Equivalence of Symbolic Tree Transducers

Vincent Hugot[2,3], Adrien Boiret[2,3], and Joachim Niehren[1,3]

[1] Inria, Lille, France
[2] University of Lille, France
[3] Links project (Inria & Cristal Lab - Umr Cnrs 9189)

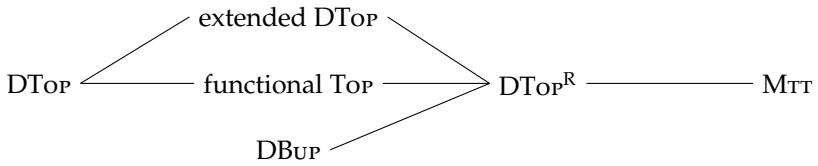
Abstract. Symbolic tree transducers are programs that transform data trees with an infinite signature. In this paper, we show that the equivalence problem of deterministic symbolic top-down tree transducers (DTop) can be reduced to that of classical DTop. As a consequence the equivalence of two symbolic DTop can be decided in NExpTime, when assuming that all operations related to the processing of data values are in PTime. This result can be extended to symbolic DTop with lookahead and thus to deterministic symbolic bottom-up tree transducers.

This is an extended version of the paper published at DLT 2017.

1 Introduction

Data trees are widely used in various domains of computer science. They represent programs in compiler construction or program analysis, syntactic sentence structure in computational linguistics, all or part of the database instances in semi-structured databases, and structured documents in document processing. The most widely used current formats for data trees are json (the Java Script Object Notation) and xml (the eXtensible Markup Language).

We are interested in deciding the equivalence of programs that define transformations on data trees. For instance, we may consider xslt programs defining xml transformations or Linux installation scripts written in bash that change the file system tree. Our approach is to compile a subclass of such programs into classes of tree transducers for which equivalence is decidable. Here we present a partial landscape of classical classes of tree transducers without data values [7, 9, 13], where inclusion is read from left to right:



The class DTop^R of deterministic top-down tree transducers with regular lookahead by a deterministic bottom-up tree automaton is particularly well behaved [9]. It is closed under composition, which makes it suitable for compilation of programs, and its equivalence problem is decidable in NExpTime, by PTime reduction to the equivalence problem of the class DTop, for which equivalence is decidable in NExpTime [15, 12, 18]. Furthermore, the class DTop^R subsumes three other classes of tree transducers with pairwise incomparable expressiveness, which capture different aspects of programs: extended DTop with nested pattern matching, functional top-down tree transducers (functional Top) that relax the determinism requirement of DTop, and deterministic bottom-up tree transducers DBup that operate the other way around. The more general class of macro tree transducers (Mtt) is much more expressive (and includes lookaheads), but has a long-standing open equivalence problem [10, 11, 19] and fails to be closed under composition (though its linear size increase subclass has better properties).

In contrast to classical machines that operate on ranked trees over finite signatures, what we need for program verification are generalised machines that operate on data trees with infinite signatures. Most typically the data values which label the nodes of data trees may be strings over some finite alphabet or natural numbers. For dealing with data trees, the classical classes of tree transducers were extended to symbolic classes [21, 20, 16], and similarly for other kinds of finite state machines. The general idea is to use patterns for describing infinitely many data values in a finite manner, and to allow the transducers to apply transformations on the data values themselves.

We first illustrate by an example that the class of symbolic extended DTop is relevant in practice. For this, we consider the following extended DTop, which performs a routine cleanup and statistics task on a list of log files in a file system, as illustrated in the example of Figure 1.

$$q\langle \text{nil}\rangle \to \text{cons}\langle \text{file}\langle \text{“log”}, \text{“”}\rangle, \text{nil}\rangle \tag{1}$$
$$q\langle \text{cons}\langle x_1, x_2\rangle\rangle \to \text{cons}\langle q_{\text{name}}\langle x_1\rangle, q\langle x_2\rangle\rangle \tag{2}$$
$$q_{\text{name}}\langle \text{file}\langle \text{“log”}, x_2\rangle\rangle \to \text{file}\langle \text{“stats.1”}, f_{\text{stats}}@x_2\rangle \tag{3}$$
$$q_{\text{name}}\langle \text{file}\langle x_1 : \text{“stats.”}(\text{“0”}..\text{“9”})^+, x_2\rangle\rangle \to \text{file}\langle f_{\text{incr}}@x_1, f_{\text{id}}@x_2\rangle \tag{4}$$

Such transducers have nested patterns with variables $x_1, x_2, \ldots$ for matching subtrees, expressions for matching data values such as $\text{“stats.”}(\text{“0”}..\text{“9”})^+$ or cons, and applications of externally defined functions, such as $f_{\text{stats}}@x_2$, where f_{stats} is a string

transformation that produces statistics from its input string (log contents). It should be noted that symbolic functional Top are insufficient for this example since rules such as (3) and (4), with nested patterns, cannot be expressed in a top-down manner. In contrast, symbolic DBup offer an alternative solution for this concrete example.

Veanes and Bjørner [21, 20] started the study of symbolic transducers. They showed that equivalence is decidable for symbolic functional Top, if the corresponding problems on data pattern and transformations are. In this paper, we notice that the landscape of tree transducers above remains unchanged when turning classes of classical tree transducers symbolic. Therefore, we can show that the equivalence problem is decidable for the symbolic counterparts of all classes in the landscape except for Mtt. To see this, note that any symbolic DTop is a symbolic functional Top, so equivalence for symbolic DTop is decidable. Furthermore, equivalence for symbolic DTop^{R}s can be reduced to equivalence of symbolic DTop as in the classical case.

We then start studying the complexity of equivalence for classes of symbolic tree transducers. Our main result is that equivalence for symbolic DTop^{R}s is in NExpTime, under the assumption that operations on patterns and data transformations can be performed in PTime. If not, one needs to multiply the worst case exponential time with the maximal time needed for such operations. We obtain this result from a novel reduction from the equivalence of symbolic DTop to the equivalence of classical DTop, using a weakened version of the *origin-equivalence* in [3]. This reduction allows us to conclude that the equivalence problem of symbolic DTop is indeed in NExpTime as for classical DTop (and not in 2NExpTime as a naive analysis would lead to believe). Due to the modularity of the construction, the equivalence testers obtained for DTop^{R}s are easy to prove correct, to analyse, and to implement.

2 Tree Automata and Transducers

Some familiarity with formal languages and automata theory, as covered for instance in [5], is assumed.

Given a set S, we denote its cardinality by $|S|$ and its powerset by 2^S. The set of Boolean values is written $\mathbb{B} = \{0, 1\}$. $\mathbb{N}$ is the set of natural integers, including zero. We write $m..n$ the integer interval $[m, n] \cap \mathbb{N}$. We will denote tuples $(a_0, a_1, \ldots, a_n)$ by $a_0\langle a_1, \ldots, a_n\rangle$ or simply as a_0 if $n = 0$.

Let $X = \{x_1, x_2, x_3, \ldots\}$ be a set of **variables**. For $K > 0$, we shall often use the subsets $X_K = \{x_1, \ldots, x_K\}$. A **ranked alphabet** Σ is a (potentially infinite) set disjoint from X paired with a function $ar_\Sigma : \Sigma \to \mathbb{N}$ (or just ar where Σ is clear from context). The set of **ranked trees over Σ with variables in X**, denoted by $\mathcal{T}_\Sigma(X)$, is the least set that contains X and all $a\langle t_1, \ldots, t_n\rangle$ where $a \in \Sigma$, $n = ar_\Sigma(a)$, $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(X)$. $\mathcal{T}_\Sigma(\varnothing)$ is the set of **ground** Σ-trees, also written $\mathcal{T}_\Sigma$. Notions of position, substitution, etc., are all defined as usual.

We next recall the definitions of deterministic top-down tree automata (Dtta) and deterministic top-down tree transducers (DTop).

Definition 1. A **quasi** Dtta is a tuple $A = (\Sigma, Q, q_{ini}, rhs)$ such that Σ is a ranked alphabet, Q a finite set, $q_{ini} \in Q$, and rhs is a partial function that maps pairs $(q, a) \in Q \times \Sigma$ to tuples of $Q^{ar(a)}$. A Dtta is a quasi Dtta for which Σ is finite, and thus so is rhs.

The elements of Q are called the states of A, q_{ini} its initial state. The rules of A have the form $q\langle a\langle x_1, \ldots, x_n\rangle\rangle \to rhs(q, a)$, where $rhs(q, a)$ is defined and $n = ar(a)$. Each state q of a quasi Dtta A recognises a tree language $[\![q]\!]_A \subseteq \mathcal{T}(\Sigma)$, (or just $[\![q]\!]$ when A is clear from the context) defined by induction on the trees: we have $a\langle t_1, \ldots, t_n\rangle \in [\![q]\!]$ iff there is a rule $q\langle a\langle x_1, \ldots, x_n\rangle\rangle \to q_1, \ldots, q_n$ and $t_k \in [\![q_k]\!]$ for all $k \in 1..n$. The semantics of the automaton is $[\![A]\!] = [\![q_{ini}]\!]$.

Bottom-up tree automata (Buta) are defined similarly and as usual, with rules of the form $a\langle q_1, \ldots, q_n\rangle \to q$.

Definition 2. A **quasi** DTop is a tuple $M = (\Sigma, \Delta, Q, ax, rhs)$ such that Σ and Δ are ranked alphabets, Q is a finite set, $q_{ini} \in Q$, and rhs is a partial function that maps pairs $(q, a) \in Q \times \Sigma$ to $\mathcal{T}_\Delta(Q \times X_{ar(a)})$. A DTop is a quasi DTop for which Σ and Δ are finite, and thus rhs as well.

Σ and Δ are called input and output alphabets; the other components are as in automata. Each state $q \in Q$ has as semantics a partial function $[\![q]\!]$ from $\mathcal{T}_\Sigma$ to $\mathcal{T}_\Delta$, defined by induction on terms $t = a\langle t_1, \ldots, t_n\rangle \in \mathcal{T}_\Sigma$ such that:

$$[\![q]\!](t) \;=\; rhs(q, a)\big[q'\langle x_k\rangle \leftarrow [\![q']\!](t_k) \;\big|\; q' \in Q,\ k \in 1..ar(a)\big]. \tag{5}$$

The transformation defined by M is the partial function $[\![M]\!] = [\![q_{ini}]\!]$.

3 Symbolic Tree Automata and Transducers

In this section, we recall the definitions of symbolic Dtta and symbolic DTop as in [16].

Symbolic machines are finite representations of potentially infinite quasi Dtta and quasi DTop. They use **descriptors** to stand for the potentially infinite sets and functions. Given a set S, we call a set D paired with a function $[\![.]\!] : D \to S$ as set of descriptors of elements of S. For instance, we can use the set E of regular expressions $e \in E$ over an alphabet A as descriptors of regular languages $[\![e]\!] \subseteq A^*$. Outside of the definitions, we shall often assimilate the descriptors and their semantics.

Definition 3. A **symbolic** Dtta is a tuple $A = (\Sigma, \Phi, Q, q_{ini}, rhs)$ such that (Φ, Q, q_{ini}, rhs) is a quasi Dtta with a finite set of rules, Φ is an alphabet of descriptors for subsets of the alphabet Σ, with $ar(a) = ar(\varphi)$ for any $a \in [\![\varphi]\!]$ and $\varphi \in \Phi$. For all $a \in \Sigma$ and $q \in Q$, there exits at most one $\varphi \in \Phi$ such that $rhs(q, \varphi)$ is defined and $a \in [\![\varphi]\!]$.

The elements of $\varphi \in \Phi$ are called (descriptors for) guards. A symbolic DTTA A is a finite representation of a (potentially) infinite quasi DTTA A' such that for every rule r of form, $q\langle\varphi\langle x_1, \ldots, x_n\rangle\rangle \to q_1, \ldots, q_n$ of A, and for every $a \in [\![\varphi]\!]$, there is a rule $q\langle a\langle x_1, \ldots, x_n\rangle\rangle \to q_1, \ldots, q_n$ in A'. The semantics of A is defined as that of A': for all $q \in Q$, $[\![q]\!]_A = [\![q]\!]_{A'}$. Symbolic BUTA are defined similarly.

Definition 4. A symbolic DTTA is **effective** if it satisfies the following conditions, which we always assume: **(1)** The set of guards Φ is closed under conjunction and negation, i.e., there exists an algorithm computing some function $\wedge : \Phi \times \Phi \to \Phi$ such that $[\![\varphi \wedge \varphi']\!] = [\![\varphi]\!] \cap [\![\varphi']\!]$ for all $\varphi, \varphi' \in \Phi$, and an algorithm computing some function $\neg : \Phi \to \Phi$ such that $[\![\neg\varphi]\!] = \Sigma \setminus [\![\varphi]\!]$. **(2)** There exists an algorithm deciding membership $a \in [\![\varphi]\!]$ given a guard $\varphi \in \Phi$ and a label $a \in \Sigma$,

Definition 5. A **symbolic** DTOP is a tuple $M = (\Sigma, \Delta, \Phi, \mathcal{F}, Q, q_{ini}, rhs)$ such that $(\Phi, \mathcal{F}, Q, q_{ini}, rhs)$ is a quasi DTOP with a finite set of rules, $\mathcal{F}$ is an alphabet of descriptors for partial functions from the input alphabet Σ to the output alphabet Δ, with $ar([\![f]\!](a)) = ar(f)$ for every $a \in \mathrm{dom}(f)$ and $f \in \mathcal{F}$. The same conditions on Σ, Φ and rhs apply as for symbolic DTTA above.

The elements of $f \in \mathcal{F}$ are called (descriptors for) data transformations. A symbolic DTOP M is a finite representation of a (potentially) infinite quasi DTOP $M' = (\Sigma, \Delta, Q, ax, rhs')$, such that for every rule $q\langle\varphi\langle x_1, \ldots, x_n\rangle\rangle \to rhs(q, \varphi)$ of M, and for every $a \in [\![\varphi]\!]$, there is a rule $q\langle a\langle x_1, \ldots, x_n\rangle\rangle \to rhs(q, \varphi)[f \leftarrow [\![f]\!](a) \mid f \in \mathcal{F}]$ in M'. The semantics of M is defined as that of M': for all $q \in Q$, $[\![q]\!]_M = [\![q]\!]_{M'}$.

Definition 6. A symbolic DTOP is **effective** (which is assumed in the remainder) if the underlying DTTA is, and **(1)** There is an algorithm that computes the value of the data transformation $[\![f]\!](a)$ for a given $f \in \mathcal{F}$ and $a \in \Sigma$ and returns $\bot$ if it is not defined. **(2)** There is an algorithm that decides whether the image of a data transformation $[\![f]\!](\Sigma)$ is empty for a given $f \in \mathcal{F}$.

In **symbolic** DTOP^R, a symbolic BUTA on the transducer's input signature Σ first annotates the tree with its states P, and then the symbolic DTOP transforms the annotated tree on $\Sigma \times P$.

Consider the logs of an application on a Unix-flavoured system. The log is a text file named "log", here containing "s" for every successful login, and "f" for every failure. Every week, the old log is discarded and replaced by statistics: the number of successful and failed logins (Figure 1). For this, we denote by f_{stats} the function counting the numbers n, m of occurrences of "s" and "f" in a string (here a log's contents), and outputting the string n";"m. A fresh "log" is then created. The older log's statistics are named "stats.1", "stats.2", etc. so that higher numbers indicate older stats. To model this with a symbolic DTOP^R, we represent the contents of the logs folder by a list (CONS and NIL being the usual constructors) of files, each file being a tree of the form

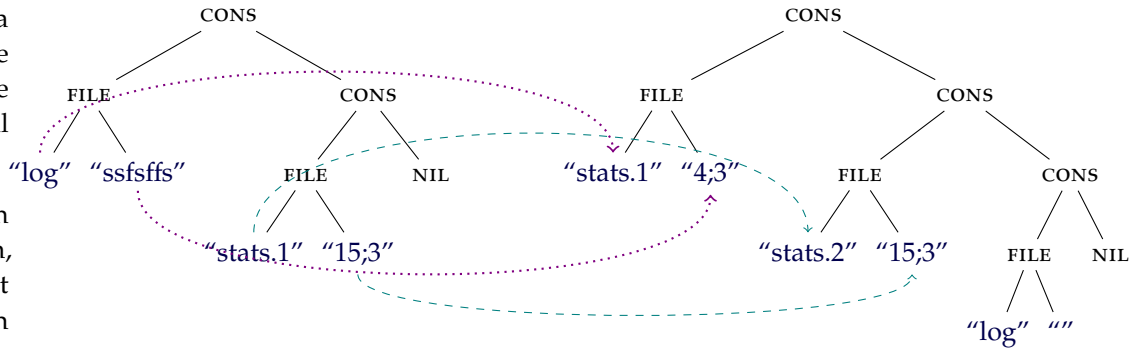


Fig. 1. Log cleanup and statistics: input tree on the left, output on the right.

FILE$\langle$"filename", "contents"$\rangle$. The input and output alphabets are thus strings, along with FILE, CONS and NIL. The guards will be a small subset of regular expressions on strings plus descriptors matching CONS and NIL. The lookahead's purpose is to check whether the filename matches a stat file or not (which cannot be done in a top-down transducer's rule, in contrast to the extended rules (3) and (4)) and to annotate the FILE nodes with its findings. Guards are regular expressions. The descriptor matching a specific string is the string itself; $*$ matches everything; "stats."("0"..."9")$^+$ is the descriptor matching stats filenames. The lookahead (LA) rules are

$$\text{"stats."}(\text{"0"}..\text{"9"})^+\langle\rangle \to p_{stats} \qquad \text{FILE}\langle p_{stats}, p\rangle \to p_{stats\ file}$$
$$\text{"log"}\langle\rangle \to p \qquad \text{FILE}\langle p, p\rangle \to p$$
$$(\text{"s"} \mid \text{"f"})^*\langle\rangle \to p \qquad \text{CONS}\langle\_, p\rangle \to p \qquad \text{NIL}\langle\rangle \to p$$

The label functions $\mathcal{F}$ are taken as, for instance, the class of rational functions, which we can implement with word transducers with lookahead [6], satisfying all requisite properties. We represent a constant function by the string it produces, the identity by f_{id}, and $f_{increment}$ for the function taking strings of the form "stats."k, where k is the decimal representation of an integer, and yielding "stats."$(k+1)$. We start in state q:

$$q\langle\text{NIL} : p\rangle \to \text{CONS}\langle\ \text{FILE}\langle\text{"log"}, \text{""}\rangle,\ \text{NIL}\rangle \tag{1'}$$
$$q\langle\text{CONS} : p\langle x_1, x_2\rangle\rangle \to \text{CONS}\langle q_{name}\langle x_1\rangle, q\langle x_2\rangle\rangle \tag{2'}$$
$$q_{name}\langle\text{FILE} : p\langle x_1, x_2\rangle\rangle \to \text{FILE}\langle q_{log}\langle x_1\rangle, q_{stats}\langle x_2\rangle\rangle \tag{3a}$$
$$q_{name}\langle\text{FILE} : p_{stats\ file}\langle x_1, x_2\rangle\rangle \to \text{FILE}\langle q_{incr}\langle x_1\rangle, q_{id}\langle x_2\rangle\rangle \tag{4a}$$
$$q_{incr}\langle\text{"stats."}(\text{"0"}..\text{"9"})^+ : p_{stats}\rangle \to f_{increment} \tag{3b}$$
$$q_{id}\langle * : p\rangle \to f_{id} \quad q_{stats}\langle * : p\rangle \to f_{stats} \quad q_{log}\langle\text{"log"} : p\rangle \to \text{"stats.1"}\ . \tag{4b}$$

4 Domain and Composition

To study problems like computation or equivalence on symbolic DTop, it is worth considering those problems as extensions of their counterparts for DTop. Indeed, most difficulties that previous papers [21, 20, 16] encountered are already relevant for the composition, normalization, or equivalence problems in the finite-labelled case [12, 17]. Most of those difficulties come from dealing with the domain of a transducer's transformation. Since these problems have been solved in DTop and proofs in symbolic DTop are essentially identical, we shall only present the results, a reference for the proofs in DTop, and the additional conditions required of Φ and $\mathcal{F}$ for them to carry over to the symbolic case.

The first important results concern automata and their expressive power. Symbolic Dtta which, as said before, we always assume to be effective, and have the classical properties of Dtta (e.g. in [5]).

Lemma 7. **(1)** *The class of languages described by symbolic* Dtta *is closed under intersection.* **(2)** *If equivalence is decidable on* Φ*, then equivalence is decidable on symbolic* Dtta.

The second important result concerns the domains of symbolic DTop.

Lemma 8. *Let* M *be a symbolic* DTop*. Then we can build a symbolic* Dtta A *such that* $[\![A]\!] = \mathrm{dom}([\![M]\!])$.

Proof. This is directly adaptable from the classical construction on DTop: see [8, 12] and [2, Lem. 39 p57]. The nub of the constrcuction is that, as subtrees can be copied, each state of A corresponds to a set of states that explore the same subtree during the run. In the worst case scenario, this results in a full powerset construction, and thus leads to an exponential blowup in the number of states. Note that in the symbolic case, we need to compute the intersections of all the guards that may be invoked on the same subtree. □

The third important result concern the composition of symbolic DTop. In [16], the authors state that *"the proof of the first statement* [of [21], that their algorithm computes the composition of any two symbolic DTop, and that the result is a symbolic DTop] *is insufficient.* [They proceed to show why that specific algorithm is incorrect] *We even conjecture that this statement is wrong, i.e., symbolic* DTop *are not closed under composition."* Indeed they are not, and for the same reasons that DTop themselves are not closed under composition. This is mainly due to the problem of domain restriction.

Example 9. For $a, b \in U$, it is easy enough to make a DTop that computes τ_1, the identity over trees of form $a(b, t_2)$, and a DTop that computes τ_2, that sends all trees of the form $a(t_1, t_2)$ to their right subtree t_2. However, no DTop can compute $\tau_2 \circ \tau_1$, that sends all trees of form $a(b, t_2)$ to their right subtree t_2: such a transducer would need to start in a state q_0 that only has one rule of form $q_0\langle a\langle x_1, x_2\rangle\rangle \to t$. This rule cannot produce any output symbol, and thus t is of the form $q\langle x_i\rangle$. This x_i must be x_1, as the transducer needs to visit the left subtree to ensure that it is b. This x_i must also be x_2, as the transducer needs to copy the right subtree. Hence, no such DTop can exist.

Note that this argument applies equally to symbolic DTop and to DTop; only the general form of rules matter. To find a class closed under composition, a solution presented in the DTop case [12] is to consider transducers *with domain inspection*: a **symbolic** DTop **with inspection** is a pair $N = (M, A)$ of a symbolic DTop M and a symbolic Dtta A. Its semantic is $[\![N]\!] = [\![M]\!]_{|[\![A]\!]}$, the function of M restricted to the language of A. We know that DTop with inspection are closed under composition [12]. This result extends to symbolic DTop if the set of functions of $\mathcal{F}$ is itself closed under composition, and the images of guards through functions of $\mathcal{F}$ form a suitable set of guards (i.e. they satisfy the requirements for effectiveness).

Lemma 10. *Let* $\mathcal{F}$ *be closed under composition, and* N, N' *be two effective symbolic* DTop *with inspection using functions of* $\mathcal{F}$*. Then we can build a symbolic* DTop *with inspection* N'' *such that* $[\![N'']\!] = [\![N]\!] \circ [\![N']\!]$.

The main intuition behind the generalisation of the classical results [1] is presented in several papers [21, 20, 16]; roughly, a rule in N'' is the image of the right-hand side of a rule of N' by a state of N. To obtain closure by composition for symbolic DTop – or indeed for DTop, as the problem is fundamentally unchanged by the alphabets – necessitates the use of either very strong restrictions, as in [16], or the use of domain inspection, which we prefer here.

5 Deciding Equivalence

In this section, we show that, given a few basic properties on label transformations (mostly that equivalence is decidable for label transformations) the equivalence problem for symbolic DTop is decidable, regardless of linearity, by reducing that problem to equivalence for DTop, which is known to be decidable.

The main observation behind the reduction is that, although Σ, Φ and $\mathcal{F}$ may well be infinite, only a finite number of predicates and transformations are actually used in a symbolic DTop (or indeed, any pair of symbolic DTop), and give rise to a finite number of behaviours. These finite subsets of Φ and $\mathcal{F}$ will serve as finite input and output signatures in our reduction, which encodes those finitely many behaviours.

We shall first present a straightforward but coarse reduction, which captures *most* of the ways in which two symbolic DTop may differ, with the exception of some more technical cases where the *origins* of produced nodes matter. The notion of origin will then be presented and used to complete the construction.

5.1 The Basic Reduction

The first reduction is as follows: let $M = (\Phi, \Sigma, \mathcal{F}, \Delta, P, p_{ini}, R)$ and $N = (\Phi, \Sigma, \mathcal{F}, \Delta, Q, q_{ini}, S)$ be two symbolic DTop. We build their DTop **representations**, the DTop $\underline{M}$ and $\underline{N}$. Strictly speaking we should write $\underline{M}^{M,N}$ and $\underline{N}^{M,N}$, as the construction is specific to the pair of transducers under consideration, and the same applies to the representation of each component of the transducers – $\underline{\Phi}$, $\underline{\Sigma}$, etc – which we define below. In this section we assimilate descriptors φ, f and their semantics $[\![\varphi]\!], [\![f]\!]$ to lighten the notations.

We make the following additional **equivalence-testing assumptions**: for all $\varphi, \psi \in \Phi$, it is decidable whether $\varphi = \psi$. For all $f, g \in \mathcal{F}$ and all $\varphi \in \Phi$, it is decidable whether there exists some $c \in \Delta$ such that $f(\varphi) = \{c\}$, and this c is computable; and it is decidable whether $f|_\varphi = g|_\varphi$.

The finite information relevant to the behaviour of symbolic DTop is which guards are satisfied. Thus we let

$$\Pi \;=\; \{\, \mathrm{gd}(r) \mid r \in R \cup S \,\} \;\subseteq\; \Phi \tag{6}$$

be the subsets of guards actually used by either one of the two transducers.

The **representation of $a \in \Sigma$** is the equivalence class of $a \in \Sigma$ with respect to the tests of Π; that is to say, the set of all labels which are indistinguishable from a by our transducers:

$$\underline{a} \;=\; \{\, b \in \Sigma \mid \forall \pi \in \Pi,\ a \in \pi \Leftrightarrow b \in \pi \,\}, \tag{7}$$

which is computed as

$$\underline{a} \;=\; \bigcap_{a \in \pi \in \Pi} \pi \setminus \bigcup_{a \notin \pi \in \Pi} \pi\,. \tag{8}$$

The **finite alphabet $\underline{\Sigma}$ representing Σ** is defined as $\underline{\Sigma} = \{\, \underline{a} \mid a \in \Sigma \,\}$. Note that this is indeed a finite set, as Π is finite.

To properly represent a guard $\varphi \in \Pi$, we must make sure to capture all the possible circumstances in which the guard is used; recall from Lemma $8_{[p4]}$ that a subtree may be evaluated in any number of states during a run, and thus simultaneously tested by any number of guards, each combination corresponding to a different behaviour of the transducer, and to a new, finer guard. Thus the representation of a guard $\varphi \in \Pi$ must be the set of all the (finer) guards obtained by such combinations. Those combinations are precisely the equivalence classes of all labels that satisfy φ. Thus the **representation of a guard $\varphi \in \Pi$** is defined as the set of equivalence classes of labels accepted by φ:

$$\underline{\varphi} \;=\; \left\{\, \underline{a} \;\middle|\; a \in \Sigma,\ a \in \varphi \,\right\}. \tag{9}$$

The **representation of an input tree $t \in \mathcal{T}(\Sigma)$** is defined inductively as

$$\underline{a(t_1, \ldots, t_n)} \;=\; \underline{a}(\underline{t_1}, \ldots, \underline{t_n}) \in \mathcal{T}(\underline{\Sigma})\,. \tag{10}$$

For the representation of a label transformation f, it is important to store the *actual* transformation that can be performed during a run, which means restricting by the appropriate guard. There is also a more subtle technicality. We usually store the function itself, except in the case where only a constant can be produced: then one must store only the constant itself, regardless of the input domain. This is linked to the question of the *origin* of a node, which will be explained in the next sections. Thus the **representation of a label transformation f restricted to φ** is defined as

$$\underline{f|_\varphi} \;=\; \begin{cases} c & \text{if } f(\varphi) = \{c\}, \text{ and} \\ f|_\varphi & \text{otherwise.} \end{cases} \tag{11}$$

The **representation of a rule $r \in R \cup S$**, of the form $r = q\langle \varphi\langle x_1, \ldots, x_n\rangle\rangle \to t$, is given by the set $\underline{r}$ of all classical rules

$$q\langle \rho\langle x_1, \ldots, x_n\rangle\rangle \to t\big[f \leftarrow \underline{f|_\rho} \;\big|\; f \in \mathcal{F}\big]\,, \tag{12}$$

where $\rho \in \underline{\varphi}$. Letting $\underline{R} = \bigcup_{r \in R} \underline{r}$ and $\underline{S} = \bigcup_{s \in S} \underline{s}$, we finally have $\underline{M} = (\underline{\Sigma}, \underline{\Delta}, P, p_{ini}, \underline{R})$ and $\underline{N} = (\underline{\Sigma}, \underline{\Delta}, Q, q_{ini}, \underline{S})$.

5.2 Examples; The Necessity of Origins

It is easy to see that if $\underline{M}$ and $\underline{N}$ are not equivalent, then neither are M and N, as a difference in the representation means that the symbolic DTop must either take in different inputs, produce tree with different shapes, of produce different labels. Let us visuallise that on an example:

Example 11. Consider M of initial state p and rules

$$p\langle \text{“c”}*\langle x_1, x_2\rangle\rangle \to f_{id}\langle p\langle x_1\rangle, p\langle x_2\rangle\rangle \tag{13}$$

$$p\langle \text{“a”}*\rangle \to f_{id} \qquad p\langle \text{“b”}*\rangle \to f_{id} \tag{14}$$

and N of initial state q and rules

$$q\langle \text{“c”}*\langle x_1, x_2\rangle\rangle \to f_{id}\langle q\langle x_2\rangle, q\langle x_1\rangle\rangle \tag{15}$$

$$q\langle \text{“a”}*\rangle \to f_{id} \qquad q\langle \text{“b”}*\rangle \to f_{id} \tag{16}$$

The representation of M is easily computed, as the guards of Π are disjoint, and describes the transformation

$$[\![\underline{M}]\!] \;=\; \begin{array}{ccc} \text{“c”}* & \mapsto & f_{id}|_{\text{“c”}*} \\ \diagup\ \diagdown & & \diagup\ \diagdown \\ \text{“a”}*\quad \text{“b”}* & & f_{id}|_{\text{“a”}*}\quad f_{id}|_{\text{“b”}*} \end{array}\,. \tag{17}$$

In that case, this is even a finite transformation – there is a single possible input tree. Likewise for $\underline{N}$:

$$[\![\underline{N}]\!] = \begin{array}{ccc} \text{“c”}* & \mapsto & f_{id}|_{\text{“c”}*} \\ \text{“a”}* \quad \text{“b”}* & & f_{id}|_{\text{“b”}*} \quad f_{id}|_{\text{“a”}*} \end{array} . \tag{18}$$

We have $[\![\underline{M}]\!] \neq [\![\underline{N}]\!]$, as $f_{id}|_{\text{“a”}*} \neq f_{id}|_{\text{“b”}*}$, and this reflects the difference of behaviour between M and N.

Example 12. Consider now M of initial state p and rules

$$p\langle \text{“c”}*\langle x_1, x_2\rangle\rangle \to f_{id}\langle p\langle x_1\rangle, p\langle x_2\rangle\rangle \tag{19}$$
$$p\langle \text{“a”}*\rangle \to \text{“a”} \qquad p\langle \text{“b”}*\rangle \to \text{“a”} \tag{20}$$

and N of initial state q and rules

$$q\langle \text{“c”}*\langle x_1, x_2\rangle\rangle \to f_{id}\langle q\langle x_2\rangle, q\langle x_1\rangle\rangle \tag{21}$$
$$q\langle \text{“a”}*\rangle \to \text{“a”} \qquad q\langle \text{“b”}*\rangle \to \text{“a”} \tag{22}$$

Since we produce constants, by $(11)_{[p5]}$ we store this time the constant itself, and not the restricted function; thus we have the representations:

$$[\![\underline{M}]\!] = [\![\underline{N}]\!] = \begin{array}{ccc} \text{“c”}* & \mapsto & f_{id}|_{\text{“c”}*} \\ \text{“a”}* \quad \text{“b”}* & & \text{“a”} \quad \text{“a”} \end{array} . \tag{23}$$

This is good, as in that case, it does not matter whether the constant “a” was obtained from an input label in “a”$*$ or in “a”: M and N are equivalent regardless. Had we stored the constant function with its domain restrictions instead of the constant itself, we would have obtained $[\![\underline{M}]\!] = [\![\underline{N}]\!]$, and thus a false negative. However, there is work left to do in order to obtain the reciprocal property; that is to say, to ensure that if the representations are equivalent, then so must the originals. A variation of Example 11 provides the counter-example:

Example 13. Consider M of initial state p and rules

$$p\langle \text{“c”}*\langle x_1, x_2\rangle\rangle \to f_{id}\langle p\langle x_1\rangle, p\langle x_2\rangle\rangle \tag{24}$$
$$p\langle \text{“a”}*\rangle \to f_{id} \qquad p\langle \text{“a”}*\rangle \to f_{id} \tag{25}$$

and N of initial state q and rules

$$q\langle \text{“c”}*\langle x_1, x_2\rangle\rangle \to f_{id}\langle q\langle x_2\rangle, q\langle x_1\rangle\rangle \tag{26}$$
$$q\langle \text{“a”}*\rangle \to f_{id} \qquad q\langle \text{“a”}*\rangle \to f_{id} \tag{27}$$

We have:

$$[\![\underline{M}]\!] = [\![\underline{N}]\!] = \begin{array}{ccc} \text{“c”}* & \mapsto & f_{id}|_{\text{“c”}*} \\ \text{“a”}* \quad \text{“a”}* & & f_{id}|_{\text{“a”}*} \quad f_{id}|_{\text{“a”}*} \end{array} . \tag{28}$$

Yet it is not the case that M and N are equivalent: consider the input tree $t = \text{“c”}\langle \text{“a”}, \text{“aa”}\rangle$: we have $[\![M]\!](t) = t$, whereas $[\![N]\!](t) = \text{“c”}\langle \text{“aa”}, \text{“a”}\rangle$. Though the same operation is done on the two children, that does not make them interchangeable. Thus we must, additionally, have in the representation some means of tracking the *origin* is output labels; that is to say, from which node they are produced – left child or right child, in that case. This origin information must apply even if the domains of the transformations are identical.

Yet it must be noted that, if we combined this example with the previous one, producing the same constant left and right, we would *again* be in a case where no origin information should be stored, as it won't matter whether the constant “comes from” the right or left child.

The next section formally introduces the notion of *origin*.

5.3 Origins and Origin Equivalence

The problem identified in Example 13 is that a reduction from symbolic DTop to DTop can only be accurate if it stores not only the information relative to which guard or function was used to transform input labels into output labels, but also enough information to know which input label is transformed into which output label. This notion exists in previous papers: notably, [3] discusses *origin*, a relation between the input and output nodes of a transformation. We will define similar notions, identifying for each output node the input node, guard, and function used for its production.

We define a position to be a word on the alphabet $\mathbb{N}$. We can identify each node of a tree $s \in \mathcal{T}_\Sigma$ with a unique position.

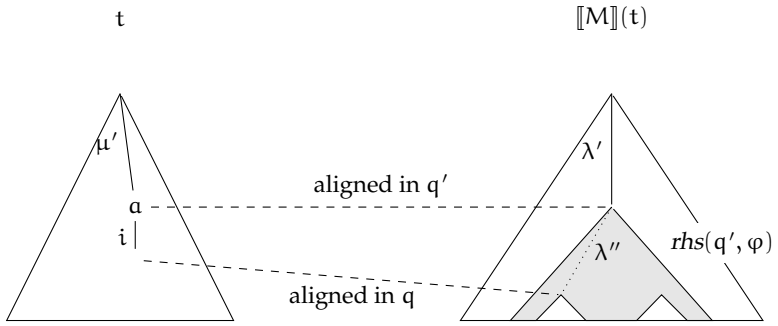
- The root of s is at the position ε.
- If a node of s is at position μ and labeled by $a \in \Sigma$ of arity k, then its i^{th} child has the position μi, where $1 \leqslant i \leqslant k$.

For instance, in Figure 1, the node 1 is labelled by FILE, and the node 12 in the input is labelled by “ssfsffs”.

We use this notion of position to talk of *syntactic alignment*. This notion, found in [2], aims to describe which part of the input is read by a symbolic DTop to produce a given part of the output. It is, as its name implies, a purely syntactic property, describing the functioning of the transducer. We define this notion recursively:

Definition 14. Let M be a symbolic DTop, and $t \in \mathrm{dom}([\![M]\!])$ a tree. We define by induction what it means for μ a position of t and λ a position of $[\![M]\!](t)$ to be syntactically aligned in state q (we write $\mu \sim_q \lambda$) :

- $\mu = \varepsilon$ and $\lambda = \varepsilon$ are aligned in state q_{ini}.
- If $\mu' \sim_{q'} \lambda'$, the node at position μ' in t is labeled by a, and there exists a rule $q'\langle\varphi\langle x_1, \ldots, x_n\rangle\rangle \to \mathit{rhs}(q', \varphi)$ such that $a \in \varphi$, and furthermore there is a node of $\mathit{rhs}(q', \varphi)$ at position λ'' labeled by $q\langle x_i\rangle$, then $\mu' i$ and $\lambda'\lambda''$ are aligned in state q.



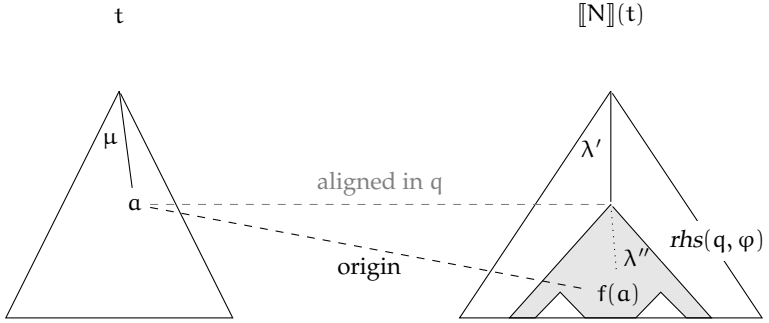
We use the notion of syntactic alignment to define a notion of origin similar to the one found in [3]. The intuition of this notion goes as follows: let M be a symbolic DTop, $t \in \mathrm{dom}([\![M]\!])$, and λ a position in $[\![M]\!](t)$. We want to know which node of t was read by M when the label at position λ was produced in $[\![M]\!](t)$. We call this input node the origin of the output node at position λ. Note that in the symbolic case, we also are interested to track the guard of the rule used to produce said node, and the function used to transform an input label in the output label at position λ.

Definition 15. Let M be a symbolic DTop, $t \in \mathrm{dom}([\![M]\!])$ a tree, and λ a position of $[\![M]\!](t)$. We say that μ is the origin of λ if:

- μ is the position of a node of t labeled by some $a \in \Sigma$
- there exists λ', λ'' such that $\lambda = \lambda'\lambda''$ and $\mu \sim_q \lambda'$
- there exists a rule $q\langle\varphi\langle x_1, \ldots, x_n\rangle\rangle \to \mathit{rhs}(q, \varphi)$ such that $a \in \varphi$, and λ'' is the position of a node of $\mathit{rhs}(q, \varphi)$ labeled by some $f \in \mathcal{F}$.

We call f the *origin function* of λ, and φ the *origin guard* of λ.

Note that for a symbolic DTop M and a tree t, each position λ of $[\![M]\!](t)$ has exactly one origin: each output node is produced during one and only one step of the computation of $[\![M]\!](t)$. For instance, in Example 13, the node at position 1 in an output of M comes from the input node at position 1. Its origin guard is $\text{“a”}*$ and its origin function is



$f_{id}|_{\text{“a”}*}$. Conversely, in N, the node at position 1 in an output of M also has the origin guard $\text{“a”}*$ and the origin function $f_{id}|_{\text{“a”}*}$, but its origin is the input node of position 2. This difference in origins is not detected in the reduction attempt of Section 5.1.

In [3], the notion of origin is accompanied by a notion of origin equivalence. In essence, two transducers are origin-equivalent if they are equivalent and every output node is produced from from the same origins in both transducers.

Definition 16. Two equivalent symbolic DTop M, N are *origin-equivalent* iff for all $t \in \mathrm{dom}[\![M]\!]$, for any node λ of $[\![M]\!](t)$, the origins of λ for M and N are identical.

The advantage of origin equivalence is that if the origins are known, origin equivalence becomes easier to test than equivalence (see [3]). Origin equivalence is stronger than equivalence: completing the reduction of 5.1 to embed information about the origin would prevent the false positives presented in Example 13, in which M and N are not equivalent. Although $\underline{M}$ and $\underline{N}$ are equivalent, they are not origin-equivalent: the output node at position 1 is of origin 1 for M and $\underline{M}$, but 2 for N and $\underline{N}$.

Unfortunately, origin equivalence is strictly stronger than equivalence, which means that by getting rid of false positives, we might introduce false negatives. In Example 12, M and N are equivalent, and, again, although $\underline{M}$ and $\underline{N}$ are equivalent, they are not origin-equivalent: the output node at position 1 is of origin 1 for M and $\underline{M}$, but 2 for N and $\underline{N}$.

To prevent this kind of false negative, we do not encode *all* origins, as we do not want to fully test for origin equivalence. Instead, we relax the notion of origin equivalence: the origin of an output node only matters if its origin function can produce different results for different input labels. If the origin function is constant, as in Example 12, then the origin node is of no relevance to determine equivalence. We define the notion of weak-origin equivalence as origin equivalence everywhere except when constants are produced.

Definition 17. Two equivalent symbolic DTop M, N are *weak-origin–equivalent* iff for all $t \in \mathrm{dom}[\![M]\!]$, for any node λ of $[\![M]\!](t)$, the origin nodes of λ for M and N are identical, or its origin functions for M and N are constant functions of same image.

Note that when we consider these origin functions, we consider them restricted to the origin guard. In the case of a rule $q\langle\varphi\langle x_1,\dots,x_n\rangle\rangle \to \mathit{rhs}(q,\varphi)$, the function we want to test as constant is in fact $f|_\varphi$, not f itself. However, it is quite possible that equivalent transducers use different tests to compute the same transformations. This would needlessly affect the notion of weak-origin equivalence. To ensure no such problem arises, we introduce the notion of harmonized symbolic DTop, to say that they use the same guards to read the same nodes.

Definition 18. Two symbolic DTop M, N are *harmonized* iff for all $t \in \mathrm{dom}[\![M]\!] \cap \mathrm{dom}[\![N]\!]$, for any node λ of $[\![M]\!](t)$ and λ' of $[\![N]\!](t)$, if the origin of λ for M and λ' of N are identical, then their origin guards are identical.

It is easy to see that any two symbolic DTop can be harmonized. A way to do so would be to render all guards disjoint, similarly to (9)[p5] by computing all possible combinations of guards from the two transducers. While this method is not parsimonious, it shows that any pair of symbolic DTop can be harmonized without any change to their semantics by dividing their rules in small enough components.

We know that weak-origin equivalence is strictly weaker than origin equivalence. As seen previously in Example 12, $\underline{M}$ and $\underline{N}$ are not origin-equivalent as the output node at position 1 is of origin 1 for M and $\underline{M}$, but 2 for N and $\underline{N}$. However, this distinction disappears with weak-origin equivalence, as the origin function of 1 in both M and N is constant of image "a": M and N are weak-origin–equivalent.

By definition, weak-origin equivalence is at least as strong as equivalence. We will show that for harmonized symbolic DTop, both these notions are, in fact, identical.

Lemma 19. *Two harmonized symbolic* DTop *are equivalent if and only if they are weak-origin–equivalent.*

Proof. Since weak-origin equivalence assumes equivalence, we only have to prove that two equivalent symbolic DTop are necessarily weak-origin–equivalent.

Let M, N be two equivalent symbolic DTop, and suppose they are not weak-origin–equivalent. There is a tree $t \in \mathrm{dom}([\![M]\!])$, and a node λ of $[\![M]\!](t)$ such that μ is its origin node for M and a different node μ' is its origin node for N. Furthermore, one of its origin functions (we pick f its origin function for M, and φ its origin guard) is not constant. Since f is not constant we can replace the label at position μ in t from $a \in \varphi$ to another $a' \in \varphi$ such that $f(a) \neq f(a')$. By replacing *only* the label of μ in t we thus build another tree t'. Since M and N are harmonized, N uses the same guard φ as M to read the input label at μ: thus this substitution bears no incidence on the rules used to compute $[\![N]\!](t')$. The label of λ in $[\![M]\!](t')$ changed from $f(a)$ to $f(a')$. However, since the label of μ' has not changed in the input, the label of λ has not changed in $[\![N]\!](t')$. This means $[\![M]\!](t') \neq [\![N]\!](t')$, which contradicts the equivalence of M and N. □

5.4 An Origin-Aware Reduction

We extend the reduction of 5.1 to include enough origin information to translate weak-origin equivalence. As we have seen previously, the origin of an output node is relevant if and only if a change of input label yields a change in output labels. To reduce this property to a finite signature, we create two distinct input labels for each guard φ, which will produce one of two distinct output labels for each non-constant function f. First, we create for each guard two distinct labels by placing, on each input node, a bit: the **representations of an input tree $t \in \mathcal{T}(\Sigma)$** becomes

$$\underline{a(t_1,\dots,t_n)} \;=\; \left\{ \underline{(a,b)}(u_1,\dots,u_n) \;\middle|\; b \in \mathbb{B},\ u_i \in \underline{t_i}, \forall i \right\} \;\subseteq\; \mathcal{T}(\underline{\Sigma} \times \mathbb{B})\,, \tag{29}$$

which is as before with the addition of bits b, called **obit** (origin bit). Any of these bits can be set at 0 or 1, leading to several representations for each tree of $\mathcal{T}(\Sigma)$. To encode origin, this bit is carried over to the output by the rules of $\underline{M}$, but only when a change of input labels yields a change in output labels. Accordingly, the **representation of a label transformation f restricted to φ, with obit b** is redefined as

$$\underline{f|_\varphi, b} \;=\; \begin{cases} c & \text{if } f(\varphi) = \{c\}, \text{ and} \\ (f|_\varphi, b) & \text{otherwise,} \end{cases} \tag{30}$$

and the **representation of a rule $r \in R \cup S$**, of the form $r = q\langle\varphi\langle x_1,\dots,x_n\rangle\rangle \to t$, is given by the set $\underline{r}$ of all classical rules

$$q\langle(\rho,b)\langle x_1,\dots,x_n\rangle\rangle \to t\left[f \leftarrow \underline{f|_\rho, b} \;\middle|\; f \in \mathcal{F}\right], \tag{31}$$

for all $b \in \mathbb{B},\ \rho \in \underline{\varphi}$.

Consider the effect of this change on Example 13: we now have

$$[\![\underline{M}]\!] \;\ni\; \begin{array}{ccc} & \text{“c”}*,0 & \\ \text{“a”}*,1 & & \text{“a”}*,0 \end{array} \;\mapsto\; \begin{array}{ccc} & f_{id}|_{\text{“c”}*},0 & \\ f_{id}|_{\text{“a”}*},1 & & f_{id}|_{\text{“a”}*},0 \end{array}\,, \tag{32}$$

whereas

$$[\![\underline{N}]\!] \;\ni\; \begin{array}{ccc} & \text{“c”}*,0 & \\ \text{“a”}*,1 & & \text{“a”}*,0 \end{array} \;\mapsto\; \begin{array}{ccc} & f_{id}|_{\text{“c”}*},0 & \\ f_{id}|_{\text{“a”}*},0 & & f_{id}|_{\text{“a”}*},1 \end{array}\,, \tag{33}$$

and thus, properly, $[\![\underline{M}]\!] \neq [\![\underline{N}]\!]$.

A convenient way of thinking about obits is to envision the input trees where a single label bears a 1 as obit; then the set of output labels bearing a 1 as obit is exactly the set of nodes which have that input node as origin. For instance, in Figure 2, two output

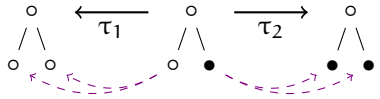


Fig. 2. Using obits to deduce origins. We represent the obits with colours: ∘ are nodes of obit 0, • are nodes of obit 1. We apply two transformations τ_1 and τ_2 on an input tree (here in the middle): τ_1 replaces its right leaf by a copy of the left one, while τ_2 replaces its left leaf by a copy of the right one. Visually, "turning on" an obit in the input makes it possible to visualise all output nodes that depend upon it.

nodes share the same origin. Of course, we do not always store the origin information: in Example 12, we would still store only the constant, and discard information about both the transformation and the origin, thus avoiding a false negative.

This construction leads naturally to some interesting properties between $\underline{M}$ and M, notably that the domains and origins are properly translated.

Lemma 20. *Let* M *be a symbolic* DTop *and* $\underline{M}$ *its representation.*

- $\mathrm{dom}[\![\underline{M}]\!] = \bigcup_{t \in \mathrm{dom}[\![M]\!]} \underline{t}$,
- *for every* $s \in \underline{t}$*, if* μ *is the origin of* λ *in* $\underline{M}$*, then* μ *is the origin of* λ *in* M.

The fact that $\underline{M}$ translates the functioning of M in a finite signature means that we can now reduce the equivalence problem in symbolic DTop to the equivalence problem in DTop.

Theorem 21. *Let* M, N *be two symbolic* DTop*. Then* $[\![M]\!] = [\![N]\!]$ *if and only if* $[\![\underline{M}]\!] = [\![\underline{N}]\!]$.

Proof. First, by Lemma 20, the domain of $\underline{M}$ is the set of all the representations of trees of $\mathrm{dom}([\![M]\!])$. Hence $\underline{M}$ and $\underline{N}$ are of same domain if and only if M and N are of same domain.

Let us now assume that the domains of M and N are the same. We suppose they are harmonized, as described under Definition 18, by cutting all guards into the fragments described in $(9)_{[p5]}$.

Suppose M and N are not equivalent; let t be an input tree such that $u = [\![M]\!](t) \neq [\![N]\!](t) = v$. We consider a position λ that exists both in u and v but where the label at λ differs in u and v. In M the origin node of λ is μ; the origin function is $f|_\varphi$. In N the origin node of λ is μ'; the origin function is $g|_\psi$.

If $\mu = \mu'$ and $f|_\varphi = g|_\psi$, there would not be a difference between the labels of u and v at position λ. Hence of of these equalities must not hold. Another point worth noting is that since the labels of u and v at position λ differ, then $f|_\varphi$ and $g|_\psi$ cannot be constant functions of same image.

If $\mu \neq \mu'$, i.e. if the origins of λ are different for M and N, then they also are for $\underline{M}$ and $\underline{N}$ for any $s \in \underline{t}$ by Lemma 20. We choose an s whose label at μ is $(\varphi, 0)$ and whose label at μ' is $(\psi, 1)$. Then the label at λ in $[\![\underline{M}]\!](s)$ is $\underline{f|_\varphi, 0}$ while the label at λ in $[\![\underline{N}]\!](s)$ is $\underline{g|_\psi, 1}$. The only way both of these labels could be the same despite the obit difference would be for $f|_\varphi$ and $g|_\psi$ to be constant of same image, which we know to be in contradiction with our suppositions. Hence in this case $[\![\underline{M}]\!] \neq [\![\underline{N}]\!]$.

If $f|_\varphi \neq g|_\psi$, i.e. if the origin functions of λ are different for M and N, then for any $s \in \underline{t}$ the label at λ in $[\![\underline{M}]\!](s)$ is of form $\underline{f|_\varphi, b}$ with some $b \in \{0, 1\}$, while the label at λ in $[\![\underline{N}]\!](s)$ is of form $\underline{g|_\psi, b'}$ with some $b' \in \{0, 1\}$. The only way both these labels could be the same despite the function difference would be for $f|_\varphi$ and $g|_\psi$ to be constant functions of same image, which is again in contradiction with our suppositions. Hence in this case $[\![\underline{M}]\!] \neq [\![\underline{N}]\!]$.

Conversely, suppose $\underline{M}$ and $\underline{N}$ are not equivalent; let s be an input tree such that $u' = [\![\underline{M}]\!](s) \neq [\![\underline{N}]\!](s) = v'$. We consider a position λ that exists both in u' and v' but where the label at λ differs in u' and v'. In $\underline{M}$ the origin node of λ is μ, and in $\underline{N}$ the origin node of λ is μ'. Thus, μ and μ' are also the origin node of λ in M and N respectively for all t such that $s \in \underline{t}$. As we did previously, we first note that the labels at position λ in u' and v' cannot be the same constant c.

If $\mu \neq \mu'$, then in M and N, for an input tree t such that $s \in \underline{t}$, the origins of λ are different, and the origin functions are not both constants of same image, as the labels at position λ in u' and v' cannot be the same constant c. This means that M and N are not weak-origin–equivalent, and thus, by Lemma 19, not equivalent.

If $\mu = \mu'$, then the label at λ in u' and v' are representations of different functions $f|_\varphi \neq g|_\psi$. Since M and N are supposed harmonised, we know that $\varphi = \psi$. This means that for $f|_\varphi$ and $g|_\psi$ to be different, there must exist at least one value $a \in \varphi$ such that $f(a) \neq g(a)$. We pick a tree t such that $s \in \underline{t}$ and the label at μ in t is labeled a. The node at position λ in $[\![M]\!](t)$ is labeled $f(a)$, while the node at position λ in $[\![N]\!](t)$ is labeled $g(a)$. Hence $[\![M]\!] \neq [\![N]\!]$. □

Corollary 22. *Under the equivalence-testing assumptions, the equivalence problem for symbolic* DTop *is decidable.*

Proof. Under the equivalence-testing assumptions, the construction of the representation is effective; thus computing the representations and testing their equivalence is an algorithm, as equivalence is decidable for DTop. □

6 Algorithmic Complexity

In this part we study the exact complexity of the algorithm to test equivalence that Corollary 22 points out. Two main points are relevant. First, we note that since our reduction involves a potentially exponential number of various operations related to the equivalence-testing assumptions (computing intersections, deciding function equivalence), their complexity plays an important part in how efficiently symbolic

DTop equivalence can be tested. Second, as previously mentioned, the number of such operations can be made more parsimonious than the method we present in (9)[p5]: while the worst case scenario inevitably requires an exponential number of intersections, negations, and emptiness tests on Φ, we do not need to build all guards $\underline{a}$. Such a construction would only combine all the guards a node can actually encounter in a given symbolic DTop: if a position λ can never be read by a rule of guard φ, we do not care know whether its label satisfies φ, even if φ is otherwise used in the transducer. This construction would be an adaptation of the constuction of the domain automaton of a DTop or the construction of a *compatible* DTop as presented in Lemma 39 and Proposition 62 of [2].

Corollary 23. *Under the equivalence-testing assumptions above, the equivalence problem for symbolic* DTop *is reducible to the equivalence problem on* DTop*, in* ExpTime *in the worst case, plus, at worst, an exponential number of operations in Φ and $\mathcal{F}$.*

We note that some realistic cases are more reasonable than this theoretical bound. In the case where guards are all disjoint, the reduction to DTop actually requires a polynomial number of operations in Φ and $\mathcal{F}$. In practice, it can be expected that few intersections actually need to be computed.

In any case, we can express the number of states and rules in $\underline{M}$ and $\underline{N}$ independently of Φ and $\mathcal{F}$: the states are unchanged, and the number of rules increases, at worst, exponentially:

Lemma 24. *For M, N two symbolic* DTop*, their* DTop *representations $\underline{M}$ and $\underline{N}$ are* DTop *with an exponential number of rules and the same number of states.*

Since the problem of DTop equivalence is NExpTime [14], a naïve approach to calculating the complexity of symbolic DTop equivalence would yield a 2NExpTime algorithm, plus an exponential number of operations in Φ and $\mathcal{F}$. However, upon finer analysis, the complexity of DTop equivalence is tied to the height of a counter-example between two non-equivalent transducers. This height is, in the worst case scenario, exponential in the number of states in the studied DTop. Since representations do not create new states, the height of the counter-examples is unchanged, and the exponentials do not compound.

Theorem 25. *The equivalence problem for symbolic* DTop *is in* NExpTime*, plus, at worst, an exponential number of operations in Φ and $\mathcal{F}$.*

7 Extension to symbolic DTop^R

We want to extend our results from symbolic DTop to the wider class of symbolic DTop with regular lookahead (symbolic DTop^R). A symbolic DTop^R is a pair of a symbolic DTop and its symbolic Dtta lookahead, straight-forwardly extending classical DTop^R [9] to the symbolic case. Note that this is different from an inspection; a DTop^R, whether symbolic or not, has access to the lookahead states in its rules, while a DTop with inspection does not. This means that if (M, A) is a symbolic DTop^R describing a transformation of $\mathcal{T}_\Sigma$, and A has a set of states Q, then M works on an input signature $\Sigma \times Q$.

The class of symbolic DTop^R is relevant for several reasons. The first is that it is more expressive than the class of symbolic DTop with inspection, and subsumes other relevant classes, such as single-valued symbolic DTop [21, 20]. It also possesses interesting properties. Notably, just as it was the case for DTop^R, the class of symbolic DTop^R is closed under composition.

We want to study the equivalence problem of symbolic DTop^R. For DTop, the addition of a regular lookahead does not prevent the equivalence problem from being decidable, as it is polynomially reduced to equivalence on plain DTop (see [18]). This result can be carried over to the symbolic case, with the same method: annotating the input trees of both symbolic DTop with the states of both lookaheads.

Lemma 26. *One can polynomially reduce the equivalence problem of symbolic* DTop^R *to the equivalence problem of symbolic* DTop *with inspection.*

Proof. We follow the same method as in the DTop case: if $N = (M, A)$ and $N' = (M', A')$ are two symbolic DTop^R. we want to annotate the trees with the information of both A and A', then test the equivalence of M and M' on those trees. The first step is to ensure that the annotation is the combination of the annotation by A and A'. This can be done by creating a new lookahead A'' that is the cartesian product of A and A'.

As a technical note, the domain of all trees annotated by A'' is regular, but not necessarily recognized by a Dtta. This is a problem, since the equivalence test we produce in this paper for symbolic DTop (as well as the preexisting equivalence test for DTop) requires a domain recognized by a Dtta. This issue is easily fixed by changing the way A'' annotates the input trees: if the information stored at each node is not only the state that A'' reaches but also the rule it uses to do so, then the domain of all annotated trees becomes recognizable by a Dtta.

Finally, testing the equivalence of N and N' is exactly testing the equivalence of M and M' on all trees annotated by A''; note that restricting a symbolic DTop M to the domain of a Dtta A'' is a simple matter of taking the cartesian product between the states of M and A''. □

We can note that all the steps of this reduction (computing the product of both lookaheads, switching from state to rule annotations, adapting M and M' to work on the proper domain) are polynomial, which makes the reduction from symbolic DTop^R equivalence to symbolic DTop equivalence polynomial. This reduction can be combined with our previous results (Cor. 23 and Thm. 25) to provide the following complexity results:

Corollary 27. *Under the equivalence-testing assumptions above, the equivalence problem for symbolic* DTop^R *is reducible to the equivalence problem on* DTop*, in* ExpTime *in the worst case, plus, at worst, an exponential number of operations in* Φ *and* $\mathcal{F}$*.*

Theorem 28. *Under the equivalence-testing assumptions, the equivalence problem for symbolic* DTop^R *is decidable in* NExpTime*, plus, at worst, an exponential number of operations in* Φ *and* $\mathcal{F}$*.*

This result is quite useful, as several DTop classes are fragments of the class of DTop^R. Notably, any DBup, nondeterministic functional Top, or extended DTop can be constructively expressed as DTop^R. Most of those inclusions stand in the symbolic case. The notable exception is extended symbolic DTop: the fact that such transducers can do multi-level pattern matching allows them to manipulate input labels in a way symbolic DTop^R cannot (e.g. switching the labels of parent and child nodes). However, symbolic DBup and nondeterministic symbolic functional Top can be expressed as symbolic DTop^R.

Corollary 29. *Under the equivalence-testing assumptions, the equivalence problem for deterministic symbolic bottom-up tree transducers and nondeterministic symbolic functional top-down tree transducers is decidable.*

8 Conclusion

The algorithm presented here provides a novel approach to deciding equivalence for symbolic DTop, and supports non-linear symbolic DTop, by reduction to DTop equivalence. Note that decidability of equivalence for DTop^R [18] works in a comparable way: rather than finding a normal form, the two regular lookaheads are "harmonized" into one, then the problem is reduced to DTop equivalence. The methods presented in this paper also apply to symbolic DTop^R without a critical jump in complexity.

Our method does not involve the computation of a normal form, which is a rather classical technique to decide transducer equivalence [4, 12], with applications to learning. It is interesting to see if normal forms could be defined for symbolic DTop. This looks challenging, however, as it seems more general than finding normal forms for DTop^R, which remains an open problem.

A first possible extension of our model would be to allow the lookahead to have registers, i.e. to memorize some data from the bottom of the tree to annotate the upper part of the tree with it. Under reasonable restrictions, it is likely that we might adapt our methods to reduce the equivalence problem for these objects to the same problem on DTop^R, thus providing a decidability result.

Furthermore, we would like to find out whether this kind of reduction can be applied to more general classes of transducers such as macro tree transducers (with linear size increase), for which equivalence is decidable [11]. If so, then the decidability results can fairly easily be lifted to symbolic generalisations of the class, at the cost of a few exponential blowups in complexity.

As a final mention, the inversion problem is interesting for symbolic transformations on words and trees, and is relevant to the applications we consider.

References

1. B. S. Baker. Composition of top-down and bottom-up tree transductions. *Information and Control*, 41(2):186–213, 1979.
2. A. Boiret. *Normalization and Learning of Transducers on Trees and Words.* PhD thesis, Lille University, France, 2016.
3. M. Bojańczyk. Transducers with origin information. In *ICALP 2014*, volume 8573 of *LNCS*, pages 26–37. Springer, 2014.
4. C. Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003.
5. 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.
6. C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, Jan. 1965.
7. J. Engelfriet. Bottom-up and top-down tree transformations - A comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975.
8. J. Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical systems theory*, 10(1):289–303, 1976.
9. J. Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.
10. J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. pages 241–286. Academic Press, 1980.
11. J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM J. Comput.*, 32(4):950–1006, 2003.
12. J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *J. Comput. Syst. Sci.*, 75(5):271–286, 2009.
13. J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985.
14. Z. Ésik. On functional tree transducers. In *FCT*, pages 121–127, 1979.
15. Z. Ésik. Decidability results concerning tree transducers II. *Acta Cybern.*, 6(3):303–314, 1983.
16. Z. Fülöp and H. Vogler. Forward and backward application of symbolic tree transducers. *Acta Inf.*, 51(5):297–325, 2014.
17. A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for top-down XML transformations. In *PODS*, pages 285–296. ACM, 2010.
18. S. Maneth. Equivalence problems for tree transducers: A brief survey. In Z. Ésik and Z. Fülöp, editors, *AFL*, volume 151 of *EPTCS*, pages 74–93, 2014.
19. H. Seidl, S. Maneth, and G. Kemper. Equivalence of deterministic top-down tree-to-string transducers is decidable. In *FOCS*, pages 943–962, 2015.

20. M. Veanes and N. Bjørner. Foundations of finite symbolic tree transducers. *EATCS*, 105:141–173, 2011.
21. M. Veanes and N. Bjørner. Symbolic tree transducers. In *PSI*, volume 7162 of *LNCS*, pages 377–393. Springer, 2011.