Symbolic Tree Transducers

Margus Veanes and Nikolaj Bjørner

Microsoft Research, Redmond, WA, USA
{margus,nbjorner}@microsoft.com

Abstract. Symbolic transducers are useful in the context of web security as they form the foundation for sanitization of potentially malicious data. We define Symbolic Tree Transducers as a generalization of Regular Transducers as finite state input-output tree automata with logical constraints over a parametric background theory. We examine key closure properties of Symbolic Tree Transducers and we develop a composition algorithm and an equivalence decision procedure for linear single-valued transducers.

1 Introduction

Several applications, ranging from web-sanitizers, XML transformations to generic functional programs, rely on finite state machines that transform strings or trees into strings or trees. Such state machines can conveniently be captured by tree transducers. This work develops symbolic tree transducers (STTs) that are defined modulo a background theory. STTs are easily seen more expressive than tree transducers defined over finite alphabets, yet our main results establish that composition of STTs and equivalence checking for linear single valued STTs is computable, modulo the background theory. Symbolic transducers are also practically useful for exploiting efficient symbolic solvers when performing basic automata-theoretic transformations. Prior work [35, 19] on symbolic string recognizers and transducers takes advantage of this observation. We here investigate the case of the more expressive class of *tree* transducers. The complexity of decision problems are highly sensitive to the expressive power given to tree transducers and we here identify a class of top-down transducers that admit decidable equivalence checking modulo decidability of the symbolic background component.

2 Preliminaries

We use basic notions from classical automata theory [20], classical logic, and model theory [18]. Our notions regarding tree transducers are consistent with [15]. For finite state (string) transducers a brief introduction is given in [36].

2.1 Background Structure

We work modulo a *background* structure $\mathcal{U}$ over a language 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 *ground*.

We use *parameterized algebraic* sorts to represent labeled trees. An algebraic sort is associated with a finite collection of *constructors* and *accessors*. In particular, we use the sort $\text{T}^k\langle\sigma\rangle$ to denote the set of σ-labeled k-ary trees, for $k \geq 1$, that is associated with the constructors

$$f : \sigma \times \text{T}^k\langle\sigma\rangle \times \cdots \times \text{T}^k\langle\sigma\rangle \to \text{T}^k\langle\sigma\rangle, \quad \epsilon : \text{T}^k\langle\sigma\rangle,$$

for constructing a nonempty tree and an empty tree respectively. The accessors of $\text{T}^k\langle\sigma\rangle$ are the subtree accessors $\boldsymbol{1}, \ldots, \boldsymbol{k} : \text{T}^k\langle\sigma\rangle \to \text{T}^k\langle\sigma\rangle$ and the label accessor $\boldsymbol{0} : \text{T}^k\langle\sigma\rangle \to \sigma$. For all $t_0 : \sigma$, $t_i : \text{T}^k\langle\sigma\rangle$, $1 \leq i \leq k$, $\boldsymbol{i}(f(t_0, t_1, \ldots, t_k)) = t_i$. Note that constructors have term interpretation, thus, for all $t, u : \text{T}\langle\sigma\rangle$,

$$t = u \quad \Leftrightarrow \quad t = u = \epsilon \vee t \neq \epsilon \wedge u \neq \epsilon \wedge \bigwedge_{i=0}^{k} \boldsymbol{i}(t) = \boldsymbol{i}(u).$$

A *position* is a sequence of accessors. We write $t|_\pi$ for the subterm of t at position π, e.g., $f(a, f(b, t, u), v)|_{\boldsymbol{12}} = u$. We write $t[\pi \leftarrow v]$ for replacing the occurrence of the subterm at position π in t by v, e.g.,

$$f(a, \epsilon, f(b, \epsilon, \epsilon))[\boldsymbol{22} \leftarrow v] = f(a, \epsilon, f(b, \epsilon, v))$$

We often omit k from $\text{T}^k\langle\sigma\rangle$. We write $\text{L}\langle\sigma\rangle$ for the *σ-list* sort $\text{T}^1\langle\sigma\rangle$. We use the notation $[e_1, e_2, \ldots, e_n|t]$ for the list $f(e_1, f(e_2, \ldots f(e_n, t)))$ and we write $[e_1, e_2, \ldots, e_n]$ when $t = \epsilon$.

The *trace ending in tree position π of a tree t* or a *π-trace of t* is the list of labels from the root of t up to π, e.g., if $t = f(a, \epsilon, f(b, f(c, \epsilon, \epsilon), \epsilon))$ then the $\boldsymbol{22}$-trace of t is $[a, b]$, the $\boldsymbol{211}$-trace of t is $[a, b, c]$, the ε-trace of t is ϵ.

2.2 Top-down tree transducers

A top-down tree transducer describes a transformation function from trees in a given input domain into trees in a given output domain. Several equivalent formal definitions are possible. Typically, the rules specify how an input tree is transformed through a recursive descent over the structure of the input domain. Here, trees are terms of sort $\text{T}\langle\sigma\rangle$ of a given label sort σ. The following definition is tailored to a generalization for symbolic tree transducers introduced below. We write $t[x_1, \ldots, x_k]$ to indicate that all free variables in t are among $x_1, \ldots, x_k$ distinct variables, and we write $t[t_1, \ldots, t_k]$ for substituting x_i in t by t_i for $1 \leq i \leq k$. The term t is *linear* if each x_i occurs at most once in t.

Definition 1. A *(top-down) tree transducer from* $\mathrm{T}\langle\iota\rangle$ *to* $\mathrm{T}\langle o\rangle$ is a tuple (Q, q^0, R) where Q is a finite set of unary constructors $q : \mathrm{T}\langle\iota\rangle \to \mathrm{T}\langle o\rangle$, called *states*, $q^0 \in Q$ is the *initial state*, R is a set of *rules* of the form

$$q(\epsilon) \to e, \quad q(f(a, y_1, \ldots, y_k)) \to u[q_1(y_1), \ldots, q_k(y_k)]$$

where $e : \mathrm{T}\langle o\rangle$ and $a : \iota$ are ground terms, $y_i : \mathrm{T}\langle\iota\rangle$ are distinct variables, $u[x_1, \ldots, x_k] : \mathrm{T}\langle o\rangle$ is a term that does not contain states, and $q, q_1, \ldots, q_k \in Q$. Given a rule $l \to r$, l is the *left-hand side* and r the *right-hand side* of the rule.

Tree transducers do not have explicit *final* states, since using a rule $q(\epsilon) \to \epsilon$ is effectively equivalent to declaring the state q as a final state. We write $A^{\mathrm{T}\langle\iota\rangle/\mathrm{T}\langle o\rangle}$ for a tree transducer from $\mathrm{T}\langle\iota\rangle$ to $\mathrm{T}\langle o\rangle$. A tree transducer *with epsilon moves* may additionally have rules of the form $p(x) \to q(x)$ where p and q are states and x is a variable. A rule is *linear* if its right-hand side is linear and A is *linear* if all of its rules are linear. A is *deterministic* if it has no epsilon moves and no two rules with overlapping left-hand sides.

Although Definition 1 is specialized for labeled trees with fixed arity and a single nonempty constructor, this does not cause any loss of generality and simplifies the presentation technically. For example, a term $f(n, t_1, t_2)$ of sort $\mathrm{T}^2\langle\mathrm{INT}\rangle$, where n is a fixed integer value, can be seen as a representation for $f_n(t_1, t_2)$ for some binary constructor f_n. A key distinction from a standard definition of tree transducers is that the universes of input labels ($\mathcal{U}^\iota$) and output labels ($\mathcal{U}^o$) may be infinite.

The definition allows *nondeterminism* and does not require the rules to be *total*. While for certain purposes it is sufficient that tree transducers are deterministic and total by definition [15], both nondeterminism and partiality of the rules in Definition 1 play an important role in the context of *symbolic* tree transducers, as discussed below.

We say that a ground term or a tree t is *basic* (with respect to a tree transducer A) if it does not contain any states from A. The *transformation* or *transduction* induced by a tree transducer $A^{\mathrm{T}\langle\iota\rangle/\mathrm{T}\langle o\rangle}$ is a function $\mathbf{T}_A$ from basic trees of sort $\mathrm{T}\langle\iota\rangle$ to sets of basic trees of sort $\mathrm{T}\langle o\rangle$. The definition of $\mathbf{T}_A$ is a direct generalization of the standard definition: $\mathbf{T}_A(t)$ is the set of all basic trees modulo $\mathcal{U}$ in the closure of $\{q^0(t)\}$ under the rules of A.

Definition 2. A tree transducer A is *single-valued* if $|\mathbf{T}_A(t)| \leq 1$ for all t.

3 Symbolic tree transducers

In this section we introduce an extension of tree transducers through a symbolic encoding of labels by predicates. The main advantage of the extension is succinctness and modularity with respect to the background theory of labels.

Definition 3. A *symbolic tree transducer (STT) from* $\mathrm{T}\langle\iota\rangle$ *to* $\mathrm{T}\langle o\rangle$ is a tuple (Q, q^0, R) with Q as a finite set of states, $q^0 \in Q$ as the initial state, and R as a

finite set of *(guarded) rules*

$$q(\epsilon) \to e, \quad q(f(x, y_1, \ldots, y_k)) \xrightarrow{\varphi[x]} u[x, q_1(y_1), \ldots, q_k(y_k)]$$

where e is a basic ground term, x is a variable, y_i, for $1 \leq i \leq k$, are distinct variables, $u[x, x_1, \ldots, x_k]$ is a basic term, $q, q_1, \ldots, q_k \in Q$, and $\varphi[x]$ is a predicate called the *guard* of the rule.

A guarded rule $\rho = q(f(x, y_1, \ldots, y_k)) \xrightarrow{\varphi[x]} u[x, q_1(y_1), \ldots, q_k(y_k)]$ denotes the set of rules

$$[\![\rho]\!] \stackrel{\text{def}}{=} \{q(f(a, y_1, \ldots, y_k)) \to u[a, q_1(y_1), \ldots, q_k(y_k)] \mid a \in \mathcal{U}^\iota,\ \varphi[a] \text{ holds}\}$$

Thus $[\![\rho]\!]$ may be infinite when $\mathcal{U}^\iota$ is infinite. Given an STT $A = (Q, q_0, R)$ we write $[\![A]\!]$ for the tree transducer $(Q, q_0, \cup\{[\![\rho]\!] \mid \rho \in R\})$ and $\mathbf{T}_A$ for $\mathbf{T}_{[\![A]\!]}$. An STT A is *linear* (resp. *single-valued*, *deterministic*) if $[\![A]\!]$ is linear (resp. single-valued, deterministic). Note that the right-hand sides of rules of a linear STT are allowed to contain multiple occurrences of the input label variable x. In particular, all STTs over lists are linear.

In the following examples, all STTs are single-valued and linear. The first example illustrates some simple STTs over $\text{T}^2\langle\text{INT}\rangle$. The point is to illustrate how global STT properties depend on the theory of labels.

Example 1. Let the input and the output domains be $\text{T}^2\langle\text{INT}\rangle$. *Swap* is an STT that swaps the left and the right subtrees if the label is non-zero. *Neg* is an STT that multiplies all labels by -1, *Double* multiplies labels by 2. *Cut* is an STT that cuts the left subtree y_1 of $f(x, y_1, y_2)$ when $x > 0$ and cuts the right subtree y_2 when $x < 0$.

$$\begin{aligned}
\mathit{Swap} = (\{q\}, q, \{q(\epsilon) \to \epsilon, q(f(x, y_1, y_2)) &\xrightarrow{x \neq 0} f(x, q(y_2), q(y_1)),\\
q(f(x, y_1, y_2)) &\xrightarrow{x=0} f(x, q(y_1), q(y_2))\})\\
\mathit{Neg} = (\{q\}, q, \{q(\epsilon) \to \epsilon, q(f(x, y_1, y_2)) &\xrightarrow{true} f(-x, q(y_1), q(y_2))\})\\
\mathit{Double} = (\{q\}, q, \{q(\epsilon) \to \epsilon, q(f(x, y_1, y_2)) &\xrightarrow{true} f(2x, q(y_1), q(y_2))\})\\
\mathit{Cut} = (\{q\}, q, \{q(\epsilon) \to \epsilon, q(f(x, y_1, y_2)) &\xrightarrow{x>0} f(x, \epsilon, q(y_2)),\\
q(f(x, y_1, y_2)) &\xrightarrow{x<0} f(x, q(y_1), \epsilon)\\
q(f(x, y_1, y_2)) &\xrightarrow{x=0} f(x, q(y_1), q(y_2))\})
\end{aligned}$$

Note that global properties such as commutativity and idempotence of the STTs clearly depend on the theory of labels, e.g., that multiplication by a positive number preserves polarity, implying in this case for example that *Swap* and *Neg* commute, *Cut* and *Double* commute, and *Cut* is idempotent. Note also that none of the examples can be expressed as a finite tree transducer. Our results about composition and equivalence checking for STTs, that are discussed in the below sections, allow to establish equivalences, such as *Cut* is equivalent to *Swap* followed by *Neg*, *Cut*, then finally *Swap*. The equivalence is modulo the theory of arithmetic that establishes logical equivalences, such as $-x < 0 \equiv x > 0$. ⊠

The following example illustrates a nontrivial use of the label theory. The STT *Encode* in the example represents the string sanitizer `AntiXSS.EncodeHtml` from version 2.0 of the Microsoft AntiXSS library. The sanitizer transforms an input string into an Html friendly format. For each character x in the input string, either x is kept verbatim or encoded through numeric Html escaping. The example can be extended to be part of a tree transducer over abstract syntax trees of Html where certain parts of the tree (corresponding to strings) are encoded using *Encode*.

Example 2. The example illustrates a single-state INT-list STT $Encode^{\mathrm{L}\langle\mathrm{INT}\rangle/\mathrm{L}\langle\mathrm{INT}\rangle}$ that transforms an input list of characters represented by positive integers, into an encoded, possibly longer, list of characters. We assume that '`...`' below represents the integer encoding of the given fixed (ASCII) character, e.g. '`a`' = 97 and '`z`' = 122. Let $\varphi[x]$ be the following linear arithmetic formula:

$$\begin{array}{l}(\text{`}\texttt{a}\text{'} \leq x \leq \text{`}\texttt{z}\text{'}) \vee (\text{`}\texttt{A}\text{'} \leq x \leq \text{`}\texttt{Z}\text{'}) \vee \\ (\text{`}\texttt{0}\text{'} \leq x \leq \text{`}\texttt{9}\text{'}) \ \vee x = \text{`}\ \text{'} \vee x = \text{`}\texttt{.}\text{'} \vee x = \text{`}\texttt{,}\text{'} \vee x = \text{`}\texttt{-}\text{'} \vee x = \text{`}\texttt{\_}\text{'}\end{array}$$

Encode contains the following seven rules ($Q_{Encode} = \{q\}$):

$$\begin{array}{lll} q(\epsilon) & \to & \epsilon \\ q([x|y]) & \xrightarrow{\varphi[x]} & [x|q(y)] \\ q([x|y]) & \xrightarrow{\neg\varphi[x] \wedge 0 \leq x < 10} & [\text{`}\texttt{\&}\text{'}, \text{`}\texttt{\#}\text{'}, \mathbf{d}_0(x), \text{`}\texttt{;}\text{'}|q(y)] \\ q([x|y]) & \xrightarrow{\neg\varphi[x] \wedge 10^n \leq x < 10^{n+1}} & [\text{`}\texttt{\&}\text{'}, \text{`}\texttt{\#}\text{'}, \mathbf{d}_n(x), \ldots, \mathbf{d}_0(x), \text{`}\texttt{;}\text{'}|q(y)] \quad (\text{for } 1 \leq n \leq 4) \end{array}$$

where

$$\mathbf{d}_i(x) \stackrel{\text{def}}{=} ((x \div 10^i)\%10) + 48$$

is a term in linear arithmetic representing the (ASCII) character value of the i'th decimal position of x, where $\div$ is integer division, $+$ is integer addition, and $\%$ computes the integer remainder after dividing its first operand by its second. By using that '`&`' = 38 (i.e., $\mathbf{d}_1(\text{`}\texttt{\&}\text{'}) = \text{`}\texttt{3}\text{'}$ and $\mathbf{d}_0(\text{`}\texttt{\&}\text{'}) = \text{`}\texttt{8}\text{'}$) and that $\varphi[\text{`}\texttt{\&}\text{'}]$ does not hold, it follows for example that

$$\mathbf{T}_{Encode}([\text{`}\texttt{\&}\text{'}, \text{`}\texttt{a}\text{'}]) = \{[\text{`}\texttt{\&}\text{'}, \text{`}\texttt{\#}\text{'}, \text{`}\texttt{3}\text{'}, \text{`}\texttt{8}\text{'}, \text{`}\texttt{;}\text{'}, \text{`}\texttt{a}\text{'}]\}.$$

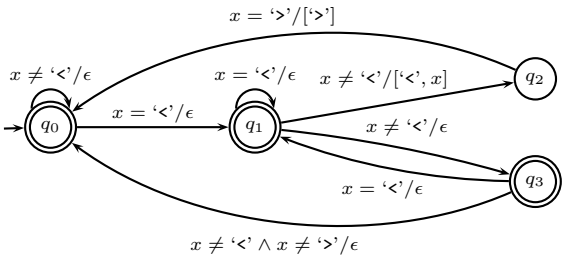
Note that *Encode* is deterministic because all the guards are mutually exclusive and therefore $[\![Encode]\!]$ contains no two rules whose left-hand sides are equal but whose right-hand sides are different. From determinism follows also that *Encode* is single-valued. ⊠

The following example illustrates another class of common single-valued list-transductions over an *infinite* label domain that are captured by a nondeterministic STT but not by any deterministic STT. While it is well-known that nondeterministic tree transducers are more expressive than deterministic tree transducers, the following example illustrates a case where a deterministic tree transducer would exist if the label domain was *finite*.

Example 3. The example illustrates an INT-list STT *Extract* that extracts from a given input list all subsequences of elements of the form [‘<’, x, ‘>’], where $x \neq$ ‘<’. For example

$$\mathbf{T}_{Extract}([\text{‘<’}, \text{‘<’}, \text{‘a’}, \text{‘>’}, \text{‘<’}, \text{‘<’}, \text{‘>’}, \text{‘<’}, \text{‘b’}, \text{‘>’}]) = [\text{‘<’}, \text{‘a’}, \text{‘>’}, \text{‘<’}, \text{‘b’}, \text{‘>’}]$$

Extract has states $\{q_0, q_1, q_2, q_3\}$ where q_0 is the initial state. *Extract* can be visualized as follows, where a rule $q(\epsilon) \to \epsilon$ is depicted by marking q as a final state, and a rule $q([x|y]) \xrightarrow{\varphi[x]} [t_1, \ldots, t_n|p(y)]$, for $n \geq 0$, is depicted as a transition from q to p having label $\varphi[x]/[t_1, \ldots, t_n]$:



A deterministic version would need a state to remember each element $x \neq$ ‘<’ from q_1 in order to later decide whether to output or to delete the elements, which depends on whether x is followed by ‘>’ or not. ⊠

4 Composition and equivalence of STTs

In this section we investigate feasibility of *composition* and *equivalence* of STTs. First, we prove that STTs are closed under composition and we provide a practical algorithm for composing STTs. The composition algorithm preserves linearity. Second, we show that equivalence of linear single-valued STTs is decidable modulo a decidable theory over labels and we provide a practical algorithm for this case. The immediate applications of these two algorithms are decision procedures for *commutativity* and *idempotence* of linear single-valued STTs.

4.1 Composition of STTs

The composition of two transductions $\mathbf{T}_1$ and $\mathbf{T}_2$ is the transduction

$$\mathbf{T}_1 \circ \mathbf{T}_2(t) \stackrel{\text{def}}{=} \bigcup_{u \in \mathbf{T}_1(t)} \mathbf{T}_2(u)$$

Notice that $\circ$ applies first $\mathbf{T}_1$, then $\mathbf{T}_2$, contrary to how $\circ$ is used for standard function composition. (The definition follows the convention used in [15].)

Note that if $\mathbf{T}_1$ and $\mathbf{T}_2$ are single-valued then so is $\mathbf{T}_1 \circ \mathbf{T}_2$. Given two STTs $A^{\iota/\sigma}$ and $B^{\sigma/o}$ we provide an algorithm for constructing an STT $A \circ B^{\iota/o}$ such that $\mathbf{T}_{A \circ B} = \mathbf{T}_A \circ \mathbf{T}_B$. The algorithm is a symbolic generalization of the classical composition algorithm for (top-down) tree transducers (cf. [15, Theorem 3.39]).

In the following let A and B be fixed STTs. The description of the algorithm assumes absence of epsilon moves.[1] We assume, for ease of presentation, that the input and the output trees have the same sort.

As a convention, rules without an explicit guard have an implicit guard that is *true*. Given a set of guarded rewrite rules R and a pair (φ, t) where φ is a formula and t a term, an *R-derivation step* is of the form:

$$(\varphi, t) \Rightarrow_R (\varphi \wedge \psi[t_0], t[\pi \leftarrow r[t_0, \bar{s}]]) \quad \text{if} \quad \begin{cases} l[x, \bar{y}] \xrightarrow{\psi[x]}_R r[x, \bar{y}] \\ t|_\pi = l[t_0, \bar{s}] \end{cases}$$

That is, in the context of condition φ, a subterm of t at position π can be rewritten using a rule from R while accumulating the side-condition for the rule. We write $(\varphi, t)\downarrow_R$ for the set of all (φ', t') such that $(\varphi, t) \Rightarrow_R^* (\varphi', t')$ and there exists no R-derivation step from t'. We use a pairing function $\langle p, q\rangle$ for $p \in Q_A, q \in Q_B$ to denote states in the composed transducer. The states come from a composition $q(p(y))$ and to give us access to the pair we augment R_B to

$$R'_B = R_B \cup \{q(p(y)) \to \langle p, q\rangle(y) \mid p \in Q_A, q \in Q_B\}$$

We can now define the composition of two transducers by $(Q, \langle q_A^0, q_B^0\rangle, R)$, where R and Q are given by a least fixed point with respect to the following conditions:

1. $\langle q_A^0, q_B^0\rangle \in Q$
2. If $\langle p, q\rangle \in Q, p(v) \xrightarrow{\varphi} u \in R_A, (\psi, t) \in (\varphi, q(u))\downarrow_{R'_B}$ then $\langle p, q\rangle(v) \xrightarrow{\psi} t \in R$
3. If $\langle p, q\rangle(v) \xrightarrow{\varphi} t[\langle p', q'\rangle(y)] \in R$ then $\langle p', q'\rangle \in Q$

The least fixed-point can be computed using a DFS traversal over the states reachable from $\langle q_A^0, q_B^0\rangle$. The algorithm for computing $(\varphi, q(u))\downarrow_{R'_B}$ can be implemented using backtracking search. A practically important optimization of the algorithm, is *satisfiability checking* of the induced guard formulas. If a formula φ' in a derivation $(\varphi, t) \Rightarrow_R^* (\varphi', t')$ is unsatisfiable, the continued search from that point on is aborted.

Example 4. Consider the self-composition of *Encode* from Example 2. For the sake of clarity let A = *Encode* but rename q_A^0 to p. Let B = *Encode*. Thus $Q_A \times Q_B = \{\langle p, q\rangle\}$.

1. Case $p(\epsilon) \to_A \epsilon$. Then $(\mathit{true}, q(\epsilon))\downarrow_R = \{(\mathit{true}, \epsilon)\}$, so $\langle p, q\rangle(\epsilon) \to \epsilon$.
2. Case $p([x|y]) \xrightarrow{\varphi[x]}_A [x|p(y)]$. We get that (formulas are simplified):

   $$(\varphi[x], q([x|p(y)])) \Rightarrow_{R'_B} (\varphi[x], [x|q(p(y))]) \Rightarrow_{R'_B} (\varphi[x], [x|\langle p, q\rangle(y)]) \not\Rightarrow_{R'_B}$$

   while any other derivation causes a conflict, e.g.

   $$(\varphi[x], q([x|p(y)])) \Rightarrow_{R'_B} (\varphi[x] \wedge \neg\varphi[x] \wedge \ldots, [\text{`\&'}, \ldots])\star$$

[1] Epsilon moves can be handled similarly, first *epsilon-loops*, that are circular paths of epsilon moves $p(y) \to \cdots \to p(y)$, are eliminated in order to avoid nonterminating derivations.

It follows that $\langle p, q\rangle([x|y]) \xrightarrow{\varphi[x]} [x|\langle p, q\rangle(y)]$.

3. Case $p([x|y]) \xrightarrow{\neg\varphi[x]\wedge 0\leq x<10}_A [\text{`\&'}, \text{`\#'}, \mathbf{d}_0(x), \text{`;'}|p(y)]$. Then

$$(\varphi[x] \wedge 0 \leq x < 10, q([\text{`\&'}, \text{`\#'}, \mathbf{d}_0(x), \text{`;'}|p(y)])) \Rightarrow^*_{R'_B} \\ (\varphi[x] \wedge 0 \leq x < 10 \wedge \varphi[\mathbf{d}_0(x)], \\ [\text{`\&'}, \text{`\#'}, \mathbf{d}_1(\text{`\&'}), \mathbf{d}_0(\text{`\&'}), \text{`;'}, \text{`\&'}, \text{`\#'}, \mathbf{d}_1(\text{`\#'}), \mathbf{d}_0(\text{`\#'}), \text{`;'}, \\ \mathbf{d}_0(x), \text{`\&'}, \text{`\#'}, \mathbf{d}_1(\text{`;'}), \mathbf{d}_0(\text{`;'}), \text{`;'}|\langle p, q\rangle(y)]) \not\Rightarrow_{R'_B}$$

The remaining cases are similar. Note that, for all x and i, $\varphi[\mathbf{d}_i(x)]$ holds because `0'} \leq \mathbf{d}_i(x) \leq \text{`9', while for $x \in \{\text{`\&'}, \text{`\#'}, \text{`;'}\}$, $\neg\varphi[x] \wedge 10 \leq x \leq 100$ holds, and thus double-encoding occurs for these characters in $A \circ B$.

The importance of early pruning of the search space using satisfiability checks is obvious in this example. Brute force exploration would cause a combinatorial explosion of the different paths, while most of them are infeasible. ⊠

The following result characterizes compositionality of STTs.

Theorem 1 (Composition). *STTs are effectively closed under composition. Moreover, linear STTs are effectively closed under composition.*

Proof. The first statement can be shown along the lines of the proof of compositionality of TOP [15, Theorem 3.39]. While the second statement can be shown similarly to compositionality of l-TOP [15, Corollary 3.41], a simpler argument, using the definition of R'_B, shows that linearity is preserved by $\Rightarrow_{R'_B}$.

Suppose A and B are linear and consider the definition of $(\varphi, t)\downarrow_{R'_B}$. Suppose t is $\bar{y}$-linear (linear with respect to $\bar{y} = (y_1, \ldots, y_k)$ assuming that the tree sort is $\mathrm{T}^k\langle\sigma\rangle$, recall that nonlinearity is allowed with respect to the label variable x). Then, $t|_\pi = l[t_0, \bar{s}]$, where $\bar{s} = (s_1, \ldots, s_k)$, is also $\bar{y}$-linear and

$$y_i \in FV(\bar{s}) \Longrightarrow y_i \notin FV(t[\pi \leftarrow \epsilon]) \\ y_i \in FV(s_j) \Longrightarrow y_i \notin FV(s_{j'}) \qquad (\text{for } j \neq j')$$

Since the rule $l[x, \bar{y}] \xrightarrow{\psi[x]}_{R'_B} r[x, \bar{y}]$ is $\bar{y}$-linear, it follows that $r[t_0, \bar{s}]$ is also $\bar{y}$-linear and consequently $t[\pi \leftarrow r[t_0, \bar{s}]]$ is $\bar{y}$-linear. Therefore, by linearity of A and by induction on the length of R'_B-derivations, all terms in $(\varphi, t)\downarrow_{R'_B}$, are $\bar{y}$-linear. It follows that each rule added to R is $\bar{y}$-linear, $\therefore A \circ B$ is linear. ⊠

4.2 Equivalence of linear single-valued STTs

Equivalence checking of finite transducers is undecidable when the possible number of outputs for a given input is unbounded [17, 21]. The case that is practically more directly relevant for us is when transducers are single-valued, since this case corresponds closely to functional transformations computed by concrete programs over structured data (possibly over a restricted input domain). For (top-down) tree transducers it is known that equivalence is decidable for the

single-valued case [8, 13], or more generally, for the *finite-valued* case [31] (when there exists k such that, for all t, $|\mathbf{T}_A(t)| \leq k$). Here we investigate the more restricted equivalence problem for *linear single-valued* STTs as the practically most common case, while the generalization to either nonlinear or finite-valued STTs is left as a future research topic.

STTs A and B are *equivalent* if $\mathbf{T}_A = \mathbf{T}_B$. Let $Dom(A) \stackrel{\text{def}}{=} \{t \mid \mathbf{T}_A(t) \neq \emptyset\}$. For a single-valued STT A and $t \in Dom(A)$ we write $A(t) = u$ for $\mathbf{T}_A(t) = \{u\}$. In the following let $A^{\mathrm{T}\langle\iota\rangle/\mathrm{T}\langle o\rangle}$ and $B^{\mathrm{T}\langle\iota\rangle/\mathrm{T}\langle o\rangle}$ be fixed linear single-valued STTs. Equivalence of A and B reduces to two separate decision problems:

1. *Domain equivalence*: $Dom(A) = Dom(B)$.
2. *Partial equivalence* $A \cong B$: for all $t \in Dom(A) \cap Dom(B)$, $A(t) = B(t)$.

Note that both problems are independent of each other and together imply equivalence. Domain equivalence requires the notion of a *symbolic tree automaton* (*STA*) that is an STT such that each rule is either $q(\epsilon) \to \epsilon$ or a linear rule $q(f(x, y_1, \ldots, y_k)) \xrightarrow{\varphi[x]} f(x, q_1(y_1), \ldots, q_k(y_k))$, i.e., the notion of the output tree is obsolete. The STA for $Dom(A)$ can be constructed directly from the STT A. By a *label theory* we mean a quantifier free set of formulas that is closed under substitutions, Boolean operations and equality, and allows free variables of the label sort.

Deciding equivalence of two STAs A and B is a generalization of the equivalence problem of tree automata [16] and is decidable modulo decidability of the label theory. The equivalence algorithm of symbolic tree automata uses the result that symbolic tree automata are closed under intersection and complement [33], that is a generalization of corresponding closure properties of symbolic automata [34]. We use the following proposition below.

Proposition 1. *Equivalence of symbolic tree automata is decidable modulo a decidable label theory.*

For many practical considerations, domain equivalence is not as important as partial equivalence because the transductions of A and B are known to correspond to *total* functions from $\mathcal{U}^{\mathrm{T}\langle\iota\rangle}$ to $\mathcal{U}^{\mathrm{T}\langle o\rangle}$, i.e., $Dom(A) = Dom(B) = \mathcal{U}^{\mathrm{T}\langle\iota\rangle}$, reflecting a *robustness* assumption of the underlying programs.

In the following we develop a practical algorithm for deciding partial equivalence $A \cong B$. First, we adjust the formalism of linear STTs a bit so that it is technically better suited for the purposes here, by separating the acceptance condition of the input term from the construction of the output term. For ease of presentation assume $k = 2$ (i.e., the input domain is $\mathrm{T}^2\langle\iota\rangle$). For each rule

$$p(f(x, y_1, y_2)) \xrightarrow{\varphi[x]} r[x, p_1(y_1), p_2(y_2)],$$

where $r[x, z_1, z_2]$ is basic (z_i is the transformation $p_i(y_i)$), is represented by the following *transition* whose i'th target component is the state that transforms y_i:

$$p \xrightarrow{\varphi[x]/r[x,z_1,z_2]} (p_1, p_2)$$

If z_i does not occur in r we use a special *sink* state $p_i = p^*$ with transition

$$p^* \xrightarrow{true/\epsilon} (p^*, p^*)$$

For each rule $p(\epsilon) \to e$ we say that p is *final* with a *final output* e, denoted by $p \xrightarrow{/e}$. In particular, $p^* \xrightarrow{/\epsilon}$. For example,

$$p(f(x, y_1, y_2)) \xrightarrow{true} f(x, \epsilon, q(y_1)) \quad \text{corresponds to} \quad p \xrightarrow{true/f(x,\epsilon,z_1)} (q, p^*).$$

Note that all input trees are accepted from p^*, e.g., in the above example y_2 can be an arbitrary input tree. Note also that the given transition view is not possible for arbitrary nonlinear STTs.

We assume that A and B are *clean*, i.e., contain no rules with unsatisfiable guards. We also assume that A and B have no *unreachable states* and no *deadends*, where an unreachable state is a state such that no derivation from the initial state can reach it, and a deadend is a state p that is reachable but is not final and, for all transitions $p \to (p_1, p_2)$, either p_1 or p_2 is a deadend. The corresponding decision problems are classical forward and backward reachability algorithms that are linear in the number of states.

The partial equivalence algorithm uses the notion of a *product* $A \times B$ of A and B that is intuitively a *2-output-STT* whose definition is based on the transition view of A and B. $A \times B$ has states $Q_A \times Q_B$ and its transitions are constructed as follows, where $\bar{z}^A$ (resp. $\bar{z}^B$) is a unique renaming of each z_i by z_i^A (resp. z_i^B),

$$\left.\begin{array}{l} p \xrightarrow{\varphi[x]/t[x,\bar{z}]}_A (p_1, p_2) \\ q \xrightarrow{\psi[x]/u[x,\bar{z}]}_B (q_1, q_2) \end{array}\right\} \Longrightarrow \langle p, q\rangle \xrightarrow{\varphi\wedge\psi[x]/(t[x,\bar{z}^A],u[x,\bar{z}^B])}_{A\times B} (\langle p_1, q_1\rangle, \langle p_2, q_2\rangle)$$

$$\left.\begin{array}{r} p \xrightarrow{/a}_A \\ q \xrightarrow{/b}_B \end{array}\right\} \Longrightarrow \langle p, q\rangle \xrightarrow{/(a,b)}_{A\times B}$$

Unsatisfiable guards, unreachable states, and deadends are eliminated from $A \times B$ (where the first two are by virtue of constructing $A \times B$ by using DFS).

Let $A^{\langle p,q\rangle}(v)$ denote the A-output of $A \times B$ that is produced starting from state $\langle p, q\rangle$ for any v that is accepted from that state. Similarly for B. The following property follows from the definitions.

$$(\forall t \in Dom(A) \cap Dom(B)) \;\; A^{\langle q_A^0, q_B^0\rangle}(t) = B^{\langle q_A^0, q_B^0\rangle}(t) \Leftrightarrow A(t) = B(t)$$

A *tree context* is a tree term with exactly one occurrence of a special free variable $\bullet$. A *promise* of a state $\langle p, q\rangle$ is a pair $\langle \alpha[\bullet], \beta[\bullet]\rangle$ of tree contexts where $\alpha = \bullet$ or $\beta = \bullet$, such that there exists a derivation in $A \times B$ for some input tree t and position π in t that reaches $\langle p, q\rangle$ at π and, for some output position π', the π'-traces of the outputs from A and B are equal and the remaining outputs from position π' in A and B are $\alpha[A^{\langle p,q\rangle}(t|_\pi)]$ and $\beta[B^{\langle p,q\rangle}(t|_\pi)]$ respectively. Two promises $\langle \alpha_1[\bullet], \beta_1[\bullet]\rangle$ and $\langle \alpha_2[\bullet], \beta_2[\bullet]\rangle$ *conflict* if $\alpha_1[u] \neq \alpha_2[u]$ or $\beta_1[u] \neq \beta_2[u]$ for some tree u. Otherwise, we say that the promises *match*. Note that matching really means that the promises are identical trees (modulo equality of labels).

Example 5. Consider two SFTs A and B over $\mathrm{T}^2\langle \textsc{int} \rangle$ where $Q_A = \{p\}$ and $Q_B = \{q\}$ that contain the following transitions:

$$p \xrightarrow{/\epsilon}_A, \quad p \xrightarrow{x \leq 2/f(x,f(x,z_1,\epsilon),z_2)}_A (p,p), \quad q \xrightarrow{/\epsilon}_B, \quad q \xrightarrow{x \geq 2/f(x,z_1,z_2)}_B (q,q).$$

Then $A \times B$ contains the following transitions

$$\langle p,q \rangle \xrightarrow{/(\epsilon,\epsilon)}_{A\times B}, \quad \langle p,q \rangle \xrightarrow{x=2/(f(x,f(x,z_1^A,\epsilon),z_2^A),f(x,z_1^B,z_2^B))}_{A\times B} (\langle p,q \rangle, \langle p,q \rangle)$$

State $\langle p,q \rangle$ has a promise $\langle \bullet, \bullet \rangle$ for position ε since $\langle p,q \rangle$ is the initial state. It also has a conflicting promise $\langle f(2,\bullet,\epsilon), \bullet \rangle$ due to the following. Consider the input tree $t = f(2,t_1,t_2)$ and position $\pi = \mathbf{1}$ in t. After a single step derivation in $A \times B$ we get that $A(t) = f(2, f(2, A(t_1), \epsilon), A(t_2))$ and $B(t) = f(2, B(t_1), B(t_2))$. Thus, for the output position $\pi' = \mathbf{1}$ we have that, the π'-traces of $A(t)$ and $B(t)$ are equal, $A(t)|_{\pi'} = f(2, A^{\langle p,q \rangle}(t|_\pi), \epsilon)$ and $B(t)|_{\pi'} = B^{\langle p,q \rangle}(t|_\pi)$. ⊠

A state $\langle p,q \rangle$ is *input dependent for* A if there exist $v_1 \neq v_2$ such that $A^{\langle p,q \rangle}(v_1) \neq A^{\langle p,q \rangle}(v_2)$. Similarly for B. A state $\langle p,q \rangle$ is *input dependent* if it is input dependent for both A and B.

We make use of the following key lemma in the main theorem that provides us with an argument to detect $A \not\cong B$ in $O(|A \times B|)$ number of steps. The lemma is not constructive; it does not provide a witness t such that $A(t) \neq B(t)$.

Lemma 1 (Promise). *If an input dependent state $\langle p,q \rangle$ is reached with two conflicting promises then $A \not\cong B$.*

Proof. Suppose there is a state $\langle p,q \rangle$ with conflicting promises $\langle \alpha_1[\bullet], \beta_1[\bullet] \rangle$ and $\langle \alpha_2[\bullet], \beta_2[\bullet] \rangle$. Since $\langle p,q \rangle$ is reachable and not a deadend, there exist input trees u_1 and u_2 where $u_1|_{\pi_1} = v$ and $u_2|_{\pi_2} = v$ for some positions π_1 and π_2, and v is an input term accepted from $\langle p,q \rangle$, and

- $\langle p,q \rangle \mapsto \langle \alpha_1[\bullet], \beta_1[\bullet] \rangle$ is reached at some position π_1' in the outputs from u_1,
- $\langle p,q \rangle \mapsto \langle \alpha_2[\bullet], \beta_2[\bullet] \rangle$ is reached at some position π_2' in the outputs from u_2.

By single-valuedness of A and B there exist $v^A = A^{\langle p,q \rangle}(v)$ and $v^B = B^{\langle p,q \rangle}(v)$ such that

$$A(u_1)|_{\pi_1'} = \alpha_1[v^A],\ A(u_2)|_{\pi_2'} = \alpha_2[v^A],\ B(u_1)|_{\pi_1'} = \beta_1[v^B],\ B(u_2)|_{\pi_2'} = \beta_2[v^B].$$

Suppose $A \cong B$. Then $\alpha_1[v^A] = \beta_1[v^B]$ and $\alpha_2[v^A] = \beta_2[v^B]$. We reach a contradiction by case analysis. Note that the cases make use of the assumption that in each promise at least one of the terms is a $\bullet$.

- Case $\alpha_1 = \bullet, \beta_1 = \bullet, \alpha_2 = \bullet, \beta_2 \neq \bullet$: Then $v^B = v^A = \beta_2[v^B]$, but $\beta_2 \neq \bullet$.
- Case $\alpha_1 \neq \bullet, \beta_1 = \bullet, \alpha_2 = \bullet, \beta_2 \neq \bullet$: Then $\alpha_1[v^A] = v^B, v^A = \beta_2[v^B]$, but $\alpha_1[\beta_2[v^B]] \neq v^B$.

– Case $\alpha_1 = \bullet, \beta_1 \neq \bullet, \alpha_2 = \bullet, \beta_2 \neq \bullet$: Then $\beta_1[v^B] = v^A = \beta_2[v^B]$. This is only possible if $\beta_1[t] = \beta_2[t]$ for a fixed tree t (e.g. $\beta_1 = f(t_0, \bullet, t)$ and $\beta_2 = f(t_0, t, \bullet)$) or else $\langle \alpha_1, \beta_1 \rangle$ and $\langle \alpha_2, \beta_2 \rangle$ match. Thus $v^B = t$. By input independence of $\langle p, q \rangle$ for B we can choose v and $v_0 \neq v$ so that $v^B \neq v_0^B = B^{\langle p,q \rangle}(v_0)$. But this contradicts that $v_0^B = t$ must hold.

The remaining cases are symmetrical. ⊠

We apply the following normalization on $A \times B$ before turning to the main algorithm. First, we effectively decide if a state $\langle p, q \rangle$ is input dependent for A (resp. B) and, otherwise compute a concrete term $t_A^{\langle p,q \rangle}$ (resp. $t_B^{\langle p,q \rangle}$) such that for all v, $A^{\langle p,q \rangle}(v) = t_A^{\langle p,q \rangle}$ (resp. $B^{\langle p,q \rangle}(v) = t_B^{\langle p,q \rangle}$). Next, for each transition

$$\langle p, q \rangle \xrightarrow{\varphi/(t,u)} (\langle p_1, q_1 \rangle, \langle p_2, q_2 \rangle)$$

if $\langle p_i, q_i \rangle$ is not input dependent for A, replace z_i in t by $t_A^{\langle p_i,q_i \rangle}$, similarly for u and B. In the following assume:

(*) *If an output variable z_i occurs in t (resp. u) then $\langle p_i, q_i \rangle$ is input dependent for A (resp. B).*

The main algorithm is a search procedure (that can be implemented using DFS). There is a set Q of reached states annotated with promises. Initially $Q = \{\langle q_A^0, q_B^0 \rangle \mapsto \langle \bullet, \bullet \rangle\}$. Given an unexplored reached state $\langle p, q \rangle \mapsto \langle \alpha[\bullet], \beta[\bullet] \rangle$ the following steps are performed.

Check final outputs: If $\langle p, q \rangle \xrightarrow{/(a,b)}$ and if $\alpha[a] \neq \beta[b]$ then $\mathbf{A} \not\cong \mathbf{B}$.
Check transitions: Perform the following steps for each transition

$$\langle p, q \rangle \xrightarrow{\varphi[x]/(t[x,z_1^A,z_2^A],u[x,z_1^B,z_2^B])} (\langle p_1, q_1 \rangle, \langle p_2, q_2 \rangle)$$

Unify $\alpha[t]$ with $\beta[u]$. If a unifier does not exist then $\mathbf{A} \not\cong \mathbf{B}$. Else, the result is a pair $(C[x], \theta)$ where C is a conjunction of equality constraints on labels and θ is a substitution from those $\bar{z}$ that occur in t or u to tree terms.

Validate labels: If $\varphi \wedge \neg C$ is satisfiable then there exists a label x such that $\alpha[t[x, \bar{z}^A]] \neq \beta[u[x, \bar{z}^B]]$ for all $\bar{z}^A$ and $\bar{z}^B$. Thus $\mathbf{A} \not\cong \mathbf{B}$.

Validate trees: Suppose $z_1^A \mapsto v[x, z_1^B, z_2^B] \in \theta$. (The other cases are symmetrical.) We proceed by case analysis on v.

Independent term: Assume v does not contain z_1^B. Suppose v' is an arbitrary ground instance of v. By (*), there exist two outputs $a_1 \neq a_2$ for z_1^A and therefore, either $t[\_, a_1, \_] \neq t[\_, v', \_]$ or $t[\_, a_2, \_] \neq t[\_, v', \_]$. Thus $\mathbf{A} \not\cong \mathbf{B}$.

Mixed dependency: Assume v contains z_1^B and also z_2^B. By using (*), there exist two different outputs for z_2^B independent of z_1^A and x. Thus, there exist two conflicting promises for $\langle p_1, q_1 \rangle$. By the Promise lemma, $\mathbf{A} \not\cong \mathbf{B}$.

Multiple labels: Assume $v[x, z_1^B]$ contains x and z_1^B but not z_2^B. Check if any possible value for x makes any difference in v. If

$$\varphi[x_1] \wedge \varphi[x_2] \wedge v[x_1, \epsilon] \neq v[x_2, \epsilon]$$

is satisfiable then there exist two conflicting promises for $\langle p_1, q_1 \rangle$. By the Promise lemma, $\mathbf{A} \ncong \mathbf{B}$.

Dependent term: Let t_0 be any value such that $\varphi[t_0]$ is true. Note that $v[t_0, \bullet]$ is the same independent of t_0 by the previous step. Let $\alpha_1 = \bullet$, $\beta_1 = v[t_0, \bullet]$:

1. If $\langle p_1, q_1 \rangle \notin Q$ then add $\langle p_1, q_1 \rangle \mapsto \langle \alpha_1, \beta_1 \rangle$ to Q and push $\langle p_1, q_1 \rangle$.
2. else let $\langle \alpha_2, \beta_2 \rangle = Q(\langle p_1, q_1 \rangle)$. If $\langle \alpha_1, \beta_1 \rangle$ and $\langle \alpha_2, \beta_2 \rangle$ conflict (are not identical), then, by the Promise lemma, $\mathbf{A} \ncong \mathbf{B}$.

Partial equivalence: The search is exhaustive and establishes that $\mathbf{A} \cong \mathbf{B}$.

Recall that a *label theory* is a quantifier free set of formulas that is closed under substitutions, Boolean operations and equality, and allows free variables of the label sort. A typical label theory is quantifier free integer linear arithmetic. Note that decidability of a theory refers to decidability of the problem of checking satisfiability of a given formula in the theory.

Theorem 2 (Equivalence). *Equivalence of linear single-valued STTs is decidable over a decidable label theory.*

Proof. Termination of the partial equivalence checking algorithm follows from finiteness of $Q_{A \times B}$ and decidability of the assumed label theory. Partial correctness of the algorithm and how the Promise lemma is used follows from the description of the core steps of the algorithm. Decidability of domain equivalence follows from Proposition 1. ⊠

Note that the satisfiability checks that are actually performed during the search require the use of *at most one* free label variable and no other free variables. In particular, the statisfiability check in the **Multiple labels** step above, that is expressed using two variables, can be replaced (given any value a such that $\varphi[a]$ holds) with the satisfiability check of the formula $\varphi[x] \wedge v[x, \epsilon] \neq v[a, \epsilon]$ containing at most one free variable x. This observation is relevant in order to provide a precise complexity bound for the given algorithm, provided that the complexity of the used label theory over at most one free variable is known. The algorithm can be implemented using any SMT solver or constraint solver as an oracle that supports satisfiability checking and model generation, such as the SMT solver Z3 [6].

4.3 Checking Non-equivalence Symbolically

The algorithm produces at most $|Q_A \times Q_B|$ states and requires examining at most $|R_A| \cdot |R_B|$ transitions. This bound obviously does not work when comparing tree

transducers whose number of outputs is unbounded. In practice, however, we can use a common symbolic algorithm that unfolds an STT. In the general case it can be used as a semi-decision algorithm for non-equivalence. Given a transducer A, that does not contain ϵ loops, we can encode a predicate $Acc_A(q_A^0, t, s, n)$, such that A takes the term t and produces the term s with at most n transitions along any given branch. Non-equivalence can then be checked by showing that

$$\exists t, s, n \;.\; \begin{pmatrix} & (Acc_A(q_A^0, t, s, n) \wedge \neg Acc_B(q_B^0, t, s, n)) \\ \vee & (\neg Acc_A(q_A^0, t, s, n) \wedge Acc_B(q_B^0, t, s, n)) \end{pmatrix}.$$

The definition is given by:

$$Acc_A(q, f(t_0, t_1, \ldots, t_k), s, n) \equiv \bigvee_{\tau \in R_A} \begin{pmatrix} n > 0 \wedge \;\; \varphi[t_0] \wedge \\ s = u[t_0, \ell_1(s), \ldots, \ell_k(s)] \wedge \\ \bigwedge_{i=1}^{k} Acc_A(q_i, t_i, \ell_i(s), n-1) \end{pmatrix}$$

$$Acc_A(q, \epsilon, \epsilon, n) \equiv true$$

where, as usual, τ is of the form $q(f(x, y_1, \ldots, y_k)) \xrightarrow{\varphi} u[x, q_1(y_1), \ldots, q_k(y_k)]$, and $\ell_1(s)$ selects the subterm of s corresponding to the path supplied in u. The formulas produced by unfolding Acc_A are always ground, and satisfiability of the formulas can be checked using the background label theory together with the theory of algebraic data-types. For single valued STTs we can fix n to $|Q_A \times Q_B|$ to bound unfolding; for general STTs we can convert the definition into first-order formulas whose instantiations correspond to step-wise unfoldings of the transition relation.

5 Related Work

Tree transducers and various extensions thereof provide a syntax-directed view of studying different formal models of transformations over tree structured data [15]. Top-down tree transducers were originally introduced in [30, 32] for studying properties of syntax-directed translations. The handbook [16] provides a uniform treatment of foundational properties of tree transducers and relations to context-free languages. Basic compositionality results of tree transducers were established in [3, 7].

Decidability of equivalence of single-valued top-down tree transducers follows from the decidability result of single-valuedness of top-down tree transducers [8, 13]. A specialized method for checking equivalence of *deterministic* top-down tree transducers is provided in [5]. Decision problems, e.g. equivalence, for specific classes of tree transducers are often based on establishing unique normal forms and considering deterministic transducers, including string transducers [4], top-down tree transducers [11], and top-down tree-to-string transducers [24].

Several extensions of top-down tree transducers have been studied in the literature (the following list is not exhaustive). Extended top-down tree transducers allow nonflat left-hand sides in rules [2]. Attributed tree-transducers

describe parse trees in attribute grammars [14]. Macro tree-transducers incorporate the notion of implicit tree contexts [12] and have been studied in the context of analysis of XML transformation languages, with macro attributed tree-transducers [15], multi-return macro tree transducers [22], and macro forest transducers [29] as further extensions. Pebble tree transducers were introduced for type checking XML query languages [26] and are extended to pebble macro tree transducers in [10]. Formal relationships between monadic second order logic and macro tree transducers is studied in [9]. Extended top-down tree transducers were recently studied in the context of natural language processing, where it is shown that several interesting cases are not closed under composition [25]. Higher-order multi-parameter tree transducers [23] allow possibly infinite trees in the output and can be applied to higher-order recursion schemes. A related notion of pattern-matching recursion schemes is introduced in [28] to model functional programs that manipulate algebraic data-types.

The above generalizations are all proper extensions of the basic top-down tree transducer model. To the best of our knowledge, none of the extensions has considered a symbolic representation of the transducers modulo a given label theory. The work in [27] introduces the first symbolic generalization of a finite state (string) transducer called a *predicate-augmented finite state transducer* in the context of natural language processing. A symbolic representation of finite (string) transducers modulo a given label theory, called *symbolic finite transducers*, is introduced in [19] in order to encode string sanitization operations over large (possibly infinite) alphabets for web security analysis. The results here extend the algorithms of the string case to trees, in order to symbolically represent transductions over tree structured data such as Html or XML documents. *Streaming transducers* [1] provide another recent symbolic extension of finite transducers where the label theories are restricted to be total orders.

6 Conclusions

We intend to investigate several extensions of our main results for STTs. Single-valued deterministic STTs correspond directly to first-order functional programs. One direction is to develop ground decision procedures for first-order functional programs given as STTs. On the other hand, first-order functional programs are of course much more expressive than STTs (and equivalence of first-order functional programs is Π^1_1 complete). For example list-reversal is not expressible as an STT. It is tempting to extend our results to more general functional programs. Recent work on counter-example guided refinement for verification of higher-order functional programs [28], for instance, is based on repeated refinements starting from tree transducers. Using STTs instead could allow using SMT tools for analyzing functional programs. A simpler extension of our results for STTs is to consider equivalence of non-linear STTs. We conjecture that our Theorem 2 can be extended to nonlinear and single-valued STTs, but our proof cannot be used directly for this case: non-linearity creates dependencies across several states from the same transducer that has to be recorded in a product

construction. We also conjecture the theorem can be extended to nonlinear and finite-valued STTs. However, a generalization to the finite-valued case is not expected to be straightforward, because the corresponding generalization of decidability of equivalence for finite-valued tree transducers [31] uses results from combinatorics and is mathematically challenging.

References

1. R. Alur and P. Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *38th ACM SIGACT-SIGPLAN Symposium on Princples of Programming Languages (POPL'11)*, pages 599–610. ACM, 2011.
2. A. Arnold and M. Dauchet. Bi-transductions de forêts. In *Proc. 3rd International Colloquium on Automata, Languages and Programming (ICALP'76)*, pages 74–86, Edinburgh, 1976. Edinburgh University Press.
3. B. S. Baker. Composition of top-down and bottom-up tree transductions. *Inform. and Control*, 41:186–213, 1979.
4. C. Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.
5. B. Courcelle and P. Franchi-Zannettacchi. Attribute grammars and recursive program schemes. *Theoretical Computer Science*, 17:163–191, 1982.
6. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS'08*, LNCS. Springer, 2008.
7. J. Engelfriet. Bottom-up and top-down tree transformations – a comparison. *Math. Systems Theory*, 9:198–231, 1975.
8. J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R. V. Book, editor, *Formal Language Theory*, pages 241–286. Academic Press, New York, 1980.
9. J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and mso definable tree translations. *Information and Computation*, 154:34–91, 1999.
10. J. Engelfriet and S. Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39:2003, 2003.
11. J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *Journal of Computer and System Science*, 75(5):271–286, 2009.
12. J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. and Syst. Sci.*, 31:71–146, 1985.
13. Z. Esik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.
14. Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.
15. Z. Fülöp and H. Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. EATCS. Springer, 1998.
16. F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.
17. T. Griffiths. The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines. *J. ACM*, 15:409–413, 1968.
18. W. Hodges. *Model theory*. Cambridge Univ. Press, 1995.
19. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Fast and precise sanitizer analysis with Bek. In *20th USENIX Security Symposium*, pages 1–16, San Francisco, CA, August 2011. USENIX Association.

20. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
21. O. Ibarra. The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. *SIAM Journal on Computing*, 4:524–532, 1978.
22. K. Inaba and H. Hosoya. Multi-return macro tree transducers. In *Proc. 6th ACM SIGPLAN Workshop on Programming Language Technologies for XML*, San Francisco, California, January 2008.
23. N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL'10, pages 495–508. ACM, 2010.
24. G. Laurence, A. Lemay, J. Niehren, S. Staworko, and M. Tommasi. Normalization of sequential top-down tree-to-word transducers. In *Language and Automata Theory and Applications (LATA)*, LNCS, pages 352–363. Springer, 2011.
25. A. Maletti, J. Graehl, M. Hopkins, and K. Knight. The power of extended top-down tree transducers. *SIAM J. Comput.*, 39:410–430, June 2009.
26. T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *Proc. 19th ACM Symposium on Principles of Database Systems (PODS'2000)*, pages 11–22. ACM, 2000.
27. G. V. Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4:263–286, 2001.
28. C.-H. L. Ong and S. J. Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *38th ACM SIGACT-SIGPLAN Symposium on Princples of Programming Languages (POPL'11)*, pages 587–598. ACM, 2011.
29. T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89(3):141–149, 2004.
30. W. C. Rounds. Context-free grammars on trees. In *Proc. ACM Symp. on Theory of Comput.*, pages 143–148. ACM, 1969.
31. H. Seidl. Equivalence of finite-valued tree transducers is decidable. *Math. Systems Theory*, 27:285–346, 1994.
32. J. W. Thatcher. Generalized sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.
33. M. Veanes and N. Bjørner. Symbolic tree automata. Submitted to *Information Processing Letters*, 2011.
34. M. Veanes, N. Bjørner, and L. de Moura. Symbolic automata constraint solving. In C. Fermüller and A. Voronkov, editors, *LPAR-17*, volume 6397 of *LNCS*, pages 640–654. Springer, 2010.
35. M. Veanes, P. de Halleux, and N. Tillmann. Rex: Symbolic regular expression explorer. In *Third International Conference on Software Testing, Verification and Validation (ICST 2010)*, pages 498–507. IEEE Computer Society, 2010.
36. S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1997.