The Logic in Computer Science Column

by

Yuri Gurevich

Microsoft Research
One Microsoft Way, Redmond WA 98052, USA
gurevich@microsoft.com

Foundations of Finite Symbolic Tree Transducers

Margus Veanes   Nikolaj Bjørner[*]

Abstract

Finite transducers on trees are fundamental to computer science. They form the basis of many applications that manipulate strings and trees. The conventional representation of finite transducers assume a finite set of states and a finite alphabet. Classical algorithms and representations make essential use of both of these assumptions. In many cases, the complexity of the algorithms is computed based on the number of states and alphabet size. But how important are these assumptions really for the main operations and decision problems? We have recently pursued applications of finite transducers in the context of web security as a foundation for sanitization of potentially malicious data. For these applications we have found that lifting the finite alphabet restriction to be useful to enable efficient symbolic analysis and we have developed symbolic counter-parts of the main classical operations on finite automata. We here define Symbolic Tree Transducers as a generalization of Regular Transducers as finite state input-output tree automata with logical constraints over a background theory. The background theory

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

is a parameter of the formalization. We examine key closure properties of Symbolic Tree Transducers and we develop a composition algorithm and an equivalence decision procedure for single-valued transducers.

1 Introduction

State machines are fundamental to computer science. General computations are captured by Abstract State Machines [20]. An important special case is captured by Finite State Machines. The languages accepted by finite state machines are the ubiquitous regular languages. 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. Finite tree transducers accept regular languages and produce regular languages. As usual, a language is characterized by a set of words or trees that are labeled by symbols from an alphabet. From a computational point of view, letters from a finite alphabet form the nucleus of the finite state machines: letters can be stored and recognized in unit space and in unit time. Furthermore, finite state machines are amenable to operations and analysis that eludes more general notions of computation. The membership problem, whether a word or tree is accepted by a finite state machine, is decidable. Finite state machines are closed under the Boolean operations of union, intersection, complementation. They form an effective Boolean Algebra. Furthermore, the question of whether two finite state machines accept the same language, known as the equivalence question, is also decidable. The algorithmic complexity of most fundamental decision problems over finite automata and transducers depend on the *size of the alphabet k* as well as *the number of states n*:[1] intersection of nondeterministic finite word automata has complexity $O(kn^2)$ (c.f. [24, p. 59]), determinization of nondeterministic finite word automata is $O(k2^n)$ (c.f. [24, Theorem 2.1]), (implying the upper bound $O(k2^n)$ for deciding the equivalence of nondeterministic finite word automata), and minimization of deterministic finite word automata is $O(k\,n\log n)$ (c.f. [24, Exercise 3.30]). Under some mild restrictions, equivalence can also be checked between two finite state transducers.

In the context of analysis and operations, is the finite alphabet assumption essential? Of course, we cannot expect unit storage space and access time for letters from an infinite alphabet, but what about analysis? Our main results establish how the representation of alphabets can be generalized. In fact, in practice, we have found that a symbolic representation of even finite alphabets can be an advantage

[1] Some algorithms do not depend on the size of the alphabet, most notably: *epsilon-elimination*, *unreachable-state-elimination*, and *dead-end-elimination* (a dead-end is a noninitial state from which no final state is reachable).

for these operations. We develop the results for the most general case of finite tree transducers, and we call these *Symbolic Tree Transducers* (STTs). The alphabets of STTs 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 composing STTs and equivalence checking for 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. In our prior work [41, 22] on symbolic string recognizers and transducers we took advantage of this observation. We here investigate the case of the more expressive class of *tree* transducers. We first examined symbolic tree transducers in [40], where we established that equivalence checking for the special case of *linear* single-valued tree transducers was decidable. Roughly, a tree transducer is linear if the state transition relation does not require more than one state per sub-tree. The decision procedure we developed there created a product automaton and checked for paths and loops in the resulting product automaton that could produce a counter-example to equivalence. We left it as an open problem whether the restriction on linearity could be lifted. The question is settled in the affirmative here. We provide a decision procedure for the equivalence problem of generial single-valued symbolic tree transducers. They don't have to be linear. This paper furthermore provides the foundations for symbolic tree transducers. We show that symbolic regular tree languages form an *effective Boolean Algebra* provided the alphabet is also effective. All decision procedures are of course provided modulo decidability of the symbolic background component.

This paper can be read as a self-contained introduction and overview of symbolic tree automata and tree transducers. While it becomes technical, we only use elementary notions, well-known from classical automata theory curricula [24]. It builds on top of results from [40] and we include several definitions and relevant examples from that work when useful.

The rest of the paper is organized as follows. Section 2 discusses some of the background for our study of tree transducers over a symbolic, or parametric, alphabet. As background for the technical development we recall preliminaries in Section 3, and then develop the foundations for Symbolic Tree Automata in Section 4, and Section 5 introduces Symbolic Tree Transducers. Our new algorithm for equivalence checking of single-valued STTs is provided in Section 6. Section 7 contains conclusions.

2 Parametricity

Before devling into technical details let us discuss a main theme of this paper. We develop Symbolic Tree Transducers that are finite state machines modulo a method for checking constraints on the alphabets used to label the trees. The results are parametric in the underlying representation of the alphabet and we identify the sufficient conditions on constraint solving for the labeling. There is also a branch of theorem proving dedicated to satisfibility of logical formulas modulo background theories. It is known as Satisfiability Modulo Theories (SMT). Here, the interpretation of the logical formulas are provided modulo a suitable background theory. The two paradigms are not disconnected: in fact the parametric alphabets in the symbolic tree automata and transducers can be instantiated by formulas that can be solved using SMT solvers. This has also been the method of choice in the Automata[2] tools available from Microsoft Research. The tool uses the state-of-the-art SMT solver Z3 [8].

Our results on STTs can also be seen as connected to the *modular* theory combination problem. The problem of modularly combining solvers for different theories is identifying the minimal necessary and sufficient interfaces between the solvers. Our procedures for symbolic tree automata are modular in a very transparent way. The algorithms on automata use symbolic solvers as black box oracles. The interface to the symbolic solver comprises checking logical satisfiability on transition guards.

2.1 Automata and Transducers

Before covering work on symbolic automata and transducers, let us here briefly recall some of the important references on classical automata theory. Tree transducers and various extensions thereof provide a syntax-directed view of studying different formal models of transformations over tree structured data [17]. Top-down tree transducers were originally introduced in [35, 37] for studying properties of syntax-directed translations. Basic compositionality results of tree transducers were established in [4, 9]. The handbook [18] provides a uniform treatment of foundational properties of tree transducers and relations to context-free languages. A newer handbook on tree automata has been available as an online resource for several years now [6]. It is a comprehensive source for recent results on tree automata.

Decidability of equivalence of single-valued top-down tree transducers follows from the decidability result of single-valuedness of top-down tree transducers [10, 15]. A specialized method for checking equivalence of *deterministic* top-

[2] http://rise4fun.com/bek

down tree transducers is provided in [7]. 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 [5], top-down tree transducers [13], and top-down tree-to-string transducers [29].

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 [3]. Attributed tree-transducers describe parse trees in attribute grammars [16]. Macro tree-transducers incorporate the notion of implicit tree contexts [14] and have been studied in the context of analysis of XML transformation languages, with macro attributed tree-transducers [17], multi-return macro tree transducers [26], and macro forest transducers [34] as further extensions. Pebble tree transducers were introduced for type checking XML query languages [31] and are extended to pebble macro tree transducers in [12]. Formal relationships between monadic second order logic and macro tree transducers is studied in [11]. 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 [30]. Higher-order multi-parameter tree transducers [28] 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 [33] to model functional programs that manipulate algebraic data-types.

The dual extension of finite automata and transducers to ours maintains a finite alphabet and instead admits an infinite number of states. For example, timed automata [2] admit an infinite number of reachable states, but maintain decidability for key problems because the states form a finite quotient with respect to the transition relations.

2.2 Symbolic Automata and Transducers

Many connections to classical automata theory and symbolic versions of automata have surfaced in different variants. The corresponding symbolic generalization of classical (Rabin-Scott) automata is originally studied in [32] where the motivation comes from *computational linguistics*. There, the symbolic generalization of a finite state (string) transducer is called a *predicate-augmented finite state transducer* and it is used in the context of natural language processing. The MONA tool [27] uses automata over a finite but large alphabet. It uses multi-terminal binary decision diagrams to label transition relations. This allows often encoding with exponential savings alphabets of size 2^n when they are represented using n binary values. The notion of symbolic automata is identified and developed in [41] where the motivation came from the need to support regular expressions in *parameterized unit testing* [38] and like-expressions (which are very much like regular expressions) in *database query analysis* [42]. Such an extension seems straightforward at first glance, but raises many challenging questions and opens up new approaches for automata algorithm design. For example, it is shown in [23] that symbolic complementation by using a technique for minimizing symbolic representations of Boolean functions leads to significant speedups compared to existing state-of-the-art automata algorithm implementations. These techniques have also been instrumental in applications for encoding string sanitization operations over large (possibly infinite) alphabets for web security analysis [22]. *Streaming transducers* [1] provide another recent symbolic extension of finite transducers where the label theories are restricted to be total orders.

3 Preliminaries

We use basic notions from classical automata theory [24], classical logic, and model theory [21]. Our notions regarding tree transducers are consistent with [17]. For finite state (string) transducers a brief introduction is given in [43].

3.1 Background Universe

We work modulo a multi-sorted *background* universe $\mathscr{U}$. For each sort σ, $\mathscr{U}^{\sigma}$ denotes a nonempty sub-universe of $\mathscr{U}$. A *predicate over* σ or σ*-predicate* is an effective finite representation φ of a subset $[\![\varphi]\!]$ of $\mathscr{U}^{\sigma}$. Given a set of σ-predicates $\mathcal{P}(\sigma)$, we say that

$$(\mathcal{P}(\sigma), \land, \lor, \neg, \top, \bot)$$

is an *effective Boolean algebra* when for each element $\mathfrak{a} \in \mathscr{U}^{\sigma}$ there is a predicate $\varphi^{\mathfrak{a}}$ in $\mathcal{P}(\sigma)$ such that $[\![\varphi^{\mathfrak{a}}]\!] = \{\mathfrak{a}\}$, $\top, \bot \in \mathcal{P}(\sigma)$, $[\![\bot]\!] = \emptyset$, $[\![\top]\!] = \mathscr{U}^{\sigma}$, and $\mathcal{P}(\sigma)$ is effectively closed under the operations for conjunction $\land$, disjunction $\lor$, and negation $\neg$, such that

$$[\![\varphi \land \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!],\ [\![\varphi \lor \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!],\ [\![\neg\varphi]\!] = \complement([\![\varphi]\!]) = \mathscr{U}^{\sigma} \setminus [\![\varphi]\!].$$

We write $\varphi \equiv \psi$ for $[\![\varphi]\!] = [\![\psi]\!]$. Without loss of generality, we will assume that σ-predicates are formulas with a fixed free varible x of sort σ.

Example 1. *A practical example of* $\mathcal{P}(\textsc{int})$ *is the set of all quantifier free linear arithmetic formulas over integers with at most one fixed free variable* x*. For example* $[\![0 < x \land x + 1 < 3]\!] = [\![0 < x]\!] \cap [\![x + 1 < 3]\!] = \{1\}$. ⊠

Example 2. *Another practical example of* $\mathcal{P}(\textsc{rational})$ *is the set of all quantifier free linear arithmetic formulas over rationals with at most one fixed free variable* x*. For example* $[\![0 < x \land x + 1 < 3]\!] = \{\mathfrak{r} \in \mathscr{U}^{\textsc{rational}} \mid 0 < \mathfrak{r} < 2\}$. ⊠

Example 3. *A theoretical example of* $\mathcal{P}(\textsc{int})$ *is the set of all* re-*indices.* ⊠

We use an abstract generic definition of finite trees that is taylored for the symbolic extension and algorithms described below. Given a sort σ we write $\textsc{t}\langle\sigma\rangle$ for the sort of *σ-labeled trees* $\mathscr{U}^{\textsc{t}\langle\sigma\rangle}$ as the *least* set $\mathscr{T}$ such that:

- the *empty tree* $\epsilon \in \mathscr{T}$,
- if $\mathfrak{a} \in \mathscr{U}^{\sigma}$ then $\langle\mathfrak{a}\rangle \in \mathscr{T}$,
- if $\mathfrak{a} \in \mathscr{U}^{\sigma}$ and, for $k \geq 1$, $\mathfrak{t}_1, \ldots, \mathfrak{t}_k \in \mathscr{T}$ then $\langle\mathfrak{a}, \mathfrak{t}_1, \ldots, \mathfrak{t}_k\rangle \in \mathscr{T}$.

For a $(k+1)$-tuple $\bar{x} = \langle x_0, \ldots, x_k\rangle$, $k \geq 0$, k is called the *rank* of $\bar{x}$, denoted $\mathit{rank}(\bar{x})$; for $0 \leq i \leq k$, we let $\bar{x}[i] \stackrel{\text{def}}{=} x_i$. For a nonempty tree $\mathfrak{t}$, $\mathfrak{t}[0]$ is called *the label of* $\mathfrak{t}$ and, for $1 \leq i \leq \mathit{rank}(\mathfrak{t})$, $\mathfrak{t}[i]$ is called *the i'th subtree of* $\mathfrak{t}$. We define the *maximum rank* of $\mathfrak{t}$, or $\mathit{maxrank}(\mathfrak{t})$, to be:

$$\mathit{maxrank}(\mathfrak{t}) \stackrel{\text{def}}{=} \begin{cases} -1, & \text{if } \mathfrak{t} = \epsilon; \\ 0, & \text{if } \mathit{rank}(\mathfrak{t}) = 0; \\ \mathit{max}(\mathit{rank}(\mathfrak{t}), \mathit{max}_{1\leq i\leq \mathit{rank}(\mathfrak{t})}\{\mathit{maxrank}(\mathfrak{t}[i])\}), & \text{otherwise.} \end{cases}$$

When we want to be more specific about the rank, we fix k above. In particular, for $k = 2$, $\textsc{t}^2\langle\sigma\rangle$ is the sort of all *σ-labeled binary trees*.

Example 4. *For example the tree* $\mathfrak{t} = \langle 1, \langle 2, \epsilon, \epsilon, \epsilon\rangle, \langle 3\rangle\rangle$ *is an* $\textsc{int}$*-labeled tree of rank 2 and maximum rank 3. It is sometimes useful to think of the label* $\mathfrak{a}$ *of a nonempty tree together with its rank k as a unique* function symbol $f_{\mathfrak{a}}^{k}$ *of arity k. Thus,* $\mathfrak{t}$ *can be thought of as the term* $f_1^2(f_2^3(\epsilon, \epsilon, \epsilon), f_3^0)$. ⊠

A *symbolic label with input sort* σ_1 *and output sort* σ_2, or *σ_1/σ_2-label* f, is an effective representation of a function $[\![f]\!]$ from $\mathscr{U}^{\sigma_1}$ to $\mathscr{U}^{\sigma_2}$. We write $\mathcal{F}(\sigma_1 \to \sigma_2)$ for a given effective set of σ_1/σ_2-functors. Without loss of generality, we assume that f is a term of sort σ_2 that has at most one *fixed* free variable x of sort σ_1 that represents the input.

Example 5. *A practical example of* $\mathcal{F}(\textsc{int}\times\textsc{int} \to \textsc{int})$ *is the set of all terms of sort* $\textsc{int}$ *with one free variable x of sort* $\textsc{int} \times \textsc{int}$*, in the combined theory of quantifier free linear arithmetic and tuples. For example if f is the term* $x[0] + x[1]$ *then* $[\![f]\!]$ *is the addition function.* ⊠

Given a finite (possibly empty) base set Y of terms, we define *terms over* $\mathcal{F}(\sigma_1 \to \sigma_2)$ *and* Y, denoted $\mathcal{T}_Y(\mathcal{F}(\sigma_1 \to \sigma_2))$, as the least set T such that:

- $\epsilon \in T$, $Y \subseteq T$,

- if $f \in \mathcal{F}(\sigma_1 \to \sigma_2)$ then $\langle f \rangle \in T$,
- if $f \in \mathcal{F}(\sigma_1 \to \sigma_2)$ and, for $k \geq 1$, $t_1, \ldots, t_k \in T$ then $\langle f, t_1, \ldots, t_k \rangle \in T$.

Given a term t and distinct variables $x_1, \ldots, x_n$, we use the standard notation $t(x_1, \ldots, x_n)$ to indicate that all variables in t occur among the variables in $x_1, \ldots, x_n$. Substitution is defined as usual, and for a *ground* term t (term without variables), we write $[\![t]\!]$ for the corresponding concrete value.

Example 6. *Suppose $t(x, y)$ is the term $\langle 2 * x, \epsilon, \langle 3 * x, y \rangle\rangle$ over $\mathcal{F}(\textsc{int} \to \textsc{int})$ and $\{y\}$. Then $t(3, \langle 4 \rangle) = \langle 2 * 3, \epsilon, \langle 3 * 3, \langle 4 \rangle\rangle\rangle$ and $[\![t(3, \langle 4 \rangle)]\!] = \langle 6, \epsilon, \langle 9, \langle 4 \rangle\rangle\rangle$.* ⊠

We write $\textsc{l}\langle\sigma\rangle$ for the σ*-list* sort $\textsc{t}^1\langle\sigma\rangle$. We use the notation $[e_1, e_2, \ldots, e_n | t]$ for $\langle e_1, \langle e_2, \ldots \langle e_n, t \rangle\rangle\rangle$ and we write $[e_1, e_2, \ldots, e_n]$ when $t = \epsilon$.

4 Symbolic tree automata

We introduce an extension of tree automata with an effective encoding of labels by predicates that denote *sets* of labels, rather than individual labels. We study basic properties and algorithms for STAs. These properties and algorithms are instrumental for proving further properties and developing decision procedures for symbolic tree transducers.

Definition 1. A *symbolic tree automaton* (*STA*) A over $\mathcal{P}(\sigma)$ is a quadruple $(Q, Q^0, Q^{\mathrm{a}}, R)$ where Q is a nonempty finite set of *states*, $Q^0 \subseteq Q$ is a set of *leaf states*, $Q^{\mathrm{a}} \subseteq Q$ is a set of *accepting states*, and R is a finite set of *rules* $(q_0, \varphi, q_1, \ldots, q_k)$, where $k \geq 0$, and $q_i \in Q$ for $0 \leq i \leq k$.

Example 7. *We illustrate an STA A that accepts integer-labeled binary trees whose labels are constrained according to* cycle order *traversal [39]:*

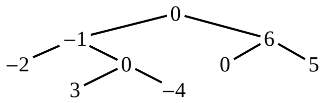
$$A = (\{q_{root}, q_{pre}, q_{in}, q_{post}, q_\epsilon\}, \{q_\epsilon\}, \{q_{root}\}, R),$$

where R consists of the rules

$$\begin{array}{l} (q_{root}, \ell{=}0, q_{pre}, q_{post}),\ (q_{in}, \ell{=}0, q_{post}, q_{pre}), \\ (q_{pre}, \ell{<}0, q_{pre}, q_{in}),\ (q_{post}, \ell{>}0, q_{in}, q_{post}), \\ (q_{pre}, \ell{<}0, q_\epsilon, q_\epsilon),\ (q_{in}, \ell{=}0, q_\epsilon, q_\epsilon),\ (q_{post}, \ell{>}0, q_\epsilon, q_\epsilon) \end{array}$$

Thus, the root has label 0, each "pre"-node has a negative label, each "post"-node has a positive label, and each "in"-node has label 0. There is a single initial state q_ϵ and a single accepting state q_{root}.

For example, the tree



is in $\mathscr{L}(A)$*. Classical tree recognizers that operate over a finite alphabet can of course not recognize* $\mathscr{L}(A)$*.* ⊠

We use A as a subscript to identify a component, unless A is clear from the context. In the following let $A = (Q, Q^0, Q^{\mathrm{a}}, R)$ be a fixed STA. Given a rule $\rho = (p, \varphi, \overline{q}) \in R$, we use the following notations for extracting its *left-hand-side* p, *guard* φ, and *right-hand-side* $\overline{q}$:

$$lhs(\rho) \stackrel{\text{def}}{=} p, \quad grd(\rho) \stackrel{\text{def}}{=} \varphi, \quad rhs(\rho) \stackrel{\text{def}}{=} \overline{q}.$$

Definition 2. The *language of A for* $q \in Q$, denoted by $\mathscr{L}(A, q)$, is the least subset of $\mathscr{U}^{\tau\langle\sigma\rangle}$ such that:

- if $q \in Q^0$ then $\epsilon \in \mathscr{L}(A, q)$;
- if $(q, \varphi, q_1, \ldots, q_k) \in R$, $\mathfrak{a} \in [\![\varphi]\!]$, and, for $1 \leq i \leq k$, $\mathfrak{t}_i \in \mathscr{L}(A, q_i)$ then $\langle \mathfrak{a}, \mathfrak{t}_1, \ldots, \mathfrak{t}_k \rangle \in \mathscr{L}(A, q)$.

The *language of A* is $\mathscr{L}(A) \stackrel{\text{def}}{=} \bigcup_{q \in Q^{\mathrm{a}}} \mathscr{L}(A, q)$.

In the following we consider STAs that accept *binary* trees in order to avoid cumbersome notations. The generalization to arbitrary trees is straightforward. The definition of A admits the following two possible classical views of tree automata:

1. As a *top-down* or *root-to-frontier* STA, a tree $\mathfrak{t}$ is *enabled at state* q if $\mathfrak{t} = \epsilon$ and $q \in Q^0$, or $\mathfrak{t} \neq \epsilon$ and there exists $(q, \varphi, q_1, q_2) \in R$ such that $\mathfrak{t}[0] \in [\![\varphi]\!]$ and $\mathfrak{t}[i]$ is enabled at q_i for $i = 1, 2$.

2. As a *bottom-up* or *frontier-to-root* STA, if $q \in Q^0$ then ϵ is enabled at q, and if $\mathfrak{t}_i$ is enabled at q_i for $i = 1, 2$ and there exists $(q, \varphi, q_1, q_2) \in R$ then $\langle \mathfrak{a}, \mathfrak{t}_1, \mathfrak{t}_2 \rangle$ is enabled at q for all $\mathfrak{a} \in [\![\varphi]\!]$.

In the general case of finite trees, the two views are symmetrical and do not affect the expressiveness of $\mathscr{L}(A)$ that is the set of all (finite) trees that are enabled at some accepting state, unless additional restrictions are placed upon the class of STAs under consideration. For top-down tree automata, Q^{a} is typically assumed to be a singleton set $\{q\}$ and q is referred to as the initial state, while Q^0 is referred to at the set of final states. For bottom-up tree automata the convention is the opposite, i.e., Q^0 is assumed to be a singleton set $\{q\}$ and q is referred to as the initial state, while Q^{a} is referred to at the set of final states. In order to avoid possible confusion we will not use this terminology for STAs. The most important subclasses of STAs are the following.

Definition 3. A is *bottom-up-deterministic* when $|Q^0| = 1$ and, for all $\rho_1, \rho_2 \in R$, if $rhs(\rho_1) = rhs(\rho_2)$ and $grd(\rho_1) \wedge grd(\rho_2) \not\equiv \bot$ then $lhs(\rho_1) = lhs(\rho_2)$.

The following basic property holds for bottom-up-deterministic SFAs. Let

$$Q_A(\mathrm{t}) \overset{\text{def}}{=} \{q \in Q \mid \mathrm{t} \in \mathscr{L}(A, q)\} \quad (\text{for } \mathrm{t} \in \mathscr{U}^{\tau\langle\sigma\rangle}).$$

Proposition 1. If A is bottom-up-deterministic then $|Q_A(\mathrm{t})| \leq 1$ for all t.

Proof. By structural induction over trees. For $\mathrm{t} = \epsilon$ the statement follows from $|Q^0| = 1$. For $\mathrm{t} = \langle \mathfrak{a}, \mathrm{t}_1, \mathrm{t}_2 \rangle$ assume $|Q_A(\mathrm{t}_i)| \leq 1$. If for some i, $Q_A(\mathrm{t}_i) = \emptyset$, then $Q_A(\mathrm{t}) = \emptyset$, otherwise $Q_A(\mathrm{t}_i) = \{q_i\}$ for some $q_i \in Q$ for $i = 1, 2$. Suppose there are two rules (p, φ_1, q_1, q_2) and (q, φ_2, q_1, q_2) in R where $\mathfrak{a} \in [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$. Then $p = q$ since A is bottom-up-deterministic and thus $|Q_A(\mathrm{t})| = |\{p\}| \leq 1$. □

Definition 4. A is *top-down-deterministic* when $|Q^{\mathrm{a}}| = 1$ and, for all $\rho_1, \rho_2 \in R$, if $lhs(\rho_1) = lhs(\rho_2)$ and $grd(\rho_1) \wedge grd(\rho_1) \not\equiv \bot$ then $rhs(\rho_1) = rhs(\rho_2)$.

For $\overline{q} \in Q \times Q$ let $grd_A(\overline{q})$ denote the disjunction of guards of all rules in A whose *right-hand-side* is $\overline{q}$:

$$grd_A(\overline{q}) \overset{\text{def}}{=} \bigvee_{\rho \in R,\ rhs(\rho) = \overline{q}} grd(\rho)$$

Note that, if there is no rule in A whose right-hand-side is $\overline{q}$ then $grd_A(\overline{q})$ is $\bot$ (the empty disjunction). We use the following property.

Definition 5. A is *total* when, for all $\overline{q} \in Q \times Q$, $grd_A(\overline{q}) \equiv \top$.

The following basic property holds for total SFAs.

Proposition 2. If A is total then $|Q_A(\mathrm{t})| \geq 1$ for all t.

Proof. Follows from the definition by structural induction over trees. □

The following construction is used for complementation of total bottom-up-deterministic SFAs.

$$\overline{A} \overset{\text{def}}{=} (Q, Q^{\epsilon}, Q \setminus Q^{\mathrm{a}}, R)$$

Let $elem(\{q\}) \overset{\text{def}}{=} q$.

Proposition 3. If A is total bottom-up-deterministic then $\mathscr{L}(\overline{A}) = \complement(\mathscr{L}(A))$.

Proof. By Propositions 1 and 2, $\mathrm{t} \in \mathscr{L}(\overline{A}) \Leftrightarrow \mathit{elem}(Q_A(\mathrm{t})) \notin Q^{\mathrm{a}} \Leftrightarrow \mathrm{t} \notin \mathscr{L}(A)$. □

Two STAs A and B are *equivalent* when $\mathscr{L}(A) = \mathscr{L}(B)$. Any STA can be effectively transformed into an equivalent total STA by using the following transformation where q_{sink} is a new state.

$$\begin{aligned}
\mathit{Tot}(A) \ \overset{\text{def}}{=}\ & (Q \cup \{q_{sink}\}, Q^0, Q^{\mathrm{a}},\\
& R \cup \{(q_{sink}, \top, \overline{q}) \mid \overline{q} \in Q \times \{q_{sink}\} \cup \{q_{sink}\} \times Q \cup \{(q_{sink}, q_{sink})\}\}\\
& \quad \cup \{(q_{sink}, \neg \mathit{grd}_A(\overline{q}), \overline{q}) \mid \overline{q} \in Q \times Q\})
\end{aligned}$$

Note that, for all $\overline{q} \in Q \times Q$, $\mathit{grd}_{\mathit{Tot}(A)}(\overline{q}) \equiv \mathit{grd}_A(\overline{q}) \vee \neg \mathit{grd}_A(\overline{q}) \equiv \top$, and for all cases when one of the states is the sink state then there is a rule with guard $\top$. Thus, $\mathit{Tot}(A)$ is total. The following properties hold.

Proposition 4. For all $q \in Q$, $\mathscr{L}(A, q) = \mathscr{L}(\mathit{Tot}(A), q)$. If A is bottom-up-deterministic then $\mathit{Tot}(A)$ is bottom-up-deterministic.

Proof. Let $q \in Q$. Clearly $\mathscr{L}(A, q) \subseteq \mathscr{L}(\mathit{Tot}(A), q)$. For the direction $\mathscr{L}(A, q) \supseteq \mathscr{L}(\mathit{Tot}(A), q)$ view A as a bottom-up STA and note that any use of a rule not in R will introduce the sink state that cannot be eliminated. Next, assume that A is bottom-up-deterministic and let $\overline{q} \in Q \times Q$. If $(p, \varphi, \overline{q}) \in R$ then for the new rule $(q_{sink}, \neg(\varphi \vee \cdots), \overline{q})$ it holds that $\neg(\varphi \vee \cdots) \wedge \varphi \equiv \neg\varphi \wedge \neg(\cdots) \wedge \varphi \equiv \bot$. The other cases are immediate. So bottom-up-determinism is preserved in $\mathit{Tot}(A)$. □

It follows from the well-known fact in classical theory of finite tree automata that top-down-deterministic STAs are less expressive than general STAs, i.e., there exists a tree language that is accepted by an STA that is not accepted by any top-down-deterministic STA. However, as in the case of finite tree automata, bottom-up-deterministic STAs have the same expressive power as general STAs. We lift the classical powerset construction to STAs. Let $\mathscr{P}(X)$ denote the powerset of a set X. Note that $\mathscr{P}(\emptyset) = \{\emptyset\}$. The *powerset STA* $\mathscr{P}(A)$ of an STA A is defined as follows.

$$\begin{aligned}
R(\mathbf{q}_1, \mathbf{q}_2) \ &\overset{\text{def}}{=}\ \{\rho \mid \rho \in R, \mathit{rhs}(\rho) \in \mathbf{q}_1 \times \mathbf{q}_2\} \quad (\text{for } (\mathbf{q}_1, \mathbf{q}_2) \in \mathscr{P}(Q) \times \mathscr{P}(Q)\ )\\
\mathit{grd}(S, \overline{\mathbf{q}}) \ &\overset{\text{def}}{=}\ \bigwedge_{\rho \in S} \mathit{grd}(\rho) \wedge \bigwedge_{\rho \in R(\overline{\mathbf{q}}) \setminus S} \neg \mathit{grd}(\rho) \quad (\text{for } S \in \mathscr{P}(R(\overline{\mathbf{q}})))\\
\mathit{lhs}(S) \ &\overset{\text{def}}{=}\ \{\mathit{lhs}(\rho) \mid \rho \in S\} \quad (\text{for } S \in \mathscr{P}(R))\\
\mathscr{P}(A) \ &\overset{\text{def}}{=}\ (\mathscr{P}(Q), \{Q^0\}, \{\mathbf{q} \in \mathscr{P}(Q) \mid \mathbf{q} \cap Q^{\mathrm{a}} \neq \emptyset\},\\
&\qquad \{(\mathit{lhs}(S), \mathit{grd}(S, \overline{\mathbf{q}}), \overline{\mathbf{q}}) \mid \overline{\mathbf{q}} \in \mathscr{P}(Q) \times \mathscr{P}(Q),\ S \in \mathscr{P}(R(\overline{\mathbf{q}}))\})
\end{aligned}$$

Note that the empty conjunuction is $\top$ and thus, when $\overline{\mathbf{q}} = (\emptyset, \_)$ or $\overline{\mathbf{q}} = (\_, \emptyset)$ then $R(\overline{\mathbf{q}}) = \emptyset$, $\mathit{grd}(\emptyset, \overline{\mathbf{q}}) = \top$, and $(\emptyset, \top, \overline{\mathbf{q}}) \in R_{\mathscr{P}(A)}$.

Theorem 1. For all STAs A:

(a) $\mathscr{P}(A)$ is total and bottom-up-deterministic;

(b) for all t, $\{Q_A(\mathrm{t})\} = Q_{\mathscr{P}(A)}(\mathrm{t})$;

(c) $\mathscr{L}(\mathscr{P}(A)) = \mathscr{L}(A)$.

Proof. Proof of (a). To show that $\mathscr{P}(A)$ is total, fix a $\overline{\mathbf{q}} = (\mathbf{q}_1, \mathbf{q}_2) \in \mathscr{P}(Q)^2$. We need to show that $grd_{\mathscr{P}(A)}(\overline{\mathbf{q}}) \equiv \top$. Assume that $\mathbf{q}_1 \neq \emptyset$ and $\mathbf{q}_2 \neq \emptyset$ or else $grd_{\mathscr{P}(A)}(\overline{\mathbf{q}}) \equiv \top$ follows directly. Let $R(\overline{\mathbf{q}}) = \{\rho_i\}_{i\in I}$ and let $\varphi_i = grd(\rho_i)$ for $i \in I$. We have that

$$\begin{aligned} grd_{\mathscr{P}(A)}(\overline{\mathbf{q}}) &\equiv \bigvee\{grd(S, \overline{\mathbf{q}}) \mid S \in \mathscr{P}(R(\overline{\mathbf{q}}))\} \\ &\equiv \bigvee_{J\in\mathscr{P}(I)} (\bigwedge_{i\in J} \varphi_i \wedge \bigwedge_{i\in I\setminus J} \neg\varphi_i) \\ &\equiv \top \end{aligned}$$

where the last equivalence follows from basic properties of Boolean algebras, since *all* possible Boolean combinations of truth assignments of φ_i are included in the disjunction.

To show that $\mathscr{P}(A)$ is bottom-up-deterministic, let $\overline{\mathbf{q}} \in \mathscr{P}(Q) \times \mathscr{P}(Q)$ and let $S_1, S_2 \in \mathscr{P}(R(\overline{\mathbf{q}}))$ such that $S_1 \neq S_2$. It suffices to show that

$$grd(S_1, \overline{\mathbf{q}}) \wedge grd(S_2, \overline{\mathbf{q}}) \equiv \bot$$

which follows from $S_1 \neq S_2$ and the definition of $grd(S, \overline{\mathbf{q}})$ because then there exists a rule $\rho \in R(\overline{\mathbf{q}})$ such that $[\![grd(S_i, \overline{\mathbf{q}})]\!] \subseteq [\![grd(\rho)]\!]$ and $[\![grd(S_j, \overline{\mathbf{q}})]\!] \subseteq [\![\neg grd(\rho)]\!]$ where $\{i, j\} = \{1, 2\}$.

Proof of (b). It follows from (a) and Propositions 1 and 4 that, for all t, $|Q_{\mathscr{P}(A)}(\mathrm{t})| = 1$. We prove (b) by induction over trees. The base case $\mathrm{t} = \epsilon$ follows immediately from the definitions since $\{Q_A(\epsilon)\} = \{Q^0\} = Q_{\mathscr{P}(A)}(\epsilon)$. For the induction case suppose $\mathrm{t} \neq \epsilon$ and as IH assume that, for $i = 1, 2$, $\{Q_A(\mathrm{t}[i])\} = \{\mathbf{q}_i\} = Q_{\mathscr{P}(A)}(\mathrm{t}[i])$. Let $\overline{\mathbf{q}} = (\mathbf{q}_1, \mathbf{q}_2)$. The following statements are equivalent by using the definitions and the IH for the equivalence between 2 and 3. Let $p \in Q$.

1. $p \in Q_A(\mathrm{t})$
2. There exists $(p, \varphi, \overline{q}) \in R$ for some $\overline{q} \in \mathbf{q}_1 \times \mathbf{q}_2$ such that $\mathrm{t}[0] \in [\![\varphi]\!]$.
3. There exists $S \in \mathscr{P}(R(\overline{\mathbf{q}}))$ such that $\mathrm{t}[0] \in [\![grd(S, \overline{\mathbf{q}})]\!]$ and $p \in lhs(S)$.
4. There exists $\mathbf{q} \in \mathscr{P}(Q)$ such that $p \in \mathbf{q}$ and $Q_{\mathscr{P}(A)}(\mathrm{t}) = \{\mathbf{q}\}$.

The equivalence of 1 and 4 for all p implies that $\{Q_A(\mathrm{t})\} = Q_{\mathscr{P}(A)}(\mathrm{t})$, that proves (b). Finally, (c) follows from (b) by definition of $Q^{\mathrm{a}}_{\mathscr{P}(A)}$. □

The above constructions enable us to effectively complement langauges accepted by STAs. For complete closure under Boolean operations we use the following product construction that is a lifting of the standard product of finite tree automata to STAs.

Definition 6. Let $A_i = (Q_i, Q_i^0, Q_i^{\mathfrak{a}}, R_i)$, for $i = 1, 2$, be STAs. The *product* of A_1 and A_2 is the following STA.

$$\begin{array}{rcl} \rho_1 \times \rho_2 & \overset{\text{def}}{=} & ((p_1, p_2), \varphi_1 \wedge \varphi_2, (q_1, q_2), (r_1, r_2)) \quad (\text{for } \rho_i = (p_i, \varphi_i, q_i, r_i) \in R_i) \\ A_1 \times A_2 & \overset{\text{def}}{=} & (Q_1 \times Q_2, Q_1^0 \times Q_2^0, Q_1^{\mathfrak{a}} \times Q_2^{\mathfrak{a}}, \{\rho_1 \times \rho_2 \mid \rho_1 \in R_1, \rho_2 \in R_2\})\} \end{array}$$

The following theorem implies that we can effectively intersect languages accepted by STAs.

Theorem 2. Let $A_i = (Q_i, Q_i^0, Q_i^{\mathfrak{a}}, R_i)$, for $i = 1, 2$, be STAs. Then:
(a) for all $q_1 \in Q_1, q_2 \in Q_2$, $\mathscr{L}(A_1 \times A_2, (q_1, q_2)) = \mathscr{L}(A_1, q_1) \cap \mathscr{L}(A_2, q_2)$;
(b) $\mathscr{L}(A_1 \times A_2) = \mathscr{L}(A_1) \cap \mathscr{L}(A_2)$;
(c) $A_1 \times A_2$ is total if and only if A_1 and A_2 are total;
(d) if A_1 and A_2 are bottom-up-deterministic then so is $A_1 \times A_2$;
(e) if A_1 and A_2 are top-down-deterministic then so is $A_1 \times A_2$.

Proof. We prove (a) by induction over trees. For the base case we have

$$\epsilon \in \mathscr{L}(A_1 \times A_2, (q_1, q_2)) \Leftrightarrow q_1 \in Q_1^0 \wedge q_2 \in Q_2^0 \Leftrightarrow \epsilon \in \mathscr{L}(A_1, q_1) \wedge \epsilon \in \mathscr{L}(A_2, q_2)$$

For the induction case assume $\mathfrak{t} = \langle \mathfrak{a}, \mathfrak{t}_1, \mathfrak{t}_2 \rangle$ and as IH assume that, forall $(q_1, q_2) \in Q_1 \times Q_2$ and $i = 1, 2$,

$$\mathfrak{t}_i \in \mathscr{L}(A_1 \times A_2, (q_1, q_2)) \Leftrightarrow \mathfrak{t}_i \in \mathscr{L}(A_1, q_1) \cap \mathscr{L}(A_2, q_2).$$

The following statements are equivalent for all $(p_1, p_2) \in Q_1 \times Q_2$, where IH is used for equivalence between 2 and 3:

1. $\mathfrak{t} \in \mathscr{L}(A_1 \times A_2, (p_1, p_2))$

2. There exist $(q_1^1, q_2^1), (q_1^2, q_2^2) \in Q_1 \times Q_2$ and φ_1, φ_2 such that

   - $\mathfrak{t}_1 \in \mathscr{L}(A_1 \times A_2, (q_1^1, q_2^1))$ and $\mathfrak{t}_2 \in \mathscr{L}(A_1 \times A_2, (q_1^2, q_2^2))$ and
   - $((p_1, p_2), \varphi_1 \wedge \varphi_2, (q_1^1, q_2^1), (q_1^2, q_2^2)) \in R_{A_1 \times A_2}$ and $\mathfrak{a} \in [\![\varphi_1 \wedge \varphi_2]\!]$.

3. There exist $q_1^1, q_1^2 \in Q_1, q_2^1, q_2^2 \in Q_2$ and φ_1, φ_2 such that

   - $\mathfrak{t}_1 \in \mathscr{L}(A_1, q_1^1)$, $\mathfrak{t}_2 \in \mathscr{L}(A_1, q_1^2)$, $(p_1, \varphi_1, q_1^1, q_1^2) \in R_1$, $\mathfrak{a} \in [\![\varphi_1]\!]$
   - $\mathfrak{t}_1 \in \mathscr{L}(A_2, q_2^1)$, $\mathfrak{t}_2 \in \mathscr{L}(A_2, q_2^2)$, $(p_2, \varphi_2, q_2^1, q_2^2) \in R_2$, $\mathfrak{a} \in [\![\varphi_2]\!]$.

4. $t \in \mathscr{L}(A_1, p_1)$ and $t \in \mathscr{L}(A_2, p_2)$

Equivalence of 1 and 4 proves the induction case and implies (a). (b) follows from (a) by definition of $Q^{\mathrm{a}}_{A_1\times A_2}$.

Proof of (c). Assume first that A_1 and A_2 are total. Let $q = ((q^1_1, q^1_2), (q^2_1, q^2_2)) \in (Q_1 \times Q_2)^2$. We need to show that $grd_{A_1\times A_2}(q) \equiv \top$. Let $q_1 = (q^1_1, q^2_1), q_2 = (q^1_2, q^2_2)$. Assume $grd_{A_1}(q_1) = \bigvee_{i\leq m} \varphi_i$ and $grd_{A_2}(q_2) = \bigvee_{j\leq n} \psi_j$. Then

$$grd_{A_1\times A_2}(q) \equiv \bigvee_{i\leq m, j\leq n} \varphi_i \wedge \psi_j \equiv grd_{A_1}(q_1) \wedge grd_{A_2}(q_2) \equiv \top \wedge \top \equiv \top$$

where the second equivalence holds by de Morgan's laws and the third equivalence holds by using the assumption. For the opposite direction suppose one of A_1 or A_2 is not total, say A_1. Then there exists $q_1 \in Q^2_1$ such that $grd_{A_1}(q_1) \not\equiv \top$ and thus, for any $q_2 \in Q^2_2$, $grd_{A_1}(q_1) \wedge grd_{A_2}(q_2) \not\equiv \top$. It follows that the product is not total.

Proof of (d). Assume that A_1 and A_2 are bottom-up-deterministic. Then clearly $|Q^0_1 \times Q^0_2| = 1$. Consider any two rules $((p, q), \varphi_1 \wedge \psi_1, (p_1, q_1), (p_2, q_2))$ and $((p', q'), \varphi_2 \wedge \psi_2, (p_1, q_1), (p_2, q_2))$ where $\varphi_1 \wedge \psi_1 \wedge \varphi_2 \wedge \psi_2 \not\equiv \bot$ in the product. Then $\varphi_1 \wedge \varphi_2 \not\equiv \bot$ and $\psi_1 \wedge \psi_2 \not\equiv \bot$ and, by using the assumption, it follows that $p = p'$ and $q = q'$.

Proof of (e). Similar to the proof of (d), note that $|Q^{\mathrm{a}}_1 \times Q^{\mathrm{a}}_2| = 1$. □

We use the following definition for constructing the union of tree languages.

Definition 7. Let $A_i = (Q_i, Q^0_i, Q^{\mathrm{a}}_i, R_i)$, for $i = 1, 2$, be STAs. The *sum* of A_1 and A_2 is the following STA. Assume states are renamed so that $Q_1 \cap Q_2 = \emptyset$.

$$A_1 + A_2 \overset{\text{def}}{=} (Q_1 \cup Q_2, Q^0_1 \cup Q^0_2, Q^{\mathrm{a}}_1 \cup Q^{\mathrm{a}}_2, R_1 \cup R_2)$$

Proposition 5. $\mathscr{L}(A_1 + A_2) = \mathscr{L}(A_1) \cup \mathscr{L}(A_2)$.

Proof. Immediate from definitions. □

Let q be some fixed state and define

$$\begin{aligned}
\bot_{STA} &\overset{\text{def}}{=} (\{q\}, \{q\}, \emptyset, \{(q, \top, q, q)\}),\\
\top_{STA} &\overset{\text{def}}{=} (\{q\}, \{q\}, \{q\}, \{(q, \top, q, q)\}),\\
A^c &\overset{\text{def}}{=} \overline{\mathscr{P}(A)}.
\end{aligned}$$

Let $STA(\mathcal{P}(\sigma))$ denote the set of all STAs for some given label theory $\mathcal{P}(\sigma)$. Let $\mathscr{L}(STA(\mathcal{P}(\sigma)))$ denote the corresponding set of tree langauges.

Theorem 3. $(STA(\mathcal{P}(\sigma)), \times, +, {}^c, \bot_{STA}, \top_{STA})$ is an effective Boolean algebra.

Proof. By using Propositions 3 and 5, and Theorems 1 and 2. Note that all STA constructions are effective. □

We say that $\mathcal{P}(\sigma)$ is *decidable* if the problem of deciding $\varphi \equiv \bot$ for $\varphi \in \mathcal{P}(\sigma)$ is decidable. A rule $\rho \in R_A$ such that $grd(\rho) \not\equiv \bot$ is *feasible*. We say A is *clean* if all rules in R_A are feasible.

Theorem 4. If $\mathcal{P}(\sigma)$ is decidable then $STA(\mathcal{P}(\sigma))$ is decidable.

Proof. Assume $\mathcal{P}(\sigma)$ is decidable and let A be an STA over $\mathcal{P}(\sigma)$. We need to show that $\mathscr{L}(A) = \emptyset$ is decidable. First, eliminate infeasible rules from A by using the decision procedure for $\mathcal{P}(\sigma)$. It follows easily from the definitions that $\mathscr{L}(A)$ is unchaged. Next, assume A is clean, view all label predicates as abstract symbols, and use a standard reachability (finite tree automata) algorithm to decide the emptiness. □

An efficient incremental procedure for product of STAs uses DFS (Depth First Search) and starts from the accepting states to build the product while eliminating all the infeasible rules, thus also eliminating all *unreachable* states. Moreover, a backwards reachability algorithm can be applied to eliminate all rules that contain *dead-ends* that are reachable states at which no tree is accepted.

An STA A can also be extended with a set of *epsilon rules* $R^{\varepsilon} \subseteq Q \times Q$, such that if $(p, q) \in R^{\varepsilon}$ then $\mathscr{L}(A, q) \subseteq \mathscr{L}(A, p)$. As with finite tree automata, epsilon rules can be effectively eliminated and do not affect the expressive power of STAs or the results.

5 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 8. A *symbolic tree transducer (STT) over* $(\mathcal{P}(\sigma_1), \mathcal{F}(\sigma_1 \to \sigma_2))$ is a tuple (Q, q^0, R) where Q is a finite set of *states*, $q^0 \in Q$ is the *initial state*, and $R = R^{\epsilon} \cup \bigcup_{k\geq 0} R^k$ is a finite set of *rules*, where a rule in R^{ϵ}, or *ϵ-rule*, is:

- $q \xrightarrow{\epsilon} u$ where $q \in Q$ and $u \in \mathcal{T}(\mathcal{F}(\sigma_1 \to \sigma_2))$ is ground,

and a rule in R^k, or *k-rank-rule*, is:

- $q \xrightarrow{\varphi} u$ where $q \in Q$, $\varphi \in \mathcal{P}(\sigma_1)$ and $u \in \mathcal{T}_{\{q(y_i) \mid q \in Q, 1 \leq i \leq k\}}(\mathcal{F}(\sigma_1 \to \sigma_2))$.

Note that a 0-rank rule does not contain any terms $q(y_i)$, but it may be non-ground (the symbolic labels may depend on the input).

A rule $q \xrightarrow{\varphi} u \in R^k$ corresponds to a conditional transformation of an input tree of rank k from q: e.g., if $k = 2$, $\mathfrak{t} = \langle \mathfrak{a}, \mathfrak{t}_1, \mathfrak{t}_2 \rangle \in \mathcal{U}^{\mathrm{T}\langle\sigma_1\rangle}$ and $\mathfrak{a} \in [\![\varphi]\!]$ then $\mathfrak{t}$ can be transformed to a tree $\mathfrak{u} \in \mathcal{U}^{\mathrm{T}\langle\sigma_2\rangle}$ by applying all the symbolic labels in u to $\mathfrak{a}$ and by replacing, for $i = 1, 2$, each occurrence of $p(y_i)$ in u by some transformation of $\mathfrak{t}_i$ from p.

In the following we assume that $R^k = \emptyset$ for $k \neq 2$, i.e., the definition is over binary trees. Generalization to arbitrary ranks is straightforward. The formal semantics is as follows. Let y be a fixed variable, $Y = \{q(y), q(y_1), q(y_2) \mid q \in Q\}$ and $u \in \mathcal{T}_Y(\mathcal{F}(\sigma_1 \to \sigma_2))$. The state $q(y)$ will be used for referencing the root of an input tree, the states $q(y_1)$ and $q(y_2)$ are used to reference the left, respectively right child of an input tree. Given a set X we write $\mathscr{P}(X)$ for the powerset of X. Then $\lfloor u \rfloor_A$ is defined as the following function from $\mathcal{U}^{\mathrm{T}\langle\sigma_1\rangle}$ to $\mathscr{P}(\mathcal{U}^{\mathrm{T}\langle\sigma_2\rangle})$.

$$\lfloor \epsilon \rfloor_A(\mathfrak{t}) \stackrel{\text{def}}{=} \{\epsilon\} \tag{1}$$

$$\lfloor q(y) \rfloor_A(\epsilon) \stackrel{\text{def}}{=} \{[\![t]\!] \mid q \xrightarrow{\epsilon} t \in R^\epsilon\} \quad (q \in Q) \tag{2}$$

$$\lfloor q(y) \rfloor_A(\mathfrak{t}) \stackrel{\text{def}}{=} \cup\{\lfloor u \rfloor_A(\mathfrak{t}) \mid q \xrightarrow{\varphi} u \in R, \mathfrak{t}[0] \in [\![\varphi]\!]\} \quad (\mathfrak{t} \neq \epsilon, q \in Q) \tag{3}$$

$$\lfloor q(y_i) \rfloor_A(\mathfrak{t}) \stackrel{\text{def}}{=} \lfloor q(y) \rfloor_A(\mathfrak{t}[i]) \quad (i \in \{1, 2\},\ q \in Q) \tag{4}$$

$$\lfloor \langle f, u_1, u_2 \rangle \rfloor_A(\mathfrak{t}) \stackrel{\text{def}}{=} \{\langle [\![f]\!](\mathfrak{t}[0]), \mathfrak{u}_g, \mathfrak{u}_2 \rangle \mid \mathfrak{u}_1 \in \lfloor u_1 \rfloor_A(\mathfrak{t}),\ \mathfrak{u}_2 \in \lfloor u_2 \rfloor_A(\mathfrak{t})\} \tag{5}$$

Informally, the rules for $\lfloor u \rfloor_A(\mathfrak{t})$ create the set of output terms obtained by filling out the states in u with the set of terms produced by transforming $\mathfrak{t}$. When A is clear from the context we often omit the index A.

Example 8. *Consider the STT $A = (\{q_0, q_1, q_2\}, q_0, R)$ where*

$$\begin{aligned} R = \{ \quad & q_0 \xrightarrow{\top} \langle x, q_2(y_1), q_1(y_2) \rangle, \\ & q_1 \xrightarrow{x<0} \langle x - 10, q_1(y_2), \epsilon \rangle,\ q_1 \xrightarrow{\epsilon} \epsilon, \\ & q_2 \xrightarrow{x>0} \langle x + 10, \epsilon, q_2(y_1) \rangle,\ q_2 \xrightarrow{\epsilon} \epsilon\ \} \end{aligned}$$

Let $\mathfrak{t} = \langle -1, \epsilon, \langle -3, \epsilon, \epsilon \rangle \rangle$. Then $\lfloor q_1(y) \rfloor_A(\mathfrak{t}) = \{\langle -11, \langle -13, \epsilon, \epsilon \rangle, \epsilon \rangle\}$ but $\mathfrak{t}$ is not accepted at q_2 because $\mathfrak{t}[0]$ is not positive, so $\lfloor q_2(y) \rfloor_A(\mathfrak{t}) = \emptyset$. ⊠

The semantics of A is the following function from $\mathcal{U}^{\mathrm{T}\langle\sigma_1\rangle}$ to $\mathscr{P}(\mathcal{U}^{\mathrm{T}\langle\sigma_2\rangle})$.

Definition 9. The *transduction* of A is the function $\mathscr{T}_A \stackrel{\text{def}}{=} \lfloor q_A^0 \rfloor_A$.

Example 9. *Consider A and $\mathfrak{t}$ from Example 8. Then $\mathscr{T}_A(\mathfrak{t}) = \{\langle -1, \epsilon, \langle -13, \epsilon, \epsilon \rangle \rangle\}$.* ⊠

Definition 10. The *domain of A for q* is $\mathscr{D}(A, q) \stackrel{\text{def}}{=} \{\mathfrak{t} \mid \lfloor q \rfloor_A(\mathfrak{t}) \neq \emptyset\}$ and the *domain of A* is $\mathscr{D}(A) \stackrel{\text{def}}{=} \mathscr{D}(A, q^0_A)$.

Given a rule $\rho = q \xrightarrow{\gamma} u \in R$, where $\gamma = \epsilon$ or $\gamma \in \mathcal{P}(\sigma_1)$, q is called the *left-hand-side* of ρ, denoted $lhs(\rho)$, γ is called the *guard* of ρ, denoted $grd(\rho)$, and u is called the *right-hand-side* of ρ, denoted $rhs(\rho)$. Two rules $\rho_1, \rho_2 \in R^\epsilon$ *overlap* when $lhs(\rho_1) = lhs(\rho_2)$. Two rules $\rho_1, \rho_2 \in R^k$ *overlap* when $lhs(\rho_1) = lhs(\rho_2)$ and $grd(\rho_1) \wedge grd(\rho_2) \not\equiv \perp$.

We say that a k-rank rule is *linear* if, for $1 \leq i \leq k$, y_i occurs at most once in its right-hand-side.

Definition 11. A is *linear* when the right-hand-sides of all rules in A are linear.

Definition 12. A is *single-valued* when, for all $\mathfrak{t}$, $|\mathscr{T}_A(\mathfrak{t})| \leq 1$.

Definition 13. A is *deterministic* when it contains no two overlapping rules with distinct right-hand-sides.

Example 10. *A classical top-down finite state transducer is over a finite alphabet, say $\{f, g\}$ with f binary and g unary, and contains rewrite rules such as*

$$q(f(y_1, y_2)) \to f(q_1(y_2), g(q_2(y_1))), \quad q_1(g(y)) \to f(q_2(y), q_1(y)),$$

where q, q_1, q_2 are states that are considered as unary functions symbols and y_1, y_2 are variables. Suppose $\mathscr{U}^{\sigma_1} = \{\mathfrak{c}_f, \mathfrak{c}_g\}$. The corresponding STT has the same states and corresponding rules

$$q \xrightarrow{x=\mathfrak{c}_f} \langle \mathfrak{c}_f, q_1(y_2), \langle \mathfrak{c}_g, q_2(y_1)\rangle\rangle, \quad q_1 \xrightarrow{x=\mathfrak{c}_g} \langle \mathfrak{c}_f, q_2(y_1), q_1(y_1)\rangle,$$

in R^2 and R^1 respectively. The second rule is not linear.

In the following examples, all STTs are single-valued and linear. The first example illustrates some simple transformations over INT-labeled binary trees. The point is to illustrate how global STT properties depend on the theory $\mathcal{P}(\sigma_1)$ of labels.

Example 11. *Let the input and the output domains be binary trees with integer-labels. 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 $\langle x, y_1, y_2\rangle$ when $x > 0$ and cuts* *the right subtree y_2 when $x < 0$.*

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

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 sections below, 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 12. *The example illustrates a single-state* INT*-list STT* $\mathit{Encode}^{\textsc{l}\langle\textsc{int}\rangle/\textsc{l}\langle\textsc{int}\rangle}$ *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:*

$$\begin{aligned}
&(\text{'}\mathtt{a}\text{'} \leq x \leq \text{'}\mathtt{z}\text{'}) \vee (\text{'}\mathtt{A}\text{'} \leq x \leq \text{'}\mathtt{Z}\text{'}) \vee\\
&(\text{'}\mathtt{0}\text{'} \leq x \leq \text{'}\mathtt{9}\text{'})\ \vee x = \text{'}\ \ \text{'} \vee x = \text{'}\mathtt{.}\text{'} \vee x = \text{'}\mathtt{,}\text{'} \vee x = \text{'}\mathtt{-}\text{'} \vee x = \text{'}\mathtt{\_}\text{'}
\end{aligned}$$

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

$$\begin{array}{lll}
q \xrightarrow{\epsilon} & \epsilon & \\
q \xrightarrow{\varphi[x]} & [x|q(y_1)] & \\
q \xrightarrow{\neg\varphi[x] \wedge 0 \leq x < 10} & [\text{'}\&\text{'}, \text{'}\#\text{'}, \mathbf{d}_0(x), \text{'}\mathtt{;}\text{'}|q(y_1)] & \\
q \xrightarrow{\neg\varphi[x] \wedge 10^n \leq x < 10^{n+1}} & [\text{'}\&\text{'}, \text{'}\#\text{'}, \mathbf{d}_n(x), \ldots, \mathbf{d}_0(x), \text{'}\mathtt{;}\text{'}|q(y_1)] & (\mathit{for}\ 1 \leq n \leq 4)
\end{array}$$

where

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

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.,* $\mathbf{d}_1$*('&') = '3' and* $\mathbf{d}_0$*('&') = '8') and that* φ*['&'] does not hold, it follows for example that*

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

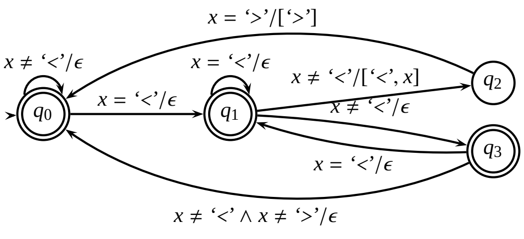
Note that Encode is deterministic because all the guards are mutually exclusive. Thus, Encode is also single-valued. ⊠

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 13. *The example illustrates an* INT*-list STT Extract that extracts from a given input list all subsequences of elements of the form* $[\text{'<'}, x, \text{'>'}]$*, where* $x \neq$ *'<'. For example*

$$\mathscr{T}_{Extract}([\text{'<'}, \text{'<'}, \text{'a'}, \text{'>'}, \text{'<'}, \text{'<'}, \text{'>'}, \text{'<'}, \text{'b'}, \text{'>'}]) = \{[\text{'<'}, \text{'a'}, \text{'>'}, \text{'<'}, \text{'b'}, \text{'>'}]\}$$

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* $q \xrightarrow{\epsilon} \epsilon$ *is depicted by marking q as a final state, and a rule* $q \xrightarrow{\varphi[x]} [t_1, \ldots, t_n | p(y_1)]$*, 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.* ⊠

5.1 Composition of STTs

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

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

Notice that $\circ$ applies first $\mathscr{T}_1$, then $\mathscr{T}_2$, contrary to how $\circ$ is used for standard function composition. (The definition follows the convention used in [17].)

Composition is well defined if the sorts used in the range of $\mathscr{T}_1$ matches the sorts used for the domain of $\mathscr{T}_2$. In [40], we prove the following composition theorem:

Theorem 5. Let A be an STT over $(\mathcal{P}(\sigma_1), \mathcal{F}(\sigma_1 \to \sigma))$, and let B be an STT over $(\mathcal{P}(\sigma), \mathcal{F}(\sigma \to \sigma_2))$.

(a) Then $A \circ B$ is an STT over $(\mathcal{P}(\sigma_1), \mathcal{F}(\sigma_1 \to \sigma_2))$ s.t. $\mathscr{T}_A \circ \mathscr{T}_B = \mathscr{T}_{A \circ B}$.

(b) If A and B are linear then $A \circ B$ is linear.

6 Single-valuedness and equivalence of STTs

Equivalence checking of finite transducers is undecidable when the possible number of outputs for a given input is unbounded [19, 25]. 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 [10, 15], or more generally, for the *finite-valued* case [36] (when there exists k such that, for all t, $|\mathscr{T}_A(t)| \leq k$). Here we investigate the more restricted equivalence problem for *single-valued* STTs as the practically most common case, while the generalization to finite-valued STTs is left as a future research topic.

STTs A and B are *equivalent* if $\mathscr{T}_A = \mathscr{T}_B$. Equivalence of A and B reduces to two separate decision problems:

- *Domain equivalence*: $\mathscr{D}(A) = \mathscr{D}(B)$.

- *Partial equivalence* $A \cong B$: for all $t \in \mathscr{D}(A) \cap \mathscr{D}(B)$, $\mathscr{T}_A(t) = \mathscr{T}_B(t)$.

Note that both problems are independent of each other and together imply equivalence. Partial equivalence in the single-valued case can be reduced to deciding single-valuedness of STTs. Similar to STAs, STTs can be extended to have *epsilon rules*, that are rules of the form $p \to q$ and, for all t, $\lfloor p \rfloor_A(t) \supseteq \lfloor q \rfloor_A(t)$.

Definition 14. Let A and B be STTs. Assume $Q_A \cap Q_B = \emptyset$ and let q be a new state. The *sum* of A and B is the STT

$$A + B \stackrel{\text{def}}{=} (Q_A \cup Q_B \cup \{q\}, q, R_A \cup R_B \cup \{q \rightarrow q_A^0, q \rightarrow q_B^0\}).$$

The following proposition follows directly from definitions.

Proposition 6. For all t, $\mathscr{T}_{A+B}(t) = \mathscr{T}_A(t) \cup \mathscr{T}_B(t)$.

Epsilon rules can be effectively eliminated and in the following we consider only STTs without epsilon rules.

6.1 Domain Equivalence.

Domain equivalence of STTs uses STAs. We use the following definition for $u \in \mathcal{T}_{\{q(y_1),q(y_2)|q\in Q\}}(\mathcal{F}(\sigma_1 \rightarrow \sigma_2))$:

$$St(y_i, u) \stackrel{\text{def}}{=} \{q \mid q(y_i) \text{ occurs in } u\}.$$

Example 14. $St(y_1,\ f(q_1(y_1),\ f(q_2(y_2), q_3(y_1)))) = \{q_1, q_3\}$. ⊠

Definition 15. The *domain automaton* for STT A is the STA $\mathbf{d}(A)$:

$$\mathbf{d}(A) \stackrel{\text{def}}{=} (Q, Q^0, \{\{q_A^0\}\}, R),$$

where, $Q^0, Q \subseteq \mathscr{P}(Q_A)$ and $R \subseteq Q \times \mathcal{P}(\sigma_1) \times Q \times Q$ are least such that

1. $\emptyset \in Q$, $\{q_A^0\} \in Q$, $\emptyset \in Q^0$, $(\emptyset, \top, \emptyset, \emptyset) \in R$,
2. if $\mathbf{q} = \{q_1, \ldots, q_n\} \in Q$ then
   (a) if, for all i, $1 \leq i \leq n$, there is a rule $q_i \xrightarrow{\varphi_i} u_i$ in R_A^2 then,
      i. let, for $j = 1, 2$, $\mathbf{p}_j = \bigcup_{i=1}^n St(y_j, u_i)$,
      ii. $(\mathbf{q}, \bigwedge_{i=1}^n \varphi_i, \mathbf{p}_1, \mathbf{p}_2) \in R$, $\mathbf{p}_1, \mathbf{p}_2 \in Q$,
   (b) if, for all i, $1 \leq i \leq n$, there is a rule $q_i \xrightarrow{\epsilon} e_i$ in R_A^ϵ, then $\mathbf{q} \in Q^0$.

Note that the state $\emptyset \in Q_{\mathbf{d}(A)}$ is used when an input subtree y_j does not occur in the right-hand-side of a rule in R_A, thus any input-subtree is allowed, i.e., $\mathbf{p}_j = \emptyset$ and $\mathscr{L}(\mathbf{d}(A), \mathbf{p}_j) = \mathscr{U}^{\mathrm{T}\langle\sigma_1\rangle}$. Note also that all states in $Q_{\mathbf{d}(A)}$ are singletons when A is linear, it is only when a nonlinear rule occurs when non-singleton states are introduced into $Q_{\mathbf{d}(A)}$ in step 2(a)ii. Moreover, $\mathbf{d}(A)$ can be implemented using DFS and where step 2(a)ii is performed only if the conjuction of the guards is feasible, thus guaranteeing that the resulting STA is clean and unreachable states are pruned away.

Proposition 7. Let A be an STT. Then

(a) For all $\mathbf{q} \in Q_{\mathbf{d}(A)}$, $\mathscr{L}(\mathbf{d}(A), \mathbf{q}) = \begin{cases} \bigcap_{q\in\mathbf{q}} \mathscr{D}(A, q), & \text{if } \mathbf{q} \neq \emptyset; \\ \mathscr{U}^{\mathrm{T}\langle\sigma_1\rangle}, & \text{if } \mathbf{q} = \emptyset. \end{cases}$

(b) $\mathscr{L}(\mathbf{d}(A)) = \mathscr{D}(A)$.

Proof. We prove (a). The case when A is linear follows directly from the definitions, then the rules in $\mathbf{d}(A)$ correspond to the reachable rules of A and where the output terms are omitted. Suppose there is a nonlinear rule $q \xrightarrow{\varphi} u$, say $q_1(y_1)$ and $q_2(y_1)$ occur in u, where $q_1 \neq q_2$, and an input tree $\langle \mathfrak{a}, \mathfrak{t}_1, \mathfrak{t}_2\rangle$ is transformed at q where $\mathfrak{a} \in [\![\varphi]\!]$. Then $\mathfrak{t}_1$ is simultaneously transformed from q_1 and q_2 and must therefore be enabled from both states at the same time, i.e., $\lfloor q_1 \rfloor_A(\mathfrak{t}_1) \neq \emptyset$ and $\lfloor q_2 \rfloor_A(\mathfrak{t}_1) \neq \emptyset$. This corresponds to $\mathfrak{t}_1$ being enabled at state $\{q_1, q_2\}$ in $\mathbf{d}(A)$. Formally, (a) follows by induction over trees. Statement (b) follows from (a) by choosing $\mathbf{q} = \{q^0_A\}$. □

Proposition 8. Domain equivalence of STTs is decidable if $\mathcal{P}(\sigma_1)$ is decidable.

Proof. By using Theorem 4 and Proposition 7. □

Note that the size of $\mathbf{d}(A)$ is at most singly expontential in the size of A and the size of guards grows at most linearly in the size of A.

For many practical considerations, domain equivalence of A and B is often not as relevant as partial equivalence because the transductions of A and B are known to correspond to *total* functions from $\mathscr{U}^{\mathrm{T}\langle\sigma_1\rangle}$ to $\mathscr{U}^{\mathrm{T}\langle\sigma_2\rangle}$, i.e., $\mathscr{D}(A) = \mathscr{D}(B) = \mathscr{U}^{\mathrm{T}\langle\sigma_1\rangle}$, reflecting a *robustness* assumption of the underlying programs.

6.2 Single-valuedness.

We design an algorithm for deciding single-valuedness of STTs. Partial equivalence of single-valued STTs reduces effectively to single-valuedness of STTs. For this reduction we make use of the following construction.

Definition 16. Let A be an STT and D and STA. Assume $F_D = \{q^0_D\}$. The *domain restriction* of A with respect to D is an STT $A{\restriction}D = (Q, q^0, R)$ with $Q = \{\langle p, q\rangle \mid p \in Q_A, q \in Q_D\}$ as a new set of states, $q^0 = \langle q^0_A, q^0_D\rangle$, and

$$R = \{\langle p, q\rangle \xrightarrow{\varphi\wedge\psi} u \otimes (q_1, q_2) \mid \langle p, q\rangle \in Q,\ p \xrightarrow{\varphi} u \in R_A,\ (q, \psi, q_1, q_2) \in R_D\}$$

where $u \otimes (q_1, q_2)$ denotes the term obtained from u by replacing all occurences of $r(y_1)$ (resp. $r(y_2)$) for $r \in Q_A$ with $\langle r, q_1\rangle(y_1)$ (resp. $\langle r, q_2\rangle(y_2)$).

Example 15. $f(f(r_1(y_1), r_2(y_2)), f(r_3(y_1), r_1(y_2))) \otimes (q_1, q_2)$
$= f(f(\langle r_1, q_1\rangle(y_1), \langle r_2, q_2\rangle(y_2)), f(\langle r_3, q_1\rangle(y_1), \langle r_1, q_2\rangle(y_2)))$. ⊠

Similar to product and domain automaton constructions, domain retriction can be implemented most efficiently using DFS that avoids unreachable states and keeps the resulting STT clean. The following property follows from the definition.

Proposition 9. Let A be an STT and D and STA. Then
(a) $\mathscr{D}(A{\restriction}D) = \mathscr{D}(A) \cap \mathscr{L}(D)$;
(b) for all $t \in \mathscr{D}(A{\restriction}D)$, $\mathscr{T}_{A{\restriction}D}(t) = \mathscr{T}_A(t)$.

Proof. By induction over trees. □

We use the following proposition to reduce partial equivalence of single-valued STTs to single-valuedness of an STT.

Proposition 10. Let A and B be single-valued STTs. Then
$A \cong B$ iff $(A + B){\restriction}(\mathbf{d}(A) \times \mathbf{d}(B))$ is single-valued.

Proof. Assume that A and B are single-valued STTs. The following statements are equivalent by making use of the properties proved above.

1. $A \cong B$
2. For $t \in \mathscr{D}(A) \cap \mathscr{D}(B)$ $\mathscr{T}_A(t) = \mathscr{T}_B(t)$
3. For $t \in \mathscr{L}(\mathbf{d}(A) \times \mathbf{d}(B))$ $\mathscr{T}_A(t) = \mathscr{T}_B(t)$
4. For $t \in \mathscr{L}(\mathbf{d}(A) \times \mathbf{d}(B))$ $|\mathscr{T}_{A+B}(t)| = 1$
5. $(A + B){\restriction}(\mathbf{d}(A) \times \mathbf{d}(B))$ is single-valued.

The single-valuedness assumption is used for equivalence of 3 and 4. □

We now develop an algorithm for deciding single-valuedness of STTs. Let $A = (Q, q^0, R)$ be a fixed STT. In the following we assume that $\mathcal{P}(\sigma_1)$ is decidable. Above, we did not make any assumptions about the symbolic labels. In the following we need to strengthen the decidability assumption to allow us to effectively reason about labels. The following properties are assumed to be decidable:

- For $f, g \in \mathcal{F}(\sigma_1 \to \sigma_2)$ and $\varphi \in \mathcal{P}(\sigma_1)$, f and g are *equivalent for* φ:

$$f \equiv_\varphi g \quad \overset{\text{def}}{=} \quad \forall\, \mathfrak{a} \in [\![\varphi]\!]\, ([\![f]\!](\mathfrak{a}) = [\![g]\!](\mathfrak{a})).$$

- For $f \in \mathcal{F}(\sigma_1 \to \sigma_2)$ and $\varphi \in \mathcal{P}(\sigma_1)$, f is *constant for* φ:

$$\mathit{Const}_\varphi(f) \quad \overset{\text{def}}{=} \quad \forall\, \mathfrak{a}, \mathfrak{b} \in [\![\varphi]\!]\, ([\![f]\!](\mathfrak{a}) = [\![f]\!](\mathfrak{b})).$$

- If $Const_{\varphi}(f)$ then find a witness $\mathfrak{b}$, such that, for $\mathfrak{a} \in [\![\varphi]\!]$, $[\![f]\!](\mathfrak{a}) = \mathfrak{b}$.

Example 16. *Suppose for example that $\mathcal{F}(\textsc{int} \to \textsc{int})$ is the set of quantifier free linear arithmetic terms with one variable x and $\mathcal{P}(\textsc{int})$ is the set of quantifier free linear arithmetic formulas with one variable x. Then $Const_{\varphi}(f)$ holds iff the following quantifier free linear arithmetic formula (with two variables) is* unsatisfiable*: $\varphi(x_1) \wedge \varphi(x_2) \wedge f(x_1) \neq f(x_2)$. Note also that $f \equiv_{\varphi} g$ holds iff the formula $\varphi(x) \wedge f(x) \neq g(x)$ is unsatisfiable, that formula is in $\mathcal{P}(\textsc{int})$ since a single variable is sufficient.* ⊠

Since A is clear from the context, we write $\mathscr{D}(q)$ for $\mathscr{D}(A, q)$ and

$$\mathscr{D}(\mathbf{q}) \overset{\text{def}}{=} \bigcap_{q \in \mathbf{q}} \mathscr{D}(q) \quad \text{(for nonempty } \mathbf{q} \subseteq Q\text{)},$$

i.e., $\mathscr{D}(\mathbf{q})$ is the set of trees that are *simultaneously* enabled at all states in $\mathbf{q}$. Let y be a fixed variable of sort $\textsc{t}\langle\sigma_1\rangle$. For a term $u \in \mathcal{T}_{\{q(y) \mid q \in Q\}}(\mathcal{F}(\sigma_1 \to \sigma_2))$ we let

$$\mathscr{D}(u) \overset{\text{def}}{=} \mathscr{D}(St(y, u)).$$

i.e., $\mathscr{D}(u)$ is the set of trees that are *simultaneously* enabled at all states q such that $q(y)$ occurs in u.

Definition 17. Given a nonempty subset $\mathbf{q}$ of Q and $q \in \mathbf{q}$ we say that *q is constant for* $\mathbf{q}$ when $|\bigcup\{\lfloor q(y)\rfloor_A(\mathrm{t}) \mid \mathrm{t} \in \mathscr{D}(\mathbf{q})\}| = 1$.

Example 17. *The state q_1 is constant for $\{q_0, q_1\}$ in $Swap_1$.*

$$Swap_1 \quad = \quad \left(\{q_0, q_1\}, q_0, \left\{ \begin{array}{l} q_0 \xrightarrow{\epsilon} \epsilon, q_0 \xrightarrow{x \neq 0} \langle x, q_1(y_2), q_0(y_1)\rangle, \\ q_0 \xrightarrow{x=0} \langle x, q_0(y_1), q_1(y_2)\rangle, q_1 \xrightarrow{\top} \langle 0, \epsilon, \langle 1, \epsilon, \epsilon\rangle\rangle \end{array} \right\} \right)$$

⊠

In other words, q is constant for $\mathbf{q}$, if independent from the input tree in $\mathscr{D}(\mathbf{q})$, the resulting transformation from q is some fixed output tree. It also follows that $\mathscr{D}(\mathbf{q})$ is non-empty. We can effectively decide if q is constat for $\mathbf{q}$ and construct a concrete output tree, by using a DFS procedure. We omit the details of this procedure but note that the decision procedure for $Const_{\varphi}(f)$ is used.

The following definition is used as a key notion in the single-valuedness algorithm.

Definition 18. Let u and v be terms, and $\mathbf{q}$ a nonempty subset of Q such that all states in u and v occur in $\mathbf{q}$. Then *u is 1-equal to v for* $\mathbf{q}$, is defined as

$$u \overset{1}{=}_{\mathbf{q}} v \quad \overset{\text{def}}{=} \quad \forall \mathrm{t}(\mathrm{t} \in \mathscr{D}(\mathbf{q}) \Rightarrow |\lfloor u\rfloor(\mathrm{t}) \cup \lfloor v\rfloor(\mathrm{t})| = 1)$$

Note that 1-equality is not reflexive because $\lfloor u \rfloor(t)$ may contain more than one element; 1-equality is a generalization of single-valuedness of STTs because $q^0(y) \overset{1}{=}_{\{q^0\}} q^0(y)$ holds precisely when A is single-valued.

We will in the following define a notion of a most general 1-unifier (1-mgu) and then develop an algorithm that finds the 1-mgu when they exist. A subterm $q(y_i)$ is treated as a *variable* in the following definitions. That is, we treat $q_1(y_1)$, $q_2(y_1)$, $q_1(y_2)$, $q_2(y_2)$ as four different variables. This convention allows us to form substitutions that map variables of the form $q(y_i)$ to output terms. The variable $q(y_i)$ is called a *y_i-output variable*.

Definition 19. The substitution θ is a 1-unifier for u, v and ψ if $u\theta$ and $v\theta$ have the same tree structure, and the symbolic labels are equal modulo the constraint ψ. In other words, for each path π to a symbolic label $v\theta|_\pi \equiv_\psi u\theta|_\pi$.

Definition 20. The substitution θ is a most general 1-unifier (1-mgu) for u, v and ψ when θ is a 1-unifier and also every other 1-unifier θ' is an instance of θ.

Unlike standard unification of first-order terms, the unification problem for STTs is not unitary. There can be many different incompatible unifiers without a most general unifier. The following algorithm succeeds only when there is a most general unifier.

Definition 21. Two tree terms u and v *1-unify for ψ* with the substitution θ when the following conditions hold. Let $\mathbf{q}_i = St(y_i, u) \cup St(y_i, v)$ for $i = 1, 2$. Initialize θ to the empty substitution [].

1. u and v unify in the usual sense with unifier θ, if all symbolic labels are assumed identical.

2. For all positions π such that $u|_\pi$ and $v|_\pi$ are symbolic labels, $u|_\pi \equiv_\psi v|_\pi$.

3. For all positions π in u and v such that $u|_\pi$ is a y_1-output variable $p(y_1)$ then (symmetrically for y_2-output variables):

   (a) If $v|_\pi$ contains a symbolic label that is not constant for ψ then **fail**, else assume that all symbolic labels in $v|_\pi$ are constant by replacing each symbolic label f in $v|_\pi$ with $[\![f]\!](\mathfrak{a})$ for some $\mathfrak{a} \in [\![\psi]\!]$

   (b) If $v|_\pi$ contains a y_2-output variable $q(y_2)$, then if q is not constant for $\mathbf{q}_2$ then **fail**, else assume that $v|_\pi$ contains no y_2-output variables by replacing them with the corresponding fixed output trees.

   (c) If $v|_\pi$ contains no y_1-output variables, then $v|_\pi$ is ground and p must be constant for $\mathbf{q}_1$ and the value must be equal to $[\![v|_\pi]\!]$.

(d) If $p(y_1)$ occurs properly under a function application in $v|_\pi$, then **fail**, else add to θ the substitution $p(y_1) \mapsto v|_\pi$. Apply the substitution θ to u and v and continue.

When u and v 1-unify for ψ, the mapping θ is called a *1-unifier of u and v*. The algorithm ensures that it maps from y_i-output variables to terms that have constant symbolic labels and contain at least one y_i-output variable and no y_j-output variables, where $\{i, j\} = \{1, 2\}$.

Example 18. *The terms $\langle 2x, q_1(y_1), q_2(y_2)\rangle$ and $\langle 3x, q_1(y_1), q_2(y_2)\rangle$ 1-unify for $\psi \equiv x = 0$. They don't 1-unify for $\psi \equiv x > 0$. The terms $\langle 2x, q_1(y_1), q_2(y_2)\rangle$ and $\langle 3x, q_1(y_1), \langle 1, \epsilon, \epsilon\rangle\rangle$ 1-unify for $\psi \equiv x = 0$ when the transition for q_2 derives a term that is equal to $\langle 1, \epsilon, \epsilon\rangle$ under ψ. For example: $q_2 \xrightarrow{\top} \langle x + 1, \epsilon, \epsilon\rangle$.*

The terms $\langle \max(0, x), q_1(y_1), q_2(y_2)\rangle$ and $\langle 0, \langle x, q_2(y_1), q_3(y_2)\rangle, q_3(y_1)\rangle$ 1-unify for $\psi \equiv x \leq 0$ when q_3 is given by $q_3 \xrightarrow{\top} \langle 0, \epsilon, \epsilon\rangle$. The 1-unifier is the substitution $[q_1(y_1) \mapsto \langle x, q_2(y_1), \langle 0, \epsilon, \epsilon\rangle\rangle]$. It would also 1-unify even if q_3 does not derive the constant term, but as long as q_3 is constant for $\{q_2, q_3\}$.

The terms $f(x, q_1(y_1), g(q_2(y_2)))$ and $f(x, g(q_2(y_1)), q_1(y_2))$ 1-unify with the substitution $[q_1(y_1) \mapsto g(q_2(y_1)), q_1(y_2) \mapsto g(q_2(y_2))]$.

The terms $f(x, q_1(y_1), g(q_1(y_2)))$ and $f(x, g(q_2(y_1)), q_2(y_2))$ 1-unify with the substitution $[q_1(y_1) \mapsto g(q_2(y_1)), q_2(y_2) \mapsto g(q_1(y_2))]$.

Consider the STT $A = (\{p_0, p_1, p_2\}, p_0, R)$ where

$$\begin{aligned} R = \{ \quad & p_0 \xrightarrow{\top} \langle x, p_1(y_1), p_2(y_1)\rangle, \\ & p_1 \xrightarrow{x \geq 0} \langle x, p_1(y_1), p_1(y_2)\rangle,\; p_1 \xrightarrow{\epsilon} \epsilon, \\ & p_2 \xrightarrow{x \leq 0} \langle x, p_2(y_1), p_2(y_2)\rangle,\; p_2 \xrightarrow{\epsilon} \epsilon \;\} \end{aligned}$$

Let u be $\langle x, p_1(y_1), p_2(y_1)\rangle$ and v be $\langle -x, p_2(y_1), p_2(y_1)\rangle$. Then u and v 1-unify for $\psi : x = 0$. In the algorithm for 1-unification, we see that the set $\mathbf{q}_1$ is $\{p_1, p_2\}$ so we are only considering trees that are accepted by following both p_1 and p_2. The constraint $x = 0$ implies that the outputs generated from p_1 and p_2 will be identical. ⊠

The single-valuedness algorithm is a constraint saturation procedure over 1-equalities. The set of constraints is represented as a map C from $Q \times \mathscr{P}(Q)$ to terms in $\mathcal{T}_{\{q(y)|q\in Q\}}(\mathcal{F}(\sigma_1 \to \sigma_2))$ with constant symbolic labels. Initially

$$C = \{(q^0, \{q^0\}) \mapsto q^0(y)\}.$$

A constraint $((p, \mathbf{q}) \mapsto u) \in C$ stands for the assertion $\mathbf{q} = \mathscr{D}(u)$ and $p(y) \overset{1}{=} u$. The intuition is as follows: The state p is the current state we check for single-valuedness and we check it over a specialization over a partial output t with the states $\mathbf{q}$. All these states should produce the same output modulo the guards on transitions over $\mathbf{q}$. We will ensure that all states $\mathbf{q}$ in t are applied to either only y (the variable used for the root symbol) or only y_1 (variable for the left sub-term) or only y_2 (variable for the right sub-term). There is a frontier $F \subseteq Q \times \mathscr{P}(Q)$ of unexplored *state combinations*. Initially $F = \{(q^0, \{q^0\})\}$. The general idea is that constraints are added to C for unexplored state combinations by exhaustively considering all rules from the states in the state combination. When a conflict of 1-equalities arises, it follows that A is not single-valued. If no conflicts arise, C gets saturated (a fixpoint is reached) and it follows that A is single-valued. The detailed description is as follows.

1. Choose $(p, \mathbf{q}) \in F$ and remove $(p, \mathbf{q})$ from F. Let $\mathbf{q} = \{q_1, \ldots, q_k\}$ and $t = C(p, \mathbf{q})$.

2. For each combination of rules $p \xrightarrow{\epsilon} u, q_1 \xrightarrow{\epsilon} u_1, \ldots, q_k \xrightarrow{\epsilon} u_k \in R^\epsilon$:

   - Let $\theta = \{q_1(y_i) \mapsto u_1, \ldots, q_k(y_i) \mapsto u_k\}$, where y_i is either y, y_1 or y_2.
   - If $[\![u]\!] \neq [\![t\theta]\!]$ then **fail**.

3. For each combination of rules $p \xrightarrow{\varphi} u, q_1 \xrightarrow{\varphi_1} u_1, \ldots, q_k \xrightarrow{\varphi_k} u_k \in R^2$ such that $\psi = \varphi \wedge \varphi_1 \wedge \cdots \wedge \varphi_k \not\equiv \bot$:

   - Let $v = t\{q_1(y_i) \mapsto u_1, \ldots, q_k(y_i) \mapsto u_k\}$, where y_i is either y, y_1 or y_2.
   - If u and v do not 1-unify for ψ then **fail**.
   - Let θ be a 1-mgu for u and v. Notice that each mapping in θ will either use y_1 or y_2, but not both.
   - Call Insert(θ, ψ), which is defined recursively below:

The procedure Insert(θ, ψ) is defined:

1. For each $(p(y_i) \mapsto w(y_i)) \in \theta$ do:

   - Let $\mathbf{q}$ be $St(y_1, w)$.
   - If C contains no constraint for $(p, \mathbf{q})$ then add the constraint $(p, \mathbf{q}) \mapsto w$ to C; add $(p, \mathbf{q})$ to the frontier F.
   - Otherwise, C contains a constraint $(p, \mathbf{q}) \mapsto w'$. If w does not 1-unify with w' for ψ then **fail** else let θ' be the 1-mgu of w and w', replace $(p, \mathbf{q}) \mapsto w'$ by $(p, \mathbf{q}) \mapsto w\theta'$, and call Insert$(\theta', \psi)$.

The algorithm is a constructive proof of the following Proposition.

Proposition 11. Single-valuedness of STTs is decidable if $\mathcal{P}(\sigma_1)$ is decidable.

Correctness of the algorithm is established by checking that it implements an exhaustive case analysis for checking single-valuedness. The algorithm, and in particular the procedure Insert terminates as one should expect: Let us first notice that the algorithm maintains an invariant $((p, \mathbf{q}) \mapsto t(y_i)) \in C$ implies that $St(y_i, t) = \mathbf{q}$ and that t only contains either y_1, y_2 or y variables. Now notice that a 1-unifier substitution produces terms $t(y_i)$ with fewer states than the the states from w and w'. In other words we have the descending measure: $St(y_i, t(y_i)) \subset St(y_i, w) = St(y_i, w') = \mathbf{q}$.

The resulting algorithm works for single-valued symbolic tree transducers. This is more general than the algorithm that we developed in [40], which checked 1-equality of single-valued *linear* symbolic tree transducers. That algorithm visits at most $|Q_A| \times |Q_B|$ states. The current algorithm for *non-linear* symbolic tree transducers visits at most $|(Q_A \cup Q_B) \times \mathscr{P}(Q_A \cup Q_B)|$ states. It is an open problem whether this is the tightest upper bound.

Propositions 8 and 11 imply:

Corollary 1. *If $\mathcal{P}(\sigma_1)$ is decidable then the question whether two STTs are single valued and equivalent is decidable.*

The algorithm can be implemented using any SMT solver or constraint solver as an oracle that supports satisfiability checking and model generation (that is needed above). In our implementations we have used the SMT solver Z3 [8].

The equivalence algorithm for two symbolic transducers checks the logical statement $\forall x \,.\, A(x) \equiv B(x)$ for validity. Dually, it checks satisfiability of a formula of the form $A(x) \not\equiv B(x)$. Can we also check satisfiability of formulas of the form $A(x) \equiv B(x)$? The answer turns out to be *no* as the following simple theorem establishes.

Theorem 6. Satisfiability of equality is undecidable for finite alphabet tree transducers.

Proof. Recall the PCP (Post's Correspondence Problem). Given $v_1, \ldots, v_k$ and $w_1, \ldots, w_k$ where $v_i, w_i \in \Sigma^*$ (for some output alphabet Σ). The question, does there exist $i_1, \ldots, i_m$, $m > 0$, such that $v_{i_1} v_{i_2} \ldots v_{i_m} = w_{i_1} w_{i_2} \ldots w_{i_m}$, is known to be undecidable.

We use the following encoding into PCP: Let Σ' be the input alphabet $\{1, \ldots, k\}$ and let A have states q_0 and q_1 (q_1 is final), and transitions $\{q_0 \xrightarrow{i/v_i} q_1, q_1 \xrightarrow{i/v_i} q_1\}$, for each i. Likewise, let B have states p_0 and p_1 (p_1 is final), and transitions $\{p_0 \xrightarrow{i/w_i} p_1, p_1 \xrightarrow{i/w_i} p_1\}$, for each i. Then $\exists x(A(x) \equiv B(x))$ iff the given PCP instance has a solution Note that both A and B are deterministic. □

6.3 Checking Non-equivalence Symbolically

Let us recall from [40] also how we can formulate a simple semi-decision procedure for checking non-equivalence of symbolic tree transducers. It does not assume single-valuedness. Here, we formulate a version that applies to non-linear symbolic transducers. Given a transducer A, that does not contain ϵ loops, we can encode a predicate $Acc_A(q^0_A, t, s, n)$, 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

$$\exists t, s, n\ .\ \left(\begin{array}{ll} & (Acc_A(q^0_A, t, s, n) \wedge \neg Acc_B(q^0_B, t, s, n)) \\ \vee & (\neg Acc_A(q^0_A, t, s, n) \wedge Acc_B(q^0_B, t, s, n)) \end{array}\right).$$

The definition is given by:

$$\begin{array}{rcl} Acc_A(q, f(t_0, t_1, t_2), s, n) & \equiv & \bigvee_{\tau \in R_A} \left(\begin{array}{l} n > 0 \wedge\ \varphi[t_0] \wedge \\ s = u[t_0, \ell_1(s), \ldots, \ell_k(s)] \wedge \\ \bigwedge_{i=1}^{k} Acc_A(q_i, t_{j_i}, \ell_i(s), n-1) \end{array}\right) \\ Acc_A(q, \epsilon, \epsilon, n) & \equiv & true \end{array}$$

where, as usual, τ is of the form $q(f(x, y_1, y_2)) \xrightarrow{\varphi} u[x, q_1(y_{j_1}), q_2(y_{j_2}), \ldots q_k(y_{j_k})]$, j_i is either 1 or 2, and $\ell_i(s)$ 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 linear 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.

7 Conclusions

We investigated the classes of Symbolic Tree Automata and more generally Symbolic Tree Transducers as a generalization of tree automata and transducers over finite alphabets. We established that symbolic tree automata form an effective Boolean algebra, provided the underlying symbolic domain is also effective. As a side-effect it is possible to define tree automata over alphabets of tree-automata ad infinum. We also established, by providing an algorithm, that equivalence of single-valued symbolic tree transducers is decidable. This settled a question left open in [40].

A generalization of our equivalence checking algorithm to the finite-valued case is not expected to be straightforward, because the corresponding generalization of decidability of equivalence for finite-valued tree transducers [36] uses results from combinatorics and is mathematically challenging.

We are using symbolic automata and symbolic transducers in applications related to web-sanitizers and test case generation. There are likely many other applications of symbolic analysis of automata and transducers; and our experience so far indicates that coupling the analysis of the symbolic automata and transducers with SMT solvers offers a compelling combination. There are also many interesting problems to work on in this area. For example, ongoing work includes adding *registers* to symbolic string transducers. Registers allow storing characters from the input for an indefinite number of transitions. The resulting automata and transducers are strictly more general, and without further restrictions, are Turing complete. So a challenge is identifying such extensions that are both useful and admit practical analysis.

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] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] 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.

[4] B. S. Baker. Composition of top-down and bottom-up tree transductions. *Inform. and Control*, 41:186–213, 1979.

[5] C. Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.

[6] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[7] B. Courcelle and P. Franchi-Zannettacchi. Attribute grammars and recursive program schemes. *Theoretical Computer Science*, 17:163–191, 1982.

[8] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS'08*, LNCS. Springer, 2008.

[9] J. Engelfriet. Bottom-up and top-down tree transformations – a comparison. *Math. Systems Theory*, 9:198–231, 1975.

[10] 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.

[11] J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and mso definable tree translations. *Information and Computation*, 154:34–91, 1999.

[12] J. Engelfriet and S. Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39:2003, 2003.

[13] 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.

[14] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. and Syst. Sci.*, 31:71–146, 1985.

[15] Z. Esik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.

[16] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[17] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. EATCS. Springer, 1998.

[18] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[19] T. Griffiths. The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines. *J. ACM*, 15:409–413, 1968.

[20] Y. Gurevich, M. Veanes, and C. Wallace. Can abstract state machines be useful in language theory? *Theor. Comput. Sci.*, 376(1):17–29, 2007.

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

[22] 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.

[23] P. Hooimeijer and M. Veanes. An evaluation of automata algorithms for string analysis. In R. Jhala and D. Schmidt, editors, *VMCAI 2011*, volume 6538 of *LNCS*, pages 248–262. Springer, 2011.

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

[25] 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.

[26] 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.

[27] N. Klarlund. Mona & Fido: The Logic-Automaton Connection in Practice. In *CSL*, pages 311–326, 1997.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] G. V. Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4:263–286, 2001.

[33] 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.

[34] T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89(3):141–149, 2004.

[35] W. C. Rounds. Context-free grammars on trees. In *Proc. ACM Symp. on Theory of Comput.*, pages 143–148. ACM, 1969.

[36] H. Seidl. Equivalence of finite-valued tree transducers is decidable. *Math. Systems Theory*, 27:285–346, 1994.

[37] J. W. Thatcher. Generalized sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

[38] N. Tillmann and W. Schulte. Parameterized unit tests. In *ESEC/FSE'05*, pages 253–262. ACM, 2005.

[39] M. Veanes and J. Barklund. On the number of edges in cycletrees. *Information Processing Letters*, 57:225–229, 1996.

[40] M. Veanes and N. Bjørner. Symbolic tree transducers. In E. M. Clarke, I. Virbitskaite, and A. Voronkov, editors, *Perspectives of System Informatics (PSI'11)*, LNCS. Springer, 2011.

[41] 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.

[42] M. Veanes, N. Tillmann, and J. de Halleux. Qex: Symbolic SQL query explorer. In *LPAR-16*, LNAI. Springer, 2010.

[43] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1997.