation of operations manipulating strings. We apply our model to the derivation of sanitisers for preventing cross site scripting attacks in web application security. In this case we extract the code sanitiser directly from the backward (transduction) analysis of the program given the specification of the expected attack in terms of SFA.

Keywords: Abstract interpretation, symbolic automata, symbolic transducers.

1 Introduction

Symbolic finite automata (SFA) have been introduced as an extension of traditional finite state automata for modelling languages with a potential infinite alphabet [28]. Transitions in SFA are modelled as constraints interpreted in a given Boolean algebra, providing the semantic interpretation of constraints, and therefore the (potentially infinite) structural components of the language recognised by the automaton. This extends the class of regular languages over potentially unbound alphabets, providing a number of applications, from code sanitisation [19] to verification (see [23] for an introductory account) to models for reasoning and comparing program analysis tools, such as in the case of similarity analysis of binary executables in [10]. Similarly, symbolic finite transducers (SFT) perform language transformation and correspond to operations performed over a language, e.g., the language recognised by an SFA (see for instance [3,4,11,13,26]).

Recently the abstract interpretation of SFAs has been considered with the aim of using these symbolic (finite) structures to reason about potentially infinite structures, such as the sequences of instructions of the possible executions of a program [10]. In this case, SFA can be approximated by abstract interpretation of either its constraint part or of its interpretation, respectively leading to a syntactic or semantic abstraction. In both cases, the language recognised by the approximated (abstract) symbolic automation is an over approximation of the original (concrete) one, as usual in abstract interpretation. An abstract symbolic automaton is nothing else than a symbolic automaton over a modified (approximated) semantics or constraint structure.

Abstract symbolic automata can be used as the elements of an abstract domain of objects representing languages. This allows us to use standard results in SFA to reason about approximate regular languages parametrically on the level of abstraction chosen for interpreting or representing strings [10]. Predicate transformers in this domain require the abstract interpretation of language transformers. This is particularly important if we plan to use these symbolic structures to approximate the semantics of string manipulating programs, such as in code santizers examining an HTML document and producing a new and modified HTML file as output.

In this paper, we investigate the notion of approximate symbolic transduction. Transducers are Mealy automata that transform languages. The idea is to consider transductions as language transformers where languages are approximated by SFAs. If L is a language, an approximation of L, is a language $L^\sharp$ such that $L \subseteq L^\sharp$. These approximate languages can be obtained by considering regular approximations of, for instance, context free languages or by abstracting the concrete SFA representing some regular language over a possibly infinite alphabet. The operation of transduction transforms these (input) approximate languages into other (output) approximate languages, the so called transduced languages.

We prove that a notion of completeness, both in its forward ($\mathcal{F}$) and backward ($\mathcal{B}$) sense, can be formalised for the transduction of SFAs, similarly to what is known in Galois connection based abstract interpretation [8,16]. As in abstract interpretation, completeness formalizes the notion of precision of an approximation. Completeness is defined wrt a pair $\langle A, B\rangle$ with A and B being respectively the input and output SFA that approximate the languages of the transduction. Backward completeness means that no loss of precision is introduced by approximating the input language L of a transducer with respect to the approximation of the transduction of L. In this case the approximate transduction of L provides the same language as the approximate transduction of an approximation of L. Consider, for example, a transducer T that given, a string σ in Σ^* removes the symbols $s \in \Sigma$ by replacing them with ϵ. Consider an SFA A that recognises the language $\mathscr{L}(A) = \{a\sigma \mid \sigma \in \Sigma^*\}$ and an SFA B that recognises the language $\mathscr{L}(B) = \{(a{+}b)\sigma \mid \sigma \in (\Sigma \smallsetminus \{s\})^*\}$. Then, the pair of SFAs $\langle A, B\rangle$ is not $\mathcal{B}$-complete for the transduction T. Indeed, applying transduction T to the strings recognized by A and then projecting the output in B we obtain the set of strings $\{a\sigma \mid \sigma \in (\Sigma \smallsetminus \{s\})^*\}$, while applying transduction T to the (concrete) set of any possible strings and then projecting the output in B we obtain the set of strings $\{(a + b)\sigma \mid \sigma \in (\Sigma \smallsetminus \{s\})^*\}$. Forward completeness [15] means instead that no loss of precision is introduced by approximating the output of the transaction of an approximate language $L^\sharp$ with respect to the concrete transaction of $L^\sharp$ itself. In the example above, if we consider an SFA B' recognizing $\mathscr{L}(B') = \{ab\sigma \mid \sigma \in (\Sigma \smallsetminus \{s\})^*\}$, we have that the pair of SFAs $\langle A, B'\rangle$ is not $\mathcal{F}$-complete for the transduction T. Indeed, applying transduction T to the strings recognized by A we obtain a set strictly containing $\mathscr{L}(B')$, namely the set containing all the strings without s starting with a (not only those starting with ab).

These two forms of completeness characterise the maximal precision achievable when transducing languages approximated by SFAs. We prove that it is possible to associate a pair of SFAs with any SFT (the input and output languages of the transducer) and conversely an SFT with any SFA. When $\mathcal{B}$-completeness is not satisfied, namely when the pair of SFAs $\langle A, B\rangle$ is not $\mathcal{B}$-complete for T, we characterize how to minimally expand the language recognized by the input SFA A or how to minimally reduce the language recognised by the output SFA B in order to achieve $\mathcal{B}$-completeness. For the example above, in order to achieve $\mathcal{B}$-completeness we can either add to $\mathscr{L}(A)$ the set of strings $\{b\sigma \mid \sigma \in \Sigma^*\}$ or remove from $\mathscr{L}(B)$ the strings in $\{b\sigma \mid \sigma \in (\Sigma \smallsetminus \{s\})^*\}$. A similar construction applies to $\mathcal{F}$-completeness. In the example, in order to achieve $\mathcal{F}$-completeness we can either remove from $\mathscr{L}(A)$ all those strings whose second symbol is not b, or add to $\mathscr{L}(B')$ all the strings (without s) starting with a. This result extends to symbolic transducers and symbolic automata the characterisation of complete abstractions in [16].

We apply our construction to the synthesis of sanitisers for cross-site scripting (XSS) and injection attack sanitisation in web application security. A script program manipulating strings P can be viewed as the combination T of transducers acting on a suitable language of allowed strings. If A is a language specifying a given attack, namely a set of strings that may lead the web application to bypass some access control, then whenever the inverse image of A by T is empty, then P is free from the attack specified in A. We extract the specification of a code sanitiser directly from the abstract interpretation of P viewed as an approximate transduction of a specification A of an expected attack in terms of SFA. Interestingly, this construction means that a script program P manipulating strings is unsafe with respect to the attack A if T is $\mathcal{B}$-incomplete with respect to A and an input SFA recognising the empty language. In this case, the extraction of the minimal sanitiser corresponds precisely to the expansion of the language of the SFA recognising the empty language towards $\mathcal{B}$-completeness with respect to A. This gives a minimality result with respect to language set inclusion, in the systematic derivation of script code sanitisation. We exemplify our idea with a simple example of sanitisation for a JavaScript-like pseudo code.

2 Background

Symbolic Finite Automata (SFA). We follow [12] in specifying symbolic automata in terms of effective Boolean algebra. Let $\mathcal{A} = \langle \mathfrak{D}_{\mathcal{A}}, \Psi_{\mathcal{A}}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg\rangle$ be an effective Boolean algebra, with domain elements in a r.e. set $\mathfrak{D}_{\mathcal{A}}$, a r.e. set of predicates $\Psi_{\mathcal{A}}$ closed under boolean connectives $\wedge$, $\vee$ and $\neg$. The semantics $[\![\cdot]\!] : \Psi_{\mathcal{A}} \longrightarrow \wp(\mathfrak{D}_{\mathcal{A}})$ is a partial recursive function such that $[\![\bot]\!] = \varnothing$, $[\![\top]\!] = \mathfrak{D}_{\mathcal{A}}$, and $\forall \varphi, \phi \in \Psi_{\mathcal{A}}$ we have that $[\![\varphi \vee \phi]\!] = [\![\varphi]\!] \cup [\![\phi]\!]$, $[\![\varphi \wedge \phi]\!] = [\![\varphi]\!] \cap [\![\phi]\!]$, and $[\![\neg\varphi]\!] = \mathfrak{D}_{\mathcal{A}} \smallsetminus [\![\varphi]\!]$. For $\varphi \in \Psi_{\mathcal{A}}$ we write $IsSat(\varphi)$ when $[\![\varphi]\!] \neq \varnothing$ and say that φ is *satisfiable*. $\mathcal{A}$ is decidable if $IsSat$ is decidable.

Definition 1 (Symbolic Finite Automata). *A SFA is $A = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$ where $\mathcal{A}$ is an effective Boolean algebra, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\Delta \subseteq Q \times \Psi_{\mathcal{A}} \times Q$ is a finite set of transitions.*

A transition in $A = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$ labeled φ from state p to state q, $(p, \varphi, q) \in \Delta$ is often denoted $p \xrightarrow{\varphi} q$. φ is called the *guard* of the transition. An a-move of an SFA A

is a transition $p \xrightarrow{\varphi} q$ such that $a \in [\![\varphi]\!]$, also denoted $p \xrightarrow{a} q$. The language recognized by a state $q \in Q$ in A is defined as:

$$\mathscr{L}_q(A) = \left\{a_1, \ldots, a_n \in \mathfrak{D}^*_{\mathcal{A}} \,\middle|\, \forall 1 \leq i \leq n.\ p_{i-1} \xrightarrow{a_i} p_i, p_0 = q,\ p_n \in F \right\}$$

hence $\mathscr{L}(A) = \mathscr{L}_{q_0}(A)$. In the following we denote the string elements in bold. Moreover, given two strings $\mathbf{s}_1, \mathbf{s}_2 \in S^*$ then we write $\mathbf{s}_1 \preceq \mathbf{s}_2$ when $\mathbf{s}_1$ is a prefix of $\mathbf{s}_2$, we denote with $\mathbf{s}_1 \cdot \mathbf{s}_2$ the string concatenation $\mathbf{s}_1\mathbf{s}_2$ and with $\mathbf{s}[n]$ the n-th symbol in $\mathbf{s}_1$. Consider $\mathbf{a} \in \mathcal{D}^*_{\mathcal{A}}$, we write $q \overset{\mathbf{a}}{\twoheadrightarrow} p$ to denote that state p is reachable from state q by reading the string $\mathbf{a}$.

The following terminology holds for SFA: A is *complete SFA* when all states hold an out-going a-move for any $a \in \mathfrak{D}$. A is *deterministic* whenever $p \xrightarrow{\varphi} q, p \xrightarrow{\beta} q' \in \Delta$: if $IsSat(\varphi \wedge \beta)$ then $q = q'$. A is *clean* if for all $p \xrightarrow{\varphi} q \in \Delta$: p is reachable from q_0 and $IsSat(\varphi)$. A is *normalized* if for all $p, q \in Q$: there is at most one move from p to q. A is *minimal* if it is deterministic, clean, normalized and for all $p, q \in Q$: $p = q \iff \mathscr{L}_q(A) = \mathscr{L}_p(A)$. In [12] the authors propose an efficient algorithm for SFA minimization. SFA can have ϵ-moves that can be eliminated in linear time [27]. Moreover, given two SFAs A and B it is possible to compute their union $A \oplus B$ such that $\mathscr{L}(A \oplus B) = \mathscr{L}(A) \cup \mathscr{L}(B)$, their product $A \times B$ such that $\mathscr{L}(A \times B) = \mathscr{L}(A) \cap \mathscr{L}(B)$, their difference $A - B$ such that $\mathscr{L}(A - B) = \mathscr{L}(A) \smallsetminus \mathscr{L}(B)$ [20].

Recently, it has been developed a general framework for abstracting SFAs [10]. Here SFAs form a domain ordered according to the language that they recognize, so an SFA is more concrete than another one if it recognizes a smaller language (with less noise). In particular, an SFA can be abstracted either by acting on the underlying Boolean algebra or on the automata. When abstracting at the level of Boolean algebra we can either approximate the domain of predicates (i.e., the syntax) or the domain of denotations (i.e., the semantics). In [10] there is a rigorous description of both syntactic and semantic abstraction of SFA and of their strong relation. The domain of SFA can be naturally used to represent properties of strings. As examples, consider the SFAs A and B in Fig. 1. In these SFAs the predicates are the label on the edges (we omit the label **true**), x denotes the following symbol read and the accepted language is the set of all the sequences of symbols, leading from the initial state p_0 to the final state (denoted with double line), such that each symbol satisfies the corresponding predicate. The SFA A, for instance, recognizes the language $\mathscr{L}(A) = \bigcup_{n \in \mathbb{N}} L_a^n$ where $L_a \triangleq \{<< x > \mid x \in \Sigma \smallsetminus \{<\}\}$, namely $\mathscr{L}(A)$ is a set of finite sequences of patterns of the form $<< x >$ with $x \in \Sigma$ is different from $<$. Similarly, $\mathscr{L}(B) = \bigcup_{n \in \mathbb{N}} L_b^n$, where $L_b \triangleq \{< x > \mid x \in \Sigma \smallsetminus \{<\}\}$.

Symbolic Finite Transducers (SFT). We follow [28] in the definition of SFT and of their background structure. Consider a background universe $\mathcal{U}$ which is a countable multi-carrier set equipped with a language of functions and relations with a fixed interpretation. We use σ, τ, γ to denote types, and $\mathcal{U}^\sigma$ to denote the elements of $\mathcal{U}$ that have type σ. In the following Σ refers to $\mathcal{U}^\sigma$ and Γ to $\mathcal{U}^\gamma$. We use $\mathbb{B}$ to denote the elements of boolean type, $\mathcal{U}^{\mathbb{B}} = \{true, false\}$, and $\mathbb{Z}$ for the integer type. Terms and formulas are defined by induction over the background language and are assumed to be well-typed. Terms of type $\mathbb{B}$ are treated as formulas. $t : \sigma$ denotes a term t of type σ, and $FV(t)$

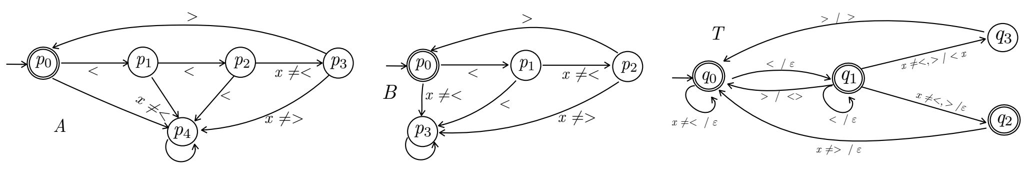


Fig. 1. An example of SFA A, SFA B and of SFT T.

denotes the set of its free variables. A term $t : \sigma$ is *closed* when $FV(t) = \emptyset$. Closed terms have a semantics $[\![t]\!]$. As usual $t[x/v]$ denotes the substitution of a variable $x : \tau$ with a term $v : \tau$. A λ-*term* f is an expression of the form $\lambda x.t$ where $x : \sigma$ is a variable and $t : \gamma$ is a term such that $FV(t) \subseteq \{x\}$. The λ-term f has type $\sigma \to \gamma$ and its semantics is a function $[\![f]\!] : \Sigma \to \Gamma$ that maps $a \in \Sigma$ to $[\![t[x/a]]\!] \in \Gamma$. Let f and g range over λ-terms. A λ-term of type $\sigma \to \mathbb{B}$ is called a σ-predicate.We use φ and ψ to denote σ-predicates. Given a σ-predicate φ, we write $a \in [\![\varphi]\!]$ for $[\![\varphi]\!](a) = \textbf{true}$. Moreover, $[\![\varphi]\!]$ can be seen as the subset of Σ that satisfies φ. We sometimes use $\mathcal{P}^\sigma$ to refer to the set of σ-predicates. We assume implicit β-reduction, namely given a λ-term $f = (\lambda x.t : \sigma \to \gamma)$ and a term $u : \sigma$, $f(u)$ stands for $t[x/u]$. φ is *unsatisfiable* when $[\![\varphi]\!] = \emptyset$ and *satisfiable* otherwise.

Definition 2 (Label Theory). *A label theory for $\sigma \to \gamma$ is associated with a effectively enumerable set of λ-terms of type $\sigma \to \gamma$ and a effectively enumerable set of σ-predicates that is effectively closed under Boolean operations and relative difference, i.e., $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \Sigma \setminus [\![\varphi]\!]$.*

A label theory Ψ is *decidable* if $IsSat(\varphi)$ is decidable for $\varphi \in \Psi$. We use X^* to denote the Kleene closure of a set X, and τ^* to denote the type of sequences over τ. $\mathbf{x}$ or equivalently $[x_0, \ldots, x_{k-1}]$ denote a sequence of length $|\mathbf{x}| = k \geq 0$, where $\mathbf{x}_i$ refers to the i-th element of $\mathbf{x}$ with $0 \leq i \leq k-1$.

Definition 3 (Symbolic Finite Transducers). *A SFT T over $\sigma \to \gamma$ is a tuple $T = \langle Q, q_0, F, R \rangle$, 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 and R is a set of rules $(p, \varphi, \mathbf{f}, q)$ where $p, q \in Q$, φ is a σ-predicate and $\mathbf{f}$ is a sequence of λ-terms over a given label theory for $\sigma \to \gamma$.*

A rule $(p, \varphi, \mathbf{f}, q)$ of an SFT T is denoted as $p \xrightarrow{\varphi/\mathbf{f}}_T q$ where φ is called guard. We omit the index T when it is clear from the context. The sequence of λ-terms $\mathbf{f} : (\sigma \to \gamma)^*$ can be treated as a function $\lambda x.[\mathbf{f}_0(x), \ldots, \mathbf{f}_k(x)]$ where $k = |\mathbf{f}| - 1$. Concrete transitions are represented as rules. Consider $p, q \in Q$, $a \in \Sigma$ and $\mathbf{b} \in \Gamma^*$ then:

$$p \xrightarrow{a/\mathbf{b}}_T q \iff p \xrightarrow{\varphi/\mathbf{f}}_T q \in R : a \in [\![\varphi]\!] \wedge \mathbf{b} = [\![\mathbf{f}]\!](a)$$

Given two sequences $\mathbf{a} \in \Sigma^*$ and $\mathbf{b} \in \Gamma^*$, we write $q \overset{\mathbf{a}/\mathbf{b}}{\twoheadrightarrow} p$ when p is reachable from q reading $\mathbf{a}$ and producing in output $\mathbf{b}$. More specifically when there exists a path of transitions from q to p in T with input sequence $\mathbf{a}$ and output sequence $\mathbf{b}$, where $\mathbf{b} = \mathbf{b}^0 \cdot \mathbf{b}^1 \cdots \mathbf{b}^n$ with $n = |\mathbf{a}| - 1$ and $\mathbf{b}^i$ denoting a subseqeunce of $\mathbf{b}$, such that:

$$p = p_0 \xrightarrow{\mathbf{a}_0/\mathbf{b}^0} p_1 \xrightarrow{\mathbf{a}_1/\mathbf{b}^1} p_2 \ldots p_n \xrightarrow{\mathbf{a}_n/\mathbf{b}^n} p_{n+1} = q$$

SFTs can have ϵ-transitions and they can be eliminated following a standard procedure. We assume $p \overset{\epsilon/\epsilon}{\twoheadrightarrow} p$ for all $p \in Q$.

Definition 4 (Transduction). *The transduction of an SFT T over $\sigma \to \gamma$ is a function $\mathfrak{T}_T : \Sigma^* \to \wp(\Gamma^*)$ where: $\mathfrak{T}_T(\mathbf{a}) \triangleq \{\mathbf{b} \in \Gamma^* \mid \exists q \in F : q_0 \overset{\mathbf{a}/\mathbf{b}}{\twoheadrightarrow} q\}$.*

An SFT is *single-valued* when $|\mathfrak{T}_T(\mathbf{a})| \leq 1$ for all $\mathbf{a} \in \Sigma^*$. In [28] the authors define the notion of SFTs *equivalence* by saying that two SFTs T and R are equivalent ($T \equiv R$) when $\{\mathbf{a} \in \Sigma^* \mid \mathfrak{T}_T(\mathbf{a}) \neq \emptyset\} = \{\mathbf{a} \in \Sigma^* \mid \mathfrak{T}_R(\mathbf{a}) \neq \emptyset\}$ (domain equivalence), and $\forall \mathbf{a} \in \mathscr{L}_{\mathcal{I}}(T)$ we have $\mathfrak{T}_T(\mathbf{a}) = \mathfrak{T}_R(\mathbf{a})$ (partial equivalence). Equivalence of SFTs is decidable when SFTs are single-valued, [28] provides an algorithm for checking equivalence.

An example of SFT is given on the right, in Fig. 1. This SFT is a slight modification of an SFT from [28]. The considered SFT T reads a string of elements in Σ and returns the string of patterns $< x >$ and $<>$ that it contains, with $x \neq <, >$.

3 Approximating transduction

Let us denote with SFA^σ the set of SFAs that recognize sequences of symbols of type σ and with A^σ an element of SFA^σ. This means that given $A^\sigma \in \text{SFA}^\sigma$ we have that $\mathscr{L}(A^\sigma) \in \wp(\Sigma^*)$, which means that the domain of denotations of the underlying Boolean algebra is Σ. Let $\text{SFT}^{\sigma/\gamma}$ be the set of SFTs over $\sigma \to \gamma$ and let us denote with $T^{\sigma/\gamma}$ an element of $\text{SFT}^{\sigma/\gamma}$. In the following we will omit the superscript denoting the type of SFA and SFT when it is not needed or when it is clear form the context. In this section, we want to show how the string transformation expressed by $T^{\sigma/\gamma}$ can been approximated as a transformation from SFA^σ to SFA^γ. Indeed, the SFT $T^{\sigma/\gamma}$ is a language transformer that, given a language of strings over Σ^*, returns a language of strings over Γ^*. By approximating the input and output languages of strings manipulated by an SFT in the domain of SFAs, we can view SFTs as SFAs transformers, thus approximating the SFT computation on the domain of SFAs.

We associate with an SFT $T^{\sigma/\gamma}$ an input language and output language that collect respectively the input strings in Σ^* that produce an output when processed by T, and the output strings in Γ^* generated by T.

Definition 5 (Input/Output language of an SFT). *Given an SFT $T^{\sigma/\gamma} = \langle Q, q_0, F, R\rangle$, we define its input language $\mathscr{L}_{\mathcal{I}}(T)$ and output language $\mathscr{L}_{\mathcal{O}}(T)$ as:*

- $\mathscr{L}_{\mathcal{I}}(T) \triangleq \{\mathbf{a} \in \Sigma^* \mid \mathfrak{T}_T(\mathbf{a}) \neq \emptyset\}$
- $\mathscr{L}_{\mathcal{O}}(T) \triangleq \{\mathbf{b} \in \Gamma^* \mid \mathbf{b} \in \mathfrak{T}_T(\mathbf{a}), \mathbf{a} \in \mathscr{L}_{\mathcal{I}}(T)\}$

The proposed notion of input language corresponds to the notion of domain of an SFT introduced in [28]. Observe that, given an SFT T it is possible to build two SFA $\text{SFA}_{\mathcal{I}}(T)$ and $\text{SFA}_{\mathcal{O}}(T)$ that recognize respectively the input and output language of T.

Definition 6. *Given an SFT $T^{\sigma/\gamma} = \langle Q, q_0, F, R\rangle$ we define:*

- $\textsc{sfa}_{\mathcal{I}}(T) \triangleq \langle \mathcal{A}, Q, q_0, F, \Delta_{\text{SFA}_{\mathcal{I}}(T)}\rangle$, *where* $\mathcal{A} = \langle \Sigma, \mathcal{P}^{\sigma}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg\rangle$ *and rules defined as:* $\Delta_{\text{SFA}_{\mathcal{I}}(T)} \triangleq \{(p, \varphi, q) \mid (p, \varphi, \mathbf{f}, q) \in R\}$.
- $\textsc{sfa}_{\mathcal{O}}(T) \triangleq \langle \mathcal{A}, Q, q_0, F, \Delta_{\text{SFA}_{\mathcal{O}}(T)}\rangle$, *where* $\mathcal{A} = \langle \Gamma^*, \mathcal{P}^{\gamma *}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg\rangle$ *and rules defined as:* $\Delta_{\text{SFA}_{\mathcal{O}}(T)} \triangleq \{(p, \mathbf{f}(\varphi), q) \mid (p, \varphi, \mathbf{f}, q) \in R\}$, *where* $[\![\mathbf{f}(\varphi)]\!] \triangleq \{[\![\mathbf{f}]\!](a) \mid a \in [\![\varphi]\!]\}$.

Observe that the language of $\textsc{sfa}_{\mathcal{O}}(T)$ is an element of $\wp(\Gamma^{**})$ because at each step the SFA $\textsc{sfa}_{\mathcal{O}}(T)$ recognizes a string of Γ^*. We define function $seq : \wp(\Gamma^{**}) \to \wp(\Gamma^*)$ that transforms a set of strings of strings into a set of strings as the additive lift of:

$$seq(\mathbb{b}) \triangleq \begin{cases} \epsilon & \text{if } \mathbb{b} = \epsilon \\ \mathbf{b} \cdot seq(\mathbb{b}') & \text{if } \mathbb{b} = \mathbf{b}\mathbb{b}' \end{cases}$$

where $\mathbb{b}$ denotes a string in Γ^{**} and $\mathbf{b}$ a string in Γ^*. Consider for instance, again the SFT T in Fig. 1. Then we obtain the input SFA $\textsc{sfa}_{\mathcal{I}}(T)$, on the left in Fig. 2, simply by erasing the output function information (and minimizing if necessary). In particular, we can observe that $\textsc{sfa}_{\mathcal{I}}(T)$ accepts any possible string, namely is it equivalent to an SFA with only one final state with only one self edge labeled with **true**. The output SFA $\textsc{sfa}_{\mathcal{O}}(T)$ keeps only the output function transformation. The minimized output SFA of T is given in Fig. 2 (top-right). Note that, the accepted strings formally are sequences of strings, since the edge connecting q_1 with q_3 accepts a string of two symbols instead of one single symbol. For this reason we need the function $seq(\mathbb{b})$ such that, for instance $seq(\mathbb{b})((<, s), >) = (<, s, >)$.

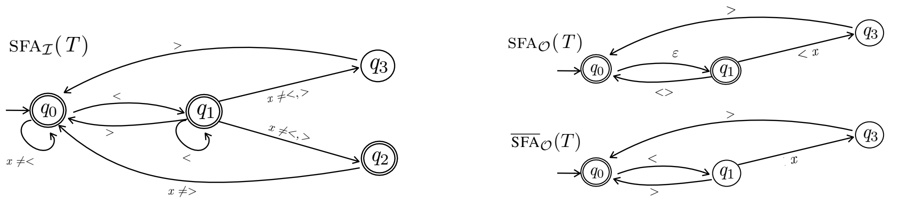


Fig. 2. The input and out SFA of the SFT T.

Lemma 1. *Given an SFT $T^{\sigma/\gamma}$, the followings hold:*

1. $\mathscr{L}_{\mathcal{I}}(T) = \mathscr{L}(\textsc{sfa}_{\mathcal{I}}(T)) \in \wp(\Sigma^*)$
2. $\mathscr{L}_{\mathcal{O}}(T) = seq(\mathscr{L}(\textsc{sfa}_{\mathcal{O}}(T))) \in \wp(\Gamma^*)$

For instance, if we consider again the SFT T in Fig. 1, $\mathscr{L}_{\mathcal{I}}(T) = \Sigma^*$ while $\mathscr{L}_{\mathcal{O}}(T) = \{(<, x, >) \mid x \neq' <'\}$.

Note that, by definition, the function $\mathbf{f}$ is a lambda term mapping each symbol in Σ satisfying the edge guard in a sequence of symbols in Γ, i.e., $\forall x \in [\![\varphi]\!] \subseteq \Sigma$ we have that $\mathbf{f}(x) = \mathbf{f}_0(x) \dots \mathbf{f}_k(x)$ with $\forall i \in [0, k]$. $\mathbf{f}_i(x) \in \Gamma$. This characteristic of transducers allows us to uniquely construct an equivalent SFA whose language is precisely $\mathscr{L}_{\mathcal{O}}(T)$.

Proposition 1. *Given a SFT T, we can always construct, starting from* $\mathrm{SFA}_{\mathcal{O}}(T)$ *the SFA* $\overline{\mathrm{SFA}}_{\mathcal{O}}(T) = \langle \overline{\mathcal{A}}, \overline{Q}, q_0, \overline{F}, \Delta_{\overline{\mathrm{SFA}}_{\mathcal{O}}(T)} \rangle$, *where* $\overline{\mathcal{A}} = \langle \Gamma, \mathcal{P}^{\gamma}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$ *such that* $\mathscr{L}(\overline{\mathrm{SFA}}_{\mathcal{O}}(T)) = seq(\mathscr{L}(\mathrm{SFA}_{\mathcal{O}}(T)))$.

The idea of this transformation is that of splitting each edge labeled with a string $s_1 s_2 \dots s_n \in \Gamma^*$ in n edges, each one labeled with one symbol $s_i \in \Gamma$, thus adding $n-1$ new states. In this transformation we do not change neither initial nor final states. For instance, in the example on the right in Fig. 2 we split the edge $q_1 \xrightarrow{<\, x} q_3$ in $q_1 \xrightarrow{<} q_1'$ and $q_1' \xrightarrow{x} q_3$. Then we erase ε-transition [27], when necessary (in this case we could change the set of final states), and we can finally minimize the resulting SFA, if possible [12]. In Fig. 2 (bottom-right) we have the SFA resulting by this transformation of $\mathrm{SFA}_{\mathcal{O}}(T)$.

We can also define how an SFA A can be associated with an SFT $\mathcal{T}_{\mathcal{O}}(A)$ whose input and output language is the one recognized by the SFA.

Definition 7. *Given an SFA* $A^{\sigma} = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$ *over* $\mathcal{A} = \langle \Sigma, \mathcal{P}^{\sigma}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$. *We define the output SFT over $\sigma \to \sigma$ associated with A as* $\mathcal{T}_{\mathcal{O}}(A) \triangleq \langle Q, q_0, F, R^{id} \rangle$ *where the set of rules is* $R^{id} \triangleq \{(p, \varphi, id, q) \mid (p, \varphi, q) \in R\}$.

Lemma 2. *Given an SFA A, the followings hold:*

1. $\mathscr{L}(A) = \mathscr{L}_{\mathcal{I}}(\mathcal{T}_{\mathcal{O}}(A)) = \mathscr{L}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A))$
2. $\mathfrak{T}_{\mathcal{T}_{\mathcal{O}}(A)}(\mathbf{a}) = \begin{cases} \mathbf{a} \text{ if } \mathbf{a} \in \mathscr{L}(A) \\ \emptyset \text{ otherwise} \end{cases}$

These definitions allow us to associate SFAs with SFTs and vice-versa and they will be usefull in providing a methodology for seeing SFTs as SFAs transformers. Let us recall that the definition of composition of SFTs. According to [28] we define the composition of two transductions $\mathfrak{T}_1$ and $\mathfrak{T}_2$ as:

$$\mathfrak{T}_1 \diamond \mathfrak{T}_2 \triangleq \lambda \mathbf{b}. \bigcup_{\mathbf{a} \in \mathfrak{T}_1(\mathbf{b})} \mathfrak{T}_2(\mathbf{a})$$

Observe that the composition $\diamond$ applies first $\mathfrak{T}_1$ and then $\mathfrak{T}_2$ and that single-value property is preserved by composition. Two label theories $\sigma \to \tau$ and $\tau \to \gamma$ are *composable* if there exists a label theory Ψ for $\sigma \to \gamma$ such that: (1) if $f : \sigma \to \tau$ and $g : \tau \to \gamma$ are λ-terms then $\lambda x.g(f(x))$ is a valid λ-term in Ψ, (2) if φ is a τ-predicate and $f : \sigma \to \tau$ is a λ-term then $\lambda x.\varphi(f(x))$ is a valid σ-predicate in Ψ. It has been proved that if T_1 and T_2 are SFTs over composable label theories, then there exists an SFT $T_1 \diamond T_2$ that is obtained effectively from T_1 and T_2 such that $\mathfrak{T}_{T_1 \diamond T_2} = \mathfrak{T}_1 \diamond \mathfrak{T}_2$ [28]. A constructive characterization of the composition of SFA can be found in [18]. We report in the following an algebra over SFTs that is obtained by extending the one in [28] with the association of SFAs to SFTs and vice versa.

$$\begin{array}{lll}
\sigma, \tau, \gamma & ::= & \text{types} \\
\mathit{sfa}^{\sigma} & ::= & \text{explicit dfn of an SFA over } \sigma \\
\mathit{sft}^{\sigma/\gamma} & ::= & \text{explicit dfn of an SFT over } \sigma \to \gamma \\
A^{\sigma} & ::= & \mathit{sfa}^{\sigma} \mid A^{\sigma} - A^{\sigma} \mid A^{\sigma} \times A^{\sigma} \mid A^{\sigma} \oplus A^{\sigma} \mid \textsc{sfa}_{\mathcal{I}}(T^{\sigma/\gamma}) \mid \textsc{sfa}_{\mathcal{O}}(T^{\gamma/\sigma}) \mid \overline{\textsc{sfa}}_{\mathcal{O}}(T^{\gamma/\sigma}) \\
T^{\sigma/\gamma} & ::= & \mathit{sft}^{\sigma/\gamma} \mid T^{\sigma/\tau} \diamond T^{\tau/\gamma} \mid \mathcal{T}_{\mathcal{O}}(A^{\sigma})
\end{array}$$

Now we have all we need for specifying SFTs as SFAs transformers. In order to use an SFT to transform an SFA we need that the two work on the same domain of elements. More specifically, an SFT $T^{\sigma/\gamma}$ transforms an SFA A^{σ} into an SFA B^{γ}. Intuitively, applying an SFT $T^{\sigma/\gamma}$ to the strings recognized by an SFA A^{σ} means to compute $\mathfrak{T}_T$ on the strings in $\mathscr{L}(A^{\sigma})$. Interestingly this precisely corresponds to the following composition of SFTs: $\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma}$. Indeed, in this composition the language recognized by the SFA A^{σ} becomes the input language of the SFT $T^{\sigma/\gamma}$. Observe that this composition is equivalent to the operation of domain restriction defined in [28].

Proposition 2. *The transduction of $\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma}$ is:*

$$\mathfrak{T}_{\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma}} = \lambda \mathbf{b}. \begin{cases} \mathfrak{T}_T(\mathbf{b}) & \textit{if } \mathbf{b} \in \mathscr{L}(A) \\ \emptyset & \textit{otherwise} \end{cases}$$

and $\mathscr{L}_{\mathcal{I}}(\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma}) = \mathscr{L}(A)$ and $\mathscr{L}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma}) = \{\mathbf{b} \in \Gamma^* \mid \mathbf{b} \in \mathfrak{T}_T(\mathbf{a}), \mathbf{a} \in \mathscr{L}(A)\}$

Observe that by computing the SFA that recognizes the output language of $\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma}$ we obtain the SFA obtained by transforming A^{σ} with $T^{\sigma/\gamma}$.

Proposition 3. $\mathscr{L}(\overline{\textsc{sfa}}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma})) = \{\mathbf{b} \in \Gamma^* \mid \mathbf{b} \in \mathfrak{T}_T(\mathbf{a}), \mathbf{a} \in \mathscr{L}(A)\}$

Thus, an SFT $T^{\sigma/\gamma}$ transforms an SFA A^{σ} into the SFA $\overline{\textsc{sfa}}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A^{\sigma}) \diamond T^{\sigma/\gamma})$, as shown in the following example where we consider the SFT $T^{\sigma/\sigma}$ and the SFA A^{σ} in Fig. 1. Now we compute the SFT $\mathcal{T}_{\mathcal{O}}(A^{\sigma})$ whose output language is exactly the

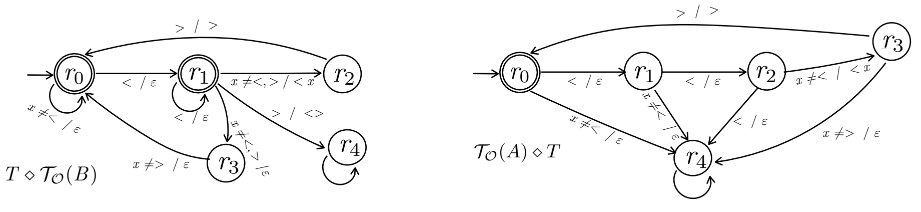


Fig. 3. Application of SFT $T^{\sigma/\sigma}$ to SFA A and to B

language recognized by the SFA A^{σ}. Next, we apply the SFT to the SFA by computing the SFT given by the composition $\mathcal{T}_{\mathcal{O}}(A) \diamond T$, depicted (minimized) on the right of Fig. 3 (on the left we have another example of composition). Finally, we extract

$\overline{\text{SFA}}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A) \diamond T)^{\sigma^*}$, which recognizes the output language of $\mathcal{T}_{\mathcal{O}}(A) \diamond T$. As expected, the output SFA is such that $\mathscr{L}(\overline{\text{SFA}}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A) \diamond T)^{\sigma^*})$ is the language $\bigcup_{n \in \mathbb{N}} L^n$ where we define $L \triangleq \{< x > \mid x \in \Sigma \smallsetminus \{<\}\}$.

4 Completeness

As in abstract interpretation-based static analysis, we wonder when the approximated computation of SFTs on the abstract domain of SFAs is precise, namely when computing on abstract values or abstracting the concrete computation provides the same loss of precision. In abstract interpretation theory, the ideal situation of no loss of precision between the concrete and abstract computation is called completeness [7,8]. There are two forms of completeness: *backward completeness* (denoted $\mathcal{B}$-completeness) and *forward completeness* (denoted $\mathcal{F}$-completeness). $\mathcal{B}$-completeness requires that the concrete and abstract computation are equivalent when we compare their outputs on the abstract domain, while $\mathcal{F}$-completeness requires that the concrete and abstract computation are equivalent when we consider abstractions of the inputs. It has been proved that, both forms of completeness, are properties of the abstract domain, and that it is possible to iteratively modify an abstract domain in order to make it complete for a given function [16]. In the following we introduce the notion of $\mathcal{B}$-completeness and $\mathcal{F}$-completeness of SFAs wrt a given SFT. When completeness is not satisfied we provide SFAs transformers that allow to achieve it by minimally modifying the language recognized by the SFAs. We consider the following notion of completeness:

Definition 8. *Let $T = \langle Q, q_0, F, R\rangle$ be an SFT and $A = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$ and $B = \langle \mathcal{A}', Q', q_0', F', \Delta'\rangle$ be SFAs.*

- *$\langle A, B\rangle$ is $\mathcal{F}$-complete for T if $\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B) \equiv \mathcal{T}_{\mathcal{O}}(A) \diamond T$*
- *$\langle A, B\rangle$ is $\mathcal{B}$-complete for T if $\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B) \equiv T \diamond \mathcal{T}_{\mathcal{O}}(B)$.*

The following result characterizes $\mathcal{F}$-completeness and $\mathcal{B}$-completeness of SFAs wrt SFTs. In order to characterize the $\mathcal{B}$-completeness we need the inverse image of the transduction function. Let us define the *inverse transduction* of an SFT T as $\mathfrak{T}_T^{-}(\mathbf{b}) \triangleq \{\mathbf{a} \mid \mathbf{b} \in \mathfrak{T}_T(\mathbf{a})\}$. It is worth noting that, by construction and definition, $\forall \mathbf{b} \in \mathscr{L}_{\mathcal{O}}(T)$ we have $\mathfrak{T}_T^{-}(\mathbf{b}) \subseteq \mathscr{L}_{\mathcal{I}}(T)$.

Theorem 1. *Let A, B be SFA and T an SFT*

1. *$\langle A, B\rangle$ is $\mathcal{F}$-complete for T iff $\forall \mathbf{a} \in \mathscr{L}(A).\ \mathfrak{T}_T(\mathbf{a}) \subseteq \mathscr{L}(B)$;*
2. *$\langle A, B\rangle$ is $\mathcal{B}$-complete for T iff $\forall \mathbf{b} \in \mathscr{L}(B).\ \mathfrak{T}_T^{-}(\mathbf{b}) \subseteq \mathscr{L}(A)$;*

Let us consider some examples of complete/incomplete SFAs for the SFT T depicted in Fig. 1. In particular, we consider the SFA S_0 and S_1 in Fig. 4. Let us verify whether the pair $\langle S_0, B\rangle$ is $\mathcal{B}$-complete for T. In this case, we observe that any string $\mathbf{b} =< x_1 >< x_2 > \dots < x_n > \in \mathscr{L}(B)$ then $\mathfrak{T}_T^{-}(\mathbf{b})$ has the form $v_1\, w_1\ < x_1 > \dots v_n\, w_n\ < x_n >$, where, for each $i \in [0, n]$, $v_i \in (\Sigma \smallsetminus \{<\})^*$ is any sequence of symbols different from $<$, $w_i \in \{<\}^*$ is an arbitrarily long sequence of $<$, and $x_i \in \Sigma \smallsetminus \{<\}$ is a symbol. It is immediate to observe that such strings are included in the language $\mathscr{L}(S_0)$, hence

$\langle S_0, B\rangle$ is $\mathcal{B}$-complete for T. Let us consider now the SFA S_1, in this case we can observe that $\mathfrak{T}_T^{-}(\mathbf{b})$ is not contained in $\mathscr{L}(S_1)$, since if a strings starts with symbols different from $<$, then the string cannot be accepted by S_1. In this case, we have that $\langle S_1, B\rangle$ is not $\mathcal{B}$-complete for T, because the input language of $\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B)$ is different from the input language of $T \diamond \mathcal{T}_{\mathcal{O}}(B)$.
Let us consider now $\mathcal{F}$-completeness, and let us check whether the pair $\langle A, S_1\rangle$ is $\mathcal{F}$-complete for T. In particular, we observe that any string $\mathbf{a} =<< x_1 ><< x_2 \dots << x_n >\in \mathscr{L}(A)$ is transformed by T in the sequence $\mathfrak{T}_T(\mathbf{a}) =< x_1 >< x_2 > \dots < x_n >$, which is clearly accepted by S_1. Hence, we can say that $\langle A, S_1\rangle$ is $\mathcal{F}$-complete for T. If we consider, instead, S_2, then we can observe that $\mathfrak{T}_T(\mathbf{a}) \notin \mathscr{L}(S_1)$ if there exists at least a value $i \in [0, n]$ such that $x_i \notin \mathbb{N}$, hence we can conclude that $\langle A, S_2\rangle$ is not $\mathcal{F}$-complete for T because the output language of $\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B)$ is different from the one of $\mathcal{T}_{\mathcal{O}}(A) \diamond T$.

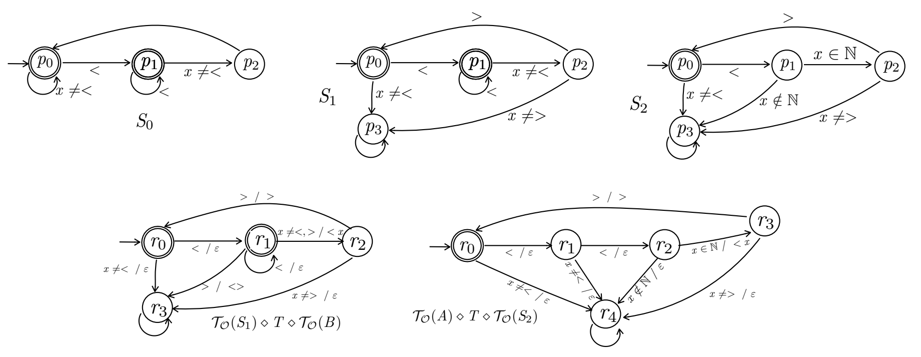


Fig. 4. Completeness examples.

Assume to have a pair of SFAs $\langle A, B\rangle$ that is not $\mathcal{B}(\mathcal{F})$-complete for an SFT T. In this case, as done in abstract-interpretation based static analysis [16], we would like to define SFAs transformers that force $\mathcal{B}(\mathcal{F})$-completeness of $\langle A, B\rangle$ for T. The idea is to transform either the input SFA A or the output SFA B in order to satisfy the completeness conditions. Consider a pair of SFAs $\langle A, B\rangle$ and an SFT T such that the $\mathcal{B}$-completeness condition is not satisfied: $\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B) \not\equiv T \diamond \mathcal{T}_{\mathcal{O}}(B)$. By Theorem 1 ,this loss of $\mathcal{B}$-completeness happens iff $\exists \mathbf{b} \in \mathscr{L}(B) : \mathfrak{T}^{-1}(\mathbf{b}) \not\subseteq \mathscr{L}(A)$. We have to possible ways to force $\mathcal{B}$-completeness:

- add to the language recognized by the SFA A the strings that $\mathfrak{T}_T$ maps to $\mathscr{L}(B)$,
- remove from the language recognized by the SFA B the strings obtained through $\mathfrak{T}_T$ from strings that do not belong to $\mathscr{L}(A)$.

Hence, in order to gain $\mathcal{B}$-completeness we can either expand the language of A or reduce the language of B. We would like to define a pair of SFA transformers that, given a pair of SFAs $\langle A, B\rangle$ and an SFT T for which $\mathcal{B}$-completeness does not hold, modify the input SFA A or the output SFA B in order to gain completeness by minimally expanding or reducing the language recognized respectively by A and B. Of course there are many SFAs over the same Boolean algebra that can recognize the same language. For this reason we consider the domain of SFAs up to language equivalence. More formally, we define a equivalence relation $\doteq$ over SFAs such that $A \doteq B \;\Leftrightarrow\; \mathscr{L}(A) = \mathscr{L}(B)$, and we denote an equivalence class as $[A]_{\doteq} = \{A \mid \mathscr{L}(A) = \mathscr{L}(B)\}$. We define the domain of SFAs up to language equivalence as follows: $\textsc{sfa}_{\doteq} \triangleq \{[A]_{\doteq} \mid A \in \textsc{sfa}\}$. We denote with $\textsc{sfa}^{\sigma}_{\doteq}$ the domain of SFAs over σ up to language equivalence. The following auxiliary functions define the language that has to be added or removed in order to gain $\mathcal{B}$-completeness.

Definition 9. *Let $T^{\sigma/\gamma}$ be an SFT we define the following functions:*

– $\mathfrak{s}^{\mathcal{B}}_{T} : \textsc{sfa}^{\sigma}_{\doteq} \rightarrow \wp(\Sigma^*)$ *such that* $\mathfrak{s}^{\mathcal{B}}_{T}([M]_{\doteq}) \triangleq \mathscr{L}_{\mathcal{I}}(T \diamond \mathcal{T}_{\mathcal{O}}(M))$
– $\mathfrak{c}^{\mathcal{B}}_{T} : \textsc{sfa}^{\sigma}_{\doteq} \rightarrow \wp(\Gamma^*)$ *such that* $\mathfrak{c}^{\mathcal{B}}_{T}([M]_{\doteq}) \triangleq \{s \in \Gamma^* \mid \mathfrak{T}^{-1}_{T}(s) \not\subseteq \mathscr{L}(M)\}$

The following result shows that the above definition is well-defined, namely that the computation of $\mathfrak{s}^{\mathcal{B}}_{T}$ and of $\mathfrak{c}^{\mathcal{B}}_{T}$ does not depend on the particular element chosen for representing the equivalence class.

Proposition 4. *Given $T^{\sigma/\gamma}$ and $[M]_{\doteq} \in \textsc{sfa}^{\sigma}_{\doteq}$ then $\forall M', M'' \in [M]_{\doteq}$ we have that $\mathfrak{s}^{\mathcal{B}}_{T}([M']_{\doteq}) = \mathfrak{s}^{\mathcal{B}}_{T}([M'']_{\doteq})$ and $\mathfrak{c}^{\mathcal{B}}_{T}([M']_{\doteq}) = \mathfrak{c}^{\mathcal{B}}_{T}([M'']_{\doteq})$.*

From Theorem 1 it is clear that the notion of completeness of SFA wrt a pair of SFAs depends on the languages recognized by the SFAs. For this reason in the following we provide completeness transformers that work on the domain of SFAs up to language equivalence. Given a pair of SFAs $\langle A, B\rangle$ and an SFT T for which $\mathcal{B}$-completeness does not hold, we define the *$\mathcal{B}$-complete shell* of the equivalence class $[A]_{\doteq}$ wrt T and B as the equivalence class of SFAs that recognize the language that minimally expand the language recognized by $[A]_{\doteq}$ in order to gain $\mathcal{B}$-completeness, and the *$\mathcal{B}$-complete core* of the equivalence class $[B]_{\doteq}$ wrt T and A as the class of SFAs that recognize the language that minimally reduces the language recognized by $[B]_{\doteq}$ in order to gain $\mathcal{B}$-completeness.

Definition 10. *Consider a pair of SFAs $\langle A^{\sigma}, B^{\gamma}\rangle$ and an SFT $T^{\sigma/\gamma}$ such that the $\mathcal{B}$-completeness condition is not satisfied. We define the following transformers:*

- *$\mathcal{B}$-complete shell transformer $\mathcal{ST}^{\mathcal{B}}_{T,B} : \textsc{sfa}^{\sigma}_{\doteq} \rightarrow \textsc{sfa}^{\sigma}_{\doteq}$ such that*

$$\mathcal{ST}^{\mathcal{B}}_{T,B}([A]_{\doteq}) \triangleq \{M \in \textsc{sfa}^{\sigma} \mid \mathscr{L}(M) = \mathscr{L}(A) \cup \mathfrak{s}^{\mathcal{B}}_{T}(B)\} \in \textsc{sfa}^{\sigma}_{\doteq}$$

- *$\mathcal{B}$-complete core transformer $\mathcal{CT}^{\mathcal{B}}_{T,A} : \textsc{sfa}^{\gamma}_{\doteq} \rightarrow \textsc{sfa}^{\gamma}_{\doteq}$ such that*

$$\mathcal{CT}^{\mathcal{B}}_{T,A}([B]_{\doteq}) \triangleq \{M \in \textsc{sfa}^{\gamma} \mid \mathscr{L}(M) = \mathscr{L}(B) \smallsetminus \mathfrak{c}^{\mathcal{B}}_{T}(A)\} \in \textsc{sfa}^{\gamma}_{\doteq}$$

Proposition 5. *Given a pair of SFAs $\langle A^{\sigma}, B^{\gamma}\rangle$ and an SFT $T^{\sigma/\gamma}$, we have that:*

- *$\forall A', A'' \in [A]_{\doteq}$ we have that $\mathcal{ST}^{\mathcal{B}}_{T,B}([A']_{\doteq}) = \mathcal{ST}^{\mathcal{B}}_{T,B}([A'']_{\doteq})$*
- *$\forall B', B'' \in [B]_{\doteq}$ we have that $\mathcal{CT}^{\mathcal{B}}_{T,A}([B']_{\doteq}) = \mathcal{CT}^{\mathcal{B}}_{T,A}([B'']_{\doteq})$*

The following result provides a characterization of an SFA in the equivalence class of $\mathcal{ST}^{\mathcal{B}}_{T,B}([A]_{\doteqdot})$ and of an SFA in the equivalence class $\mathcal{CT}^{\mathcal{B}}_{T,A}([B]_{\doteqdot})$ that proves that these classes are not empty, namely that in both cases there exists an SFA that precisely recognizes the desired language.

Proposition 6. *Consider a pair of SFAs $\langle A^\sigma, B^\gamma \rangle$ and an SFT $T^{\sigma/\gamma}$ we have that:*

- $A \oplus \text{SFA}_{\mathcal{I}}(T \diamond \mathcal{T}_{\mathcal{O}}(B)) \in \mathcal{ST}^{\mathcal{B}}_{T,B}([A]_{\doteqdot})$
- $\text{SFA}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B)) \in \mathcal{CT}^{\mathcal{B}}_{T,A}([B]_{\doteqdot})$

The following result proves that the complete shell and core transformers induce $\mathcal{B}$-completeness by minimally modifying the language of the input and output SFA.

Corollary 1. *Consider a pair of SFAs $\langle A^\sigma, B^\gamma \rangle$ and an SFT $T^{\sigma/\gamma}$ such that the $\mathcal{B}$-completeness condition is not satisfied:*

- $\forall A' \in \mathcal{ST}^{\mathcal{B}}_{T,B}([A]_{\doteqdot})$ *we have that $\langle A', B \rangle$ is $\mathcal{B}$-complete for T and $\mathscr{L}(A')$ minimally expands $\mathscr{L}(A)$ in order to gain $\mathcal{B}$-completeness*
- $\forall B' \in \mathcal{CT}^{\mathcal{B}}_{T,A}([B]_{\doteqdot})$ *we have that $\langle A, B' \rangle$ is $\mathcal{B}$-complete for T and $\mathscr{L}(B')$ minimally reduces $\mathscr{L}(B)$ in order to gain $\mathcal{B}$-completeness*

Consider, for instance the pair $\langle S_1, B \rangle$ $\mathcal{B}$-incomplete for T (Fig. 4), in order to minimally transform S_1 for inducing $\mathcal{B}$-completeness we should add to $\mathscr{L}(S_1)$ the language $\mathscr{L}_{\mathcal{I}}(T \diamond \mathcal{T}_{\mathcal{O}}(B))$ (see Fig. 3), namely we fall in the equivalence class of an SFA such as S_1' accepting also strings starting with a sequence of symbols different from $<$ (on the left of Fig. 5). We can observe that $\mathscr{L}(S_1') \subset \mathscr{L}(S_0)$ since we minimally transform the language for gaining completeness. Unfortunately, in this case the core is not meaning-

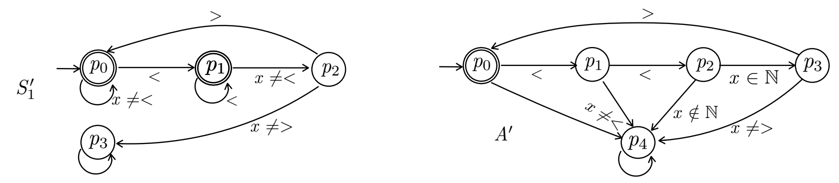


Fig. 5. $\mathcal{B}$-completeness shell example and $\mathcal{F}$-completeness core example.

ful since the only $\mathcal{B}$-complete transformation of B reducing the language would take the empty language since any string in $\mathscr{L}(B)$ is obtained also as transformation of a string not in $\mathscr{L}(S_1)$, starting with symbols different from $<$. Dual reasoning holds for $\mathcal{F}$-completeness. Also in this case we can provide a pair of functions that define the languages that have to be either added or removed on order to gain $\mathcal{F}$-completeness.

Definition 11. *Let $T^{\sigma/\gamma}$ be an SFT we define the following functions:*

- $\mathfrak{s}^{\mathcal{F}}_T : \text{SFA}^{\sigma}_{\doteqdot} \rightarrow \wp(\Gamma^*)$ *such that* $\mathfrak{s}^{\mathcal{F}}_T([M]_{\doteqdot}) \triangleq \mathscr{L}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(M) \diamond T)$

- $\mathfrak{c}_T^{\mathcal{F}} : \text{SFA}^{\sigma}_{\doteqdot} \rightarrow \wp(\Sigma^*)$ *such that* $\mathfrak{c}_T^{\mathcal{F}}([M]_{\doteqdot}) \triangleq \{s \in \Sigma^* \mid \mathfrak{T}_T(s) \nsubseteq \mathscr{L}(M)\}$

Lemma 3. *Given* $T^{\sigma/\gamma}$ *and* $[M]_{\doteqdot} \in \text{SFA}^{\sigma}_{\doteqdot}$ *then* $\forall M', M'' \in [M]_{\doteqdot}$ *we have that* $\mathfrak{s}_T^{\mathcal{F}}([M']_{\doteqdot}) = \mathfrak{s}_T^{\mathcal{F}}([M'']_{\doteqdot})$ *and* $\mathfrak{c}_T^{\mathcal{F}}([M']_{\doteqdot}) = \mathfrak{c}_T^{\mathcal{F}}([M'']_{\doteqdot})$.

Given a pair of SFAs $\langle A, B\rangle$ and an SFT T for which $\mathcal{F}$-completeness does not hold, we define the $\mathcal{B}$-complete shell of the equivalence class $[B]_{\doteqdot}$ wrt T and A as the equivalence class of SFAs that recognize the language that minimally expands the language recognized by $[B]_{\doteqdot}$ in order to gain $\mathcal{F}$-completeness, and the $\mathcal{F}$-complete core of the equivalence class $[A]_{\doteqdot}$ wrt T and B as the equivalence class of SFAs that recognize the language that minimally reduces the language recognized by $[A]_{\doteqdot}$ in order to gain $\mathcal{F}$-completeness.

Definition 12. *Consider a pair of SFAs* $\langle A^{\sigma}, B^{\gamma}\rangle$ *and an SFT* $T^{\sigma/\gamma}$ *such that the* $\mathcal{F}$*-completeness condition is not satisfied. We define the following transformers:*

- $\mathcal{F}$*-complete shell transformer* $\mathcal{ST}^{\mathcal{F}}_{T,A} : \text{SFA}^{\gamma}_{\doteqdot} \rightarrow \text{SFA}^{\gamma}_{\doteqdot}$ *such that*

$$\mathcal{ST}^{\mathcal{F}}_{T,A}([B]_{\doteqdot}) \triangleq \{M \in \text{SFA}^{\gamma} \mid \mathscr{L}(M) = \mathscr{L}(B) \cup \mathfrak{s}_T^{\mathcal{F}}(A)\} \in \text{SFA}^{\gamma}_{\doteqdot}$$

- $\mathcal{F}$*-complete core transformer* $\mathcal{CT}^{\mathcal{F}}_{T,B} : \text{SFA}^{\sigma}_{\doteqdot} \rightarrow \text{SFA}^{\sigma}_{\doteqdot}$ *such that*

$$\mathcal{CT}^{\mathcal{F}}_{T,B}([A]_{\doteqdot}) \triangleq \{M \in \text{SFA}^{\sigma} \mid \mathscr{L}(M) = \mathscr{L}(A) \smallsetminus \mathfrak{c}_T^{\mathcal{F}}(B)\} \in \text{SFA}^{\sigma}_{\doteqdot}$$

Lemma 4. *Given a pair of SFAs* $\langle A^{\sigma}, B^{\gamma}\rangle$ *and an SFT* $T^{\sigma/\gamma}$ *we have that:*

- $\forall B', B'' \in [B]_{\doteqdot}$ *we have that* $\mathcal{ST}^{\mathcal{F}}_{T,A}([B']_{\doteqdot}) = \mathcal{ST}^{\mathcal{F}}_{T,A}([B'']_{\doteqdot})$
- $\forall A', A'' \in [A]_{\doteqdot}$ *we have that* $\mathcal{CT}^{\mathcal{F}}_{T,B}([A']_{\doteqdot}) = \mathcal{CT}^{\mathcal{F}}_{T,B}([A'']_{\doteqdot})$

Also in this case we provide a characterization of an SFA in the equivalence class $\mathcal{ST}^{\mathcal{B}}_{T,B}([A]_{\doteqdot})$ and of an SFA in the equivalence class $\mathcal{CT}^{\mathcal{B}}_{T,A}([B]_{\doteqdot})$, that proves that these classes are not empty.

Proposition 7. *Consider a pair of SFAs* $\langle A^{\sigma}, B^{\gamma}\rangle$ *and an SFT* $T^{\sigma/\gamma}$ *we have that:*

- $B \oplus \text{SFA}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A) \diamond T) \in \mathcal{ST}^{\mathcal{F}}_{T,A}([B]_{\doteqdot})$
- $\text{SFA}_{\mathcal{I}}(\mathcal{T}_{\mathcal{O}}(A) \diamond T \diamond \mathcal{T}_{\mathcal{O}}(B)) \in \mathcal{CT}^{\mathcal{F}}_{T,B}([A]_{\doteqdot})$

The following result proves that the complete shell and core transformers induce $\mathcal{F}$-completeness by minimally modifying the language of the input and output SFA.

Proposition 8. *Consider a pair of SFAs* $\langle A^{\sigma}, B^{\gamma}\rangle$ *and an SFT* $T^{\sigma/\gamma}$ *such that the* $\mathcal{F}$*-completeness condition is not satisfied:*

- $\forall B' \in \mathcal{ST}^{\mathcal{F}}_{T,A}([B]_{\doteqdot})$ *we have that* $\langle A, B'\rangle$ *is* $\mathcal{F}$*-complete for* T *and* $\mathscr{L}(B')$ *minimally expands* $\mathscr{L}(B)$ *in order to gain* $\mathcal{F}$*-completeness*
- $\forall A' \in \mathcal{CT}^{\mathcal{F}}_{T,B}([A]_{\doteqdot})$ *we have that* $\langle A', B\rangle$ *is* $\mathcal{F}$*-complete for* T *and* $\mathscr{L}(A')$ *minimally reduces* $\mathscr{L}(A)$ *in order to gain* $\mathcal{F}$*-completeness*

Consider, for example the pair $\langle A, S_2\rangle$ $\mathcal{F}$-incomplete for T. In this case, in order to gain completeness we should transform S_2 in order to add to $\mathscr{L}(S_2)$ the language $\mathscr{L}_{\mathcal{O}}(\mathcal{T}_{\mathcal{O}}(A) \diamond T)$ (see Fig. 3) which recognizes also the sequences involving $x \notin \mathbb{N}$ ($x \neq <$). We fall, in this way, in the equivalence class of the SFA B (Fig. 1), which is such that $\mathscr{L}(B) \subset \mathscr{L}(S_1)$, since we minimally enrich the language.
In this case, also the core is meaningful, since it erases from $\mathscr{L}(A)$ all the sequences leading to strings not belonging to $\mathscr{L}(S_2)$, which all all those strings involving symbols not in $\mathbb{N}$, hence a representative of the resulting equivalence class is A' provided on the right of Fig. 5.

5 Code sanitisation as complete approximate transduction

Among the top ten most dangerous vulnerabilities we can find the code injection and the XSS vulnerabilities [1]. These attacks are mainly based on the evaluation (by means of a reflection operation) of a string containing code where code is not attended. Hence, following also the idea proposed in [24], we can characterize the language of the strings containing a possible attack. This language can be represented by regular expressions, CF grammars or SFA. For instance, in JavaScript, we can suppose that a string containing an attack can be any string containing `"<script"`, which means that the string contains something that will be evaluated as code and therefore executed.

Characterising these attacks is difficult when we do expect code. In this case we need to discriminate between benign and malign code. The idea is that, once we have a specification of what we don't want to execute in output and a (potentially abstract) characterisation of how this output string is previously manipulated by the program, we can derive an approximation of the input language we can or cannot accept. As a consequence we have a specification of the kind of filter/sanitizer we have to implement in order to guarantee protection against the considered attacks. We contribute to this task as follows: (1) We characterize the transducer $\mathbf{T}$, obtained by composing the different string manipulations in the program, focusing on the transformations of the strings of interest, finally evaluated in the reflection operation; (2) We consider an SFA $\mathbf{A}$ characterizing the attack language we want to avoid, i.e., the language of strings whose unexpected execution should be avoided. (3) If the language $\mathscr{L}_{\mathcal{I}}(\mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A}))$ is empty it means that the transducer $\mathbf{T}$ does not produce in output any string recognized by $\mathbf{A}$. In other words, the application is safe since no attack strings can be generated by the reflection operation. Otherwise, $\mathscr{L}_{\mathcal{I}}(\mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A}))$ precisely describes the input strings leading to elements of the undesired output language, characterized by $\mathbf{A}$.

Consider the example in Fig. 6[1], it is written in a simple pseudo JavaScript language (with the reflection operation **eval**). In this example we suppose to be on a site, e.g., `mysite.com`, where the user logs in providing a username and a password. In order to simulate a real web application, for instance written in JavaScript, we suppose that stores $\mathbb{S}$ are split in high level stores $\mathbb{S}_{\mathfrak{h}}$, representing, for instance, trusted sources such as web addresses, and low level stores $\mathbb{S}_{\mathfrak{l}}$, representing untrusted sources of data. In the following, we will use the pedix $\mathfrak{h}$ or $\mathfrak{l}$ for variables respectively in the high and

[1] The JavaScript version of this example is inspired by an example in [6]

```
if (date$_\mathfrak{l}$.dd $< 16$)
      src := display1$_\mathfrak{l}$.'vendor$_\mathfrak{l}$ :=' vendor1'';
if (date$_\mathfrak{l}$.dd $> 15$)
      src := display2$_\mathfrak{l}$.'vendor$_\mathfrak{l}$ :=' vendor2'';
params$_\mathfrak{h}$ := 'user = '.user$_\mathfrak{h}$.'and password = '.pwd$_\mathfrak{h}$
baseUrl$_\mathfrak{l}$ := params$_\mathfrak{h}$;
eval(src$_\mathfrak{l}$);
```

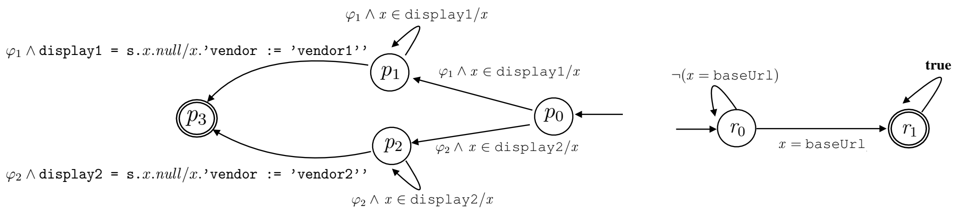
Example of attack scenario in display1$_\mathfrak{l}$:

```
display1$_\mathfrak{l}$ =' Evil$_\mathfrak{l}$ := baseUrl'$_\mathfrak{l}$;
```

Fig. 6. Code with XSS vulnerable and example of attach.

low level store, i.e., $x_\mathfrak{h}$ or $x_\mathfrak{l}$. Hence, for instance, we can simulate in our language the action of sending information to a web server by saving the information on a low level variable. Moreover, suppose that `date` is a public object variable with fields `dd`, `mm` and `yy`. Hence, username and password are saved in the variable `params`$_\mathfrak{h}$ (which is high level security). The page also loads two different third-party scripts (depending on the period of the month), which will be evaluated when the string is received from the network. The executed script is then followed by the setting of the vendor of the visualized add. The showed possible attack scenario corresponds to a bad network string `display1` sending, to an untrusted site represented by `Evil`$_\mathfrak{l}$, the parameters. This is obtained by assigning the variable containing sensitive data to the evil site. In a real application, we could use a bad network string `display1.js`, returned by the malicious or compromised site, overwriting the pages settings. Then, by clicking the login button, username and password are sent to a bad site instead of to `mysite.com` [6].

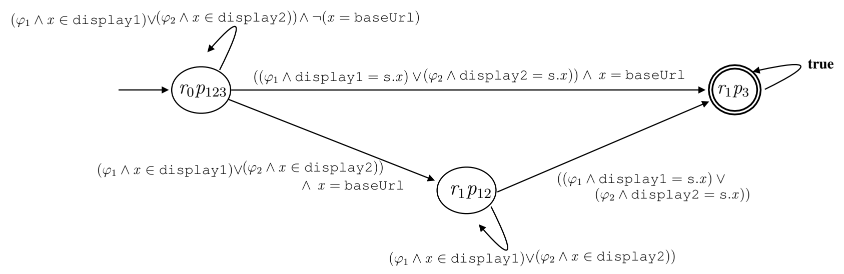
Since we are interested only in the string executed by means of the **eval** statement, we need to characterize the transducer that transforms the **eval** input string, and this is obtained by composing the transducers corresponding to the different string manipulations in the program. A simplification of this transducer is given in Fig. 7, where φ_1 is the test on the date true when the day is less than 16, while φ_2 is the other condition, i.e., day greater or equal to 16. Moreover, by the notation `s.`*x.null* we denote that `x` is the last symbol of the string `display1`. Suppose now that we aim at avoiding the



$\varphi_1 \wedge x \in$ display1$/x$
$\varphi_1 \wedge$ display1 = s.x.$null/x$.'vendor := 'vendor1''
p_1
$\varphi_1 \wedge x \in$ display1$/x$
p_3
p_0
p_2
$\varphi_2 \wedge x \in$ display2$/x$
$\varphi_2 \wedge$ display2 = s.x.$null/x$.'vendor := 'vendor2''
$\varphi_2 \wedge x \in$ display2$/x$
$\neg(x =$ baseUrl$)$
true
r_0
r_1
$x =$ baseUrl

Fig. 7. Transducer manipulating the **eval** input string and SFA approximating the above attack scenario.

attack showed in Fig. 6, in this case we could simply check whether the variable name `baseUrl` is in the string evaluated by the **eval** statement, since in this case the url may be arbitrarily modified. The SFA representing the language of strings containing `baseUrl` is given on the right of Fig. 7. At this point, in order to characterize the input language leading, through the execution of the transducer, to strings accepted by the SFA in Fig. 7, we first compute the composition $\mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A})$ and then we extract the SFA recognizing its input language, i.e., $\text{SFA}_{\mathcal{I}}(\mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A}))$, depicted in Fig. 8.



$(\varphi_1 \land x \in \texttt{display1}) \lor (\varphi_2 \land x \in \texttt{display2})) \land \neg(x = \texttt{baseUrl})$
$r_0 p_{123}$
$((\varphi_1 \land \texttt{display1} = \texttt{s}.x) \lor (\varphi_2 \land \texttt{display2} = \texttt{s}.x)) \land\ x = \texttt{baseUrl}$
$r_1 p_3$
true
$(\varphi_1 \land x \in \texttt{display1}) \lor (\varphi_2 \land x \in \texttt{display2}))$
$\land\ x = \texttt{baseUrl}$
$r_1 p_{12}$
$((\varphi_1 \land \texttt{display1} = \texttt{s}.x) \lor$
$(\varphi_2 \land \texttt{display2} = \texttt{s}.x))$
$(\varphi_1 \land x \in \texttt{display1}) \lor (\varphi_2 \land x \in \texttt{display2}))$

Fig. 8. Input SFA leading to the given attack.

Note that this SFA can be used in order to specify a filter/sanitizer making the program safe, namely avoiding dangerous inputs. As far as our example is concerned, the following code is a possible filter avoiding the strings recongized by the SFA in Fig. 8:

```
1 : $\texttt{i} := 0; \texttt{temp}_{\mathfrak{l}} := \texttt{display1}_{\mathfrak{l}};$
2 : **while** $\texttt{i} < \textbf{length}(\texttt{temp}_{\mathfrak{l}})$
3 :       $\{$**if** $\texttt{temp}_{\mathfrak{l}}[\texttt{i}] = \texttt{baseUrl}$ **then**
4 :             $\{\texttt{display1}_{\mathfrak{l}} := \mathit{nil}; \texttt{i} = \textbf{length}(\texttt{temp}); \}$
5 :       $\texttt{i} := \texttt{i} + 1; \}$
```

This filter should be executed for both $\texttt{display1}_{\mathfrak{l}}$ and $\texttt{display2}_{\mathfrak{l}}$, before the rest of the program code. It is interesting to observe that the language of input strings to be avoid corresponds precisely to a $\mathcal{B}$-complete shell computation. In particular, consider an SFA recognizing the empty language $\mathbf{E}$, then if we have $\mathcal{T}_{\mathcal{O}}(\mathbf{E}) \diamond \mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A}) \equiv \mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A})$ it means that both the output languages are empty, which means, as said before, that the application is safe. So the property of being safe of a program (expressed as a transduction $\mathbf{T}$) wrt an attack (expressed as an SFA $\mathbf{A}$) can be expressed as a $\mathcal{B}$-completeness property. When we do not have $\mathcal{B}$-completeness, namely when $\mathcal{T}_{\mathcal{O}}(\mathbf{E}) \diamond \mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A}) \not\equiv \mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A})$, we can compute the $\mathcal{B}$-complete shell of $\mathbf{E}$ wrt $\mathbf{T}$ and $\mathbf{A}$ and obtain in this way a characterization of the smallest set of input strings that should be avoided in order to be sure that attack $\mathbf{A}$ cannot happen. Indeed, the $\mathcal{B}$-complete shell of $\mathbf{E}$ wrt $\mathbf{T}$ and $\mathbf{A}$ precisely corresponds to the computation of an SFA that recognizes the language $\mathscr{L}_{\mathcal{I}}(\mathbf{T} \diamond \mathcal{T}_{\mathcal{O}}(\mathbf{A}))$, as for example the SFA depicted in Fig. 8. It could also be interesting to compute the $\mathcal{B}$-complete core of $\mathbf{A}$ wrt $\mathbf{T}$ and $\mathbf{E}$, since this would provide a characterization of the attacks that can actually occur, which may in general be a subset of $\mathscr{L}(\mathbf{A})$.

6 Conclusion and Related Works

The main contribution of this paper is in introducing the notion of approximate transduction induced by abstract symbolic automata. We formalized the notion of $\mathcal{B}$ and $\mathcal{F}$-completeness of the approximate transduction on the abstract domain of SFAs. In particular, we gave necessary and sufficient conditions that guarantee the completeness of the approximate computation of transductions on SFAs, thus avoiding the increase of the loss of precision (viz., producing larger languages) when applied to approximate SFAs. Moreover we provided a formal characterization of the minimal modifications of the languages recognized by the SFAs that can guarantee $\mathcal{B}$ and $\mathcal{F}$-completeness (namely the $\mathcal{B}/\mathcal{F}$-complete shell and the $\mathcal{B}/\mathcal{F}$-complete core). An example of the computation of the $\mathcal{B}$-complete shell and of the $\mathcal{B}$-complete core has been given in the context of XSS attack prevention. Indeed, in this scenario the $\mathcal{B}$-complete shell of a SFA specification with respect to the corresponding approximate transduction can be used for the synthesis of a sanitiser from the symbolic specification of a given XSS attack. This provides an interesting bridge between the synthesis of a sanitiser and the completeness of the associated abstract interpretation in the domain of SFA.

The analysis of strings is nowadays a relatively common practice in program analysis due to the widespread of dynamic scripting languages that require advanced analysis tools for predicting bugs and therefore to overcome their unpredictable intrinsic dynamic nature. Examples of analyses for string manipulation are in [14,5,30,25,22,21]. None of these analyses formalise the loss of precision in approximating operations on strings by transducers in terms of the loss of completeness of the corresponding language transformation. The use of symbolic (grammar-based) objects in abstract domains is not new (see [9,17,29]). With respect to these works we exploit symbolic automata and transducers as abstract domain with a specific analysis of the completeness conditions for predicate transformers specified as symbolic transducers.

The use of transducers for script sanitisation is known in the literature (see for instance [19] and [30]). With respect to [19] we prove that by exploiting completeness it is possible to extract minimal sanitisers by a simple backward transduction. This better fits the assumption that sanitisers have to be small amount of code (possibly the smallest possible code for the task). Our application on XSS sanitisation is largely inspired from [30]. With respect to [30] the specification of the analysis in terms of abstract interpretation makes it suitable for being combined with other analyses, with a better potential in terms of tuning in accuracy and costs.

As future work, regular model checking [2] and the static analysis of dynamically generated code represent natural application fields for abstract symbolic automata and transducers. In the first case, sets of states and the transfer functions are respectively represented as automata and transducers. Our work provides completeness conditions for approximate (abstract) regular model checking. In the second case, the code dynamically generated in dynamic languages such as JavaScript and PHP can be approximated in the abstract domain of symbolic automata, and then further synthesised to statically extract the dynamic evolution of code as if it is a generic mutable data structure.

Acknowledgments: This work is partly supported by the MIUR FIRB project FACE (Formal Avenue for Chasing malwarE) RBFR13AJFT.

References

1. OWASP Top Ten Project 2013. https://www.owasp.org.
2. P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In P. Gardner and N. Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.
3. J. Berstel. *Transductions and Context-Free Languages*. Teubner-Verlag, 2009.
4. N. Bjørner and M. Veanes. Symbolic transducers. Technical Report MSR-TR-2011-3, Microsoft Research, 2011.
5. A. S. Christensen, A. Møller, and M. I. Schwartzbach. Precise analysis of string expressions. In R. Cousot, editor, *Static Analysis, 10th International Symposium, SAS 2003, San Diego, CA, USA, June 11-13, 2003, Proceedings*, volume 2694 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2003.
6. R. Chugh, J. A. Meister, R. Jhala, and S. Lerner. Staged information flow for JavaScript. In M. Hind and A. Diwan, editors, *PLDI*, pages 50–62. ACM, 2009.
7. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages* (*POPL '77*), pages 238–252. ACM Press, 1977.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages* (*POPL '79*), pages 269–282. ACM Press, 1979.
9. P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proceedings of the Seventh ACM Conference on Functional Programming Languages and Computer Architecture*, pages 170–181. ACM Press, New York, NY, 25–28 June 1995.
10. M. Dalla Preda, R. Giacobazzi, A. Lakhotia, and I. Mastroeni. Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables. In S. K. Rajamani and D. Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 329–341. ACM, 2015.
11. L. D'Antoni and M. Veanes. Equivalence of extended symbolic finite transducers. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 624–639. Springer, 2013.
12. L. D'Antoni and M. Veanes. Minimization of symbolic automata. In S. Jagannathan and P. Sewell, editors, *POPL*, pages 541–554. ACM, 2014.
13. L. D'Antoni and M. Veanes. Extended symbolic finite automata and transducers. *Formal Methods in System Design*, 47(1):93–119, 2015.
14. K. Doh, H. Kim, and D. A. Schmidt. Abstract parsing: Static analysis of dynamically generated string output using lr-parsing technology. In J. Palsberg and Z. Su, editors, *Static Analysis, 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings*, volume 5673 of *Lecture Notes in Computer Science*, pages 256–272. Springer, 2009.
15. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model-checking. In P. Cousot, editor, *Proc. of The 8th Internat. Static Analysis Symp. (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science*, pages 356–373. Springer-Verlag, 2001.
16. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretation complete. *Journal of the ACM*, 47(2):361–416, March 2000.

17. N. Heintze and J. Jaffar. Set constraints and set-based analysis. In A. Borning, editor, *Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2-4, 1994, Proceedings*, volume 874 of *Lecture Notes in Computer Science*, pages 281–298. Springer, 1994.
18. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Bek: Modeling imperative string operations with symbolic transducers. Technical Report MSR-TR-2010-154, November 2010.
19. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Fast and precise sanitizer analysis with BEK. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.
20. P. Hooimeijer and M. Veanes. An evaluation of automata algorithms for string analysis. In *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'11, pages 248–262, Berlin, Heidelberg, 2011. Springer-Verlag.
21. H. Kim, K. Doh, and D. A. Schmidt. Static validation of dynamically generated HTML documents based on abstract parsing and semantic processing. In F. Logozzo and M. Fähndrich, editors, *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 194–214. Springer, 2013.
22. Y. Minamide. Static approximation of dynamically generated web pages. In A. Ellis and T. Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 432–441. ACM, 2005.
23. A. Podelski. Automata as proofs. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 13–14. Springer, 2013.
24. D. Ray and J. Ligatti. Defining code-injection attacks. In J. Field and M. Hicks, editors, *POPL*, pages 179–190. ACM, 2012.
25. P. Thiemann. Grammar-based analysis of string expressions. In J. G. Morrisett and M. Fähndrich, editors, *Proceedings of TLDI'05: 2005 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, Long Beach, CA, USA, January 10, 2005*, pages 59–70. ACM, 2005.
26. M. Veanes. Symbolic string transformations with regular lookahead and rollback. In A. Voronkov and I. Virbitskaite, editors, *Perspectives of System Informatics - 9th International Ershov Informatics Conference, PSI 2014, St. Petersburg, Russia, June 24-27, 2014. Revised Selected Papers*, volume 8974 of *Lecture Notes in Computer Science*, pages 335–350. Springer, 2014.
27. M. Veanes, P. d. Halleux, and N. Tillmann. Rex: Symbolic regular expression explorer. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ICST '10, pages 498–507, Washington, DC, USA, 2010. IEEE Computer Society.
28. M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner. Symbolic finite state transducers: algorithms and applications. In J. Field and M. Hicks, editors, *POPL*, pages 137–150. ACM, 2012.
29. A. Venet. Automatic analysis of pointer aliasing for untyped programs. *Sci. Comput. Program.*, 35(2):223–248, 1999.
30. F. Yu, M. Alkhalaf, and T. Bultan. Patching vulnerabilities with sanitization synthesis. In R. N. Taylor, H. C. Gall, and N. Medvidovic, editors, *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 251–260. ACM, 2011.