Nested Pebbles and Transitive Closure

Joost Engelfriet and Hendrik Jan Hoogeboom

Leiden University, Institute of Advanced Computer Science,
P.O.Box 9512, 2300 RA Leiden, The Netherlands

Abstract. First-order logic with k-ary deterministic transitive closure has the same power as two-way k-head deterministic automata that use a finite set of nested pebbles. This result is valid for strings, ranked trees, and in general for families of graphs having a fixed automaton that can be used to traverse the nodes of each of the graphs in the family. Other examples of such families are grids, toruses, and rectangular mazes.

1 Introduction

The complexity class $\mathsf{DSPACE}(\log n)$ of string languages accepted in logarithmic space by deterministic Turing machines, has two well-known distinct characterizations. The first one is in terms of deterministic two-way automata with several heads working on the input tape (and no additional storage). Second, Immerman [20] showed that these languages can be specified using first-order logic with an additional deterministic transitive closure operator – it is one of the main results in the field of descriptive complexity [11, 21]. Similar characterizations of $\mathsf{NSPACE}(\log n)$ hold for their nondeterministic counterparts.

The two characterizations each have a natural parameter indicating the relative complexity of the mechanism used. For multi-head automata the parameter is the number of heads used to scan the input. It is known that $k + 1$ heads are better than k, even for a single-letter input alphabet [26]. For transitive closure logics, the parameter is the arity of the transitive closure operators used. It seems to be open whether $(k + 1)$-ary transitive closure is more powerful than k-ary transitive closure.

Bargury and Makowsky [2] characterize k-head automata by a "k-regular" subset of first-order logic with k-ary transitive closure but their characterization only works in the nondeterministic case: "the modification of the k-regular formulas needed to take out the nondeterminism will spoil their elegant form, and we do not pursue this further".

Here we set out from the other side and present an automata-theoretic characterization of first-order logic with deterministic k-ary transitive closure. Our deterministic two-way automaton model has k heads, as expected, but is augmented with the possibility to put an arbitrary finite number of pebbles on its input tape, to mark positions for further use. If these pebbles can be used at will it is folklore that we obtain again $\mathsf{DSPACE}(\log n)$, a family too large for our purpose. Instead we only allow pebbles that are used in a nested (or LIFO) fashion: all pebbles can be 'seen' by the automaton as usual, but only the last

[0] STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science (B. Durand, W. Thomas, eds.), LNCS vol. 3884, 477-488, 2006.

one dropped can be picked up [19, 13, 25, 16]. On the other hand our pebbles are more flexible than the usual ones: they can be 'retrieved from a distance', i.e., a pebble can be picked up even when no head is scanning its position.

Our equivalence result (Theorem 5) is stated and proved for ranked trees in general, of which strings are a special case. The automaton model is the deterministic tree-walking automaton (with nested pebbles) which generalizes two-way automata on strings. One consequence of the result is that the class of tree languages accepted by these automata is closed under complement [27].

In Section 3 we translate logical formulas into automata, following [13] and additionally using the technique of Sipser [32] to deterministically search a computation space. Section 4 considers the reverse. As in [2] we adapt Kleene's construction to obtain regular formulas from automata, thus getting rid of the states of the automaton, but we need to iterate that construction: once for each nested pebble. In Section 5 we discuss the main result for single-head tree-walking automata, which are relevant as a model of XML [25, 28, 23]. Finally, in Section 6 we show how to extend our results to more general graph-like structures, such as unranked trees (important for XML), grids (as in [2]; important for picture recognition [3, 18, 24]), toruses, and, for $k \geq 2$, mazes [4, 8].

Due to space limitations many examples, technical details, explanations, footnotes, and references had to be omitted. The reader may find them in [14].

2 Preliminaries

A *ranked alphabet* is a finite set Σ together with a mapping rank : $\Sigma \to \mathbb{N}$. Terms over Σ are recursively defined: if $\sigma \in \Sigma$ is of rank n, and $t_1, \ldots, t_n$ are terms, then $\sigma(t_1, \ldots, t_n)$ is a term. As usual, terms are visualized as *trees*, which are special labelled graphs; $\sigma(t_1, \ldots, t_n)$ as a tree which has a root labelled by σ and outgoing edges labelled by $1, \ldots, n$ leading to the roots of trees for $t_1, \ldots, t_n$. The root of subtree t_i has child number i; for the root of the full tree it is 0.

For $k \geq 1$, a k-head *tree-walking automaton* or *twa* is a finite-state device that moves its k heads from node to node along the edges of the input tree. It determines its next move based on its present state, and the label and child number of the nodes visited. Accordingly, it changes state and, for each of its heads, it stays at the node, or moves either up to the parent of the node, or down to a specified child. If the automaton has no next move, we say it *halts*. The *language accepted* by the k-head twa $\mathcal{A}$ is the set of all trees on which $\mathcal{A}$ has a computation starting with all its heads at the root of the tree in the initial state and halting in an accepting state, again at the root of the tree. The family of languages accepted by k-head (deterministic) twa is denoted by $\mathsf{NW}^k\mathsf{A}$ ($\mathsf{DW}^k\mathsf{A}$).

A twa is able to make a systematic search of the tree (a preorder traversal), even using a single head, as follows. When a node is reached for the first time (entering it from above) the automaton continues in the direction of the first child; when a leaf is reached, the automaton goes up again. If a node is reached from below, from a child, it goes down again, to the next child, if that exists;

otherwise it moves to the parent of the node. The search ends when the root is entered from its last child. This traversal underlies our basic constructions.

In both [29] and [30], as an example, the authors explicitly construct a deterministic 1-head twa that evaluates boolean trees, i.e., terms with binary operators 'and' and 'or' and constants 0 and 1.

Strings form a special case. Tree-walking automata on monadic trees (each symbol has rank one except a special symbol with rank zero) are equivalent to the usual two-way automata on strings.

For an overview of the theory of first-order and monadic second-order logic on strings and trees in relation to formal language theory, see [34]. Here we consider *first-order* logic, describing properties of trees. The logic has node variables $x, y, \ldots$, which for a given tree range over its nodes. There are four types of atomic formulas over Σ: $\mathrm{lab}_\sigma(x)$, for every $\sigma \in \Sigma$, meaning that x has label σ; $\mathrm{edg}_i(x, y)$, for every i at most the rank of a symbol in Σ, meaning that the i-th child of x is y; $x \leq y$, meaning that x is an ancestor of y; and $x = y$. Formulas are built using the connectives $\neg$, $\wedge$, and $\vee$, and quantifiers $\exists$ and $\forall$ as usual.

If t is a tree with nodes $u_1, \ldots, u_n$, and ϕ is a formula such that its free variables are $x_1, \ldots, x_n$, then we write $t \models \phi(u_1, \ldots, u_n)$ if formula ϕ holds for t where the free x_i are valuated as u_i.

For fixed $k \geq 1$, by overlined symbols like $\overline{x}$ we denote k-tuples of objects of the type referred to by x, like logical variables, nodes in a tree, or pebbles.

Let $\phi(\overline{x}, \overline{y})$ be a formula where $\overline{x}, \overline{y}$ are distinct k-tuples of variables occurring free in ϕ. We use $\phi^*(\overline{x}, \overline{y})$ to denote the *k-ary transitive closure* of ϕ with respect to $\overline{x}, \overline{y}$. Informally, $\phi^*(\overline{x}, \overline{y})$ means that we can make a series of jumps from nodes $\overline{x}$ to nodes $\overline{y}$ such that each pair of consecutive k-tuples $\overline{x}', \overline{y}'$ connected by a jump satisfies $\phi(\overline{x}', \overline{y}')$. The formula ϕ may have additional free variables.

A predicate $\phi(\overline{x}, \overline{y})$ with free variables $\overline{x}, \overline{y}$ is *functional* (in $\overline{x}, \overline{y}$) if for every tree t and k-tuple of nodes $\overline{u}$ there is at most one k-tuple $\overline{v}$ such that $t \models \phi(\overline{u}, \overline{v})$. If ϕ has more free variables than $\overline{x}, \overline{y}$, this should hold for each fixed valuation of those variables. The transitive closure $\phi^*(\overline{x}, \overline{y})$ is *deterministic* if ϕ is functional (in the variables with respect to which the transitive closure is taken).

The *tree language* defined by a closed formula ϕ consists of all trees t such that $t \models \phi$. The family of all tree languages that are first-order definable is denoted by FO; if one additionally allows k-ary (deterministic) transitive closure we have the family $\mathsf{FO{+}TC}^k$ ($\mathsf{FO{+}DTC}^k$). For strings, general (deterministic) transitive closure (i.e., over unbounded values of k) characterizes the complexity class $\mathsf{NSPACE}(\log n)$ ($\mathsf{DSPACE}(\log n)$), see [11, 21].

3 From Logic to Nested Pebbles

A k-head tree-walking automaton with *nested pebbles* is a k-head twa that is additionally equipped with a finite set of pebbles. During the computation it may drop these pebbles (one by one) on nodes visited by its heads, to mark specific positions. It may test the currently visited nodes to see which pebbles are present. Moreover, it may retrieve a pebble from anywhere in the tree, provided

the life times of the pebbles are nested (which means that only the last one dropped can be retrieved). This can be formalized by keeping a (bounded) stack in the configuration of the automaton, pushing and popping pebbles when they are dropped and retrieved. Pebbles can be reused any number of times (but there is only one copy of each pebble). Computations should start and end with all heads at the root without pebbles on the input tree. The family of tree languages accepted by (deterministic) k-head twa with nested pebbles is denoted by $\mathsf{NPW}^k\mathsf{A}$ ($\mathsf{DPW}^k\mathsf{A}$).

Note that pebbles (1) are nested, as in e.g., [19, 13, 25, 16]; without this restriction again the classes $\mathsf{DSPACE}(\log n)$ and $\mathsf{NSPACE}(\log n)$ are obtained, (2) behave as *pointers*: we can store the address of a node when we visit it, and we can later whipe the address from memory without returning to the node itself ("abstract markers" [3] as opposed to the usual "physical markers"), (3) always remain visible to the automaton (not only the last one dropped, as in [19]).

Example 1. As in [6], consider a ranked alphabet with one binary symbol and two nullary symbols a and b, and consider the trees for which the path to each a-labelled leaf contains an even number of nodes on the 'branching structure' of the tree, i.e., nodes for which both the left and right subtree contain an a-labelled leaf. This is a first-order definable tree language that cannot be accepted by any single-head nondeterministic twa (without pebbles) [6].

However, it can be accepted by a (single-head) deterministic twa with two nested pebbles as follows. Using a preorder traversal, the first pebble is placed consecutively on a-labelled leaves. For each such leaf we follow the path upwards to the root counting the number of nodes that belong to the branching structure. To test whether a node belongs to that structure we place the second pebble on the node and test whether its other subtree, i.e., the one that does not contain the first pebble, contains an a-labelled leaf (using again a traversal of that subtree, the root of which can be recognized through the second pebble). □

We now generalize the inclusion $\mathsf{FO} \subseteq \mathsf{DPW}^1\mathsf{A}$ from [13], introducing k-ary transitive closure, as well as allowing k heads. Note that here we use 'pointer-like' pebbles, rather than the usual pebbles.

Lemma 2. *For ranked trees,* $\mathsf{FO{+}DTC}^k \subseteq \mathsf{DPW}^k\mathsf{A}$.

Proof. By induction on the structure of the formula ϕ we construct an automaton $\mathcal{A}$ that always halts on its input tree t. Generally speaking, each variable of ϕ acts as a pebble for $\mathcal{A}$. In case of k-ary transitive closure we need $3k$ pebbles to test the formula. Most features can be simulated using a single head, moving pebbles around, only for transitive closure we need all the k heads.

For intermediate formulas with free variables we fix the valuation of these variables by putting pebbles on the tree, one for each variable, and $\mathcal{A}$ should evaluate the formula according to this valuation; it may test these pebbles but is not allowed to retrieve them. Automaton $\mathcal{A}$ is started in the initial state with all heads at the root of the tree t, it may use additional pebbles (in a nested fashion), and it should halt again with all heads at the root.

For the atomic formulas (single-head) automata are easily constructed. As an example, for $\mathrm{edg}_i(x, y)$ the automaton searches for pebble x, determines whether x has an i-th child (the arity of the node can be seen from its label), moves to that child, and checks there whether pebble y is present.

For the negation $\phi = \neg\phi_1$ we use the automaton for ϕ_1, changing its accepting states to the complementary set. This works thanks to the fact that the automata we build are always halting. A similar argument works for conjunction and disjunction, running the automata for the two constituents consecutively.

For quantification $\phi = (\forall x)\phi_1$ the automaton $\mathcal{A}$ makes a systematic traversal through the tree, using a single head. Reaching a node it drops a pebble x, returns to the root, and runs the automaton for ϕ_1 as a subroutine; the free variable x of ϕ_1 is marked by the pebble, as requested by the inductive hypothesis. When the test for $\phi_1(x)$ is positive, $\mathcal{A}$ returns to the node marked x (searching for it), picks up the pebble, and places it on the next node of the traversal; $\mathcal{A}$ accepts if it has succesfully run the test for ϕ_1 for each node. Existential quantification is treated similarly.

For transitive closure $\phi = \phi_1^*$ we need to walk from one k-tuple of nodes $\overline{x}$ to another k-tuple $\overline{y}$ with 'jumps' specified by the $2k$-ary formula ϕ_1. Doing this in a straightforward way, we might end 'jumping around' in a cycle. To obtain an automaton that always halts we use the technique of Sipser [32], and run this walk backwards. It is based on the observation that the computation space is actually a tree. Consider all k-tuples of nodes of the input tree t, and connect vertex[1] $\overline{u}$ to vertex $\overline{v}$ if the pair $(\overline{u}, \overline{v})$ in t satisfies $\phi_1(\overline{x}, \overline{y})$. As ϕ_1 is functional, for each vertex there is at most one outgoing arc. Choosing vertex $\overline{y}$ as root we obtain a directed tree $t_k(\overline{y})$, with arcs defined by ϕ_1 pointing towards the root $\overline{y}$; there is no bound on the number of arcs incident to each vertex. It consists of all vertices $\overline{u}$ that satisfy $t \models \phi(\overline{u}, \overline{y})$.

The automaton $\mathcal{A}$ traverses that tree $t_k(\overline{y})$ and tries to find the vertex marked by pebbles $\overline{x}$. However, $t_k(\overline{y})$ is not explicitly available and has to be reconstructed while walking on the input tree t, using the automaton $\mathcal{A}_1$ for ϕ_1 as a subroutine. Note that k-tuples of nodes of t can be enumerated (ordered) using the lexicographical ordering based on the preorder in t. To find the successor of a k-tuple $\overline{z}$ we act like adding one to a k-ary number: change the last coordinate of the tuple $\overline{z}$ into its successor (here the preorder successor in t) if that exists, otherwise reset that coordinate to the first element in the ordering (here the root of t), and consider the last-but-one coordinate, etc. In fact, this can be done by a single-head twa using the pebbles marking $\overline{z}$ in a nested fashion.

We traverse the tree $t_k(\overline{y})$, with $2k$ pebbles $\overline{x}$ and $\overline{y}$ fixed, with the help of $3k$ additional pebbles $\overline{x}'$, $\overline{y}'$, and $\overline{z}'$. Starting in $\overline{y}$ we determine whether vertex $\overline{x}$ belongs to $t_k(\overline{y})$. During this traversal, $\mathcal{A}$ keeps track of the current vertex of $t_k(\overline{y})$ with its k heads. The order of dropping the pebbles $\overline{x}'$ and $\overline{y}'$ differs in the two algorithmic steps below: in the first we have to check $\phi_1(\overline{x}', \overline{y}')$ 'backwards', finding $\overline{x}'$ given $\overline{y}'$, while in the second it is the other way around.

[1] For clarity we distinguish 'node' in the input tree from 'vertex' in the computation space, i.e., a k-tuple of nodes. Similarly we use 'edge' and 'arc'.

Step one: check whether the current vertex has a first child in $t_k(\overline{y})$, and go there if it exists. We drop pebbles $\overline{y}'$ to fix the current vertex, and we 'lexicographically' place pebbles $\overline{x}'$ on each candidate vertex (except $\overline{v}$). For each k-tuple $\overline{x}'$ we check $\phi_1(\overline{x}', \overline{y}')$ using automaton $\mathcal{A}_1$ as a subroutine. If the formula is true, we have found the first child in $t_k(\overline{y})$ and we move the k heads to the nodes marked by $\overline{x}'$, lift pebbles $\overline{x}'$, and retrieve pebbles $\overline{y}'$ (from a distance). Otherwise we move $\overline{x}'$ to the next candidate vertex. If none of the candidates $\overline{x}'$ satisfies $\phi_1(\overline{x}', \overline{y}')$, the vertex $\overline{y}'$ apparently has no child in $t_k(\overline{y})$.

Step two: check for a right sibling in $t_k(\overline{y})$, and go there if it exists, or go up (to the parent of the current vertex) otherwise. The problem here is to adhere to the proper nesting of the pebbles. First drop pebbles $\overline{x}'$ on the current vertex. Then determine its parent in $t_k(\overline{y})$; this is the unique vertex $\overline{y}'$ that satisfies $\phi_1(\overline{x}', \overline{y}')$, thanks to the functionality of ϕ_1. It can be found in a 'lexicographic' traversal of all k-tuples of nodes of t using pebbles $\overline{y}'$ and subroutine $\mathcal{A}_1$. Leave $\overline{y}'$ on the parent and return to $\overline{x}'$ (by searching for it in t). Using the third set of k pebbles $\overline{z}'$, traverse the k-tuples of nodes of t from $\overline{x}'$ onwards and try to find the next k-tuple that satisfies $\phi_1(\overline{z}', \overline{y}')$ when $\overline{z}'$ is dropped. If found, it is the right sibling of $\overline{x}'$; return there, lift $\overline{z}'$, and retrieve $\overline{y}'$ and $\overline{x}'$. Otherwise, the current vertex has no right sibling; go up in the tree $t_k(\overline{y})$, i.e., return to $\overline{y}'$, lift $\overline{y}'$, and retrieve $\overline{x}'$. □

4 From Nested Pebbles to Logic

The classical result of Kleene shows how to transform a finite-state automaton into a regular expression, which basically means that we have a way to dispose of the states of the automaton. It is observed in [2] that this technique can also be used to transform multi-head automata on grids into equivalent formulas with transitive closure: transitive closure may very well specify sequences of consecutive positions on the input, but has no direct means to store states. A similar technique is used here. As our model includes pebbles, this imposes an additional problem, which we solve by iterating the construction for each pebble. Unlike [2] we have managed to find a formulation that works well for both the nondeterministic and deterministic case.

If the step relation of a deterministic finite-state device with k heads is specified by logical formulas, then its computation relation can be expressed using k-ary deterministic transitive closure. This is formalized as follows.

Let Φ be a $Q \times Q$ matrix of predicates $\phi_{p,q}(\overline{x}, \overline{y})$, $p, q \in Q$ for some finite set Q (of states), where $\overline{x}, \overline{y}$ each are k distinct variables occurring free in all $\phi_{p,q}$. We define the *computation closure* of Φ with respect to $\overline{x}, \overline{y}$ as the matrix $\Phi^{\#}$ consisting of predicates $\phi^{\#}_{p,q}(\overline{x}, \overline{y})$ where $t \models \phi^{\#}_{p,q}(\overline{u}, \overline{v})$ iff there exists a sequence of k-tuples of nodes $\overline{u}_0, \overline{u}_1, \ldots, \overline{u}_n$ and a sequence of states $p_0, p_1, \ldots, p_n$, $n \geq 1$, such that $\overline{u} = \overline{u}_0$, $\overline{v} = \overline{u}_n$, $p = p_0$, $q = p_n$, and $t \models \phi_{p_i,p_{i+1}}(\overline{u}_i, \overline{u}_{i+1})$ for $0 \leq i < n$.([2]) Intuitively $t \models \phi^{\#}_{p,q}(\overline{u}, \overline{v})$ means that there is a Φ-path of consecutive steps (as specified by Φ) leading from nodes $\overline{u}$ in state p to nodes $\overline{v}$ in state q.

[2] For simplicity, we disregard the remaining free variables of the $\phi_{p,q}$ and $\phi^{\#}_{p,q}$.

We say that Φ is *deterministic* if its predicates are both functional and exclusive, i.e., for any $p, q, q' \in Q$ and $3k$ nodes $\overline{u}, \overline{v}, \overline{v}'$ of any tree t, if both $t \models \phi_{p,q}(\overline{u}, \overline{v})$ and $t \models \phi_{p,q'}(\overline{u}, \overline{v}')$ then $q = q'$ and $\overline{v} = \overline{v}'$.

Lemma 3. *If Φ is in* FO+DTC^k *and deterministic, then $\Phi^\#$ is in* FO+DTC^k.

Proof. Assume that $Q = \{1, 2, \ldots, m\}$. We construct matrices $\Phi^{(\ell)}$ of formulas $\phi^{(\ell)}_{p,q}$ in FO+TC^k which are defined as $\phi^\#_{p,q}$, except that the intermediate states $p_1, \ldots, p_{n-1}$ are chosen from $\{1, \ldots, \ell\}$. In particular, $\Phi^{(0)} = \Phi$, and $\Phi^{(m)} = \Phi^\#$. Inductively we obtain $\Phi^{(\ell+1)}$ as follows: $\phi^{(\ell+1)}_{p,q}(\overline{x}, \overline{y}) = \phi^{(\ell)}_{p,q}(\overline{x}, \overline{y}) \vee (\exists \overline{x}' \, \overline{y}')[\, \phi^{(\ell)}_{p,\ell+1}(\overline{x}, \overline{x}') \wedge (\phi^{(\ell)}_{\ell+1,\ell+1})^*(\overline{x}', \overline{y}') \wedge \phi^{(\ell)}_{\ell+1,q}(\overline{y}', \overline{y}) \,]$. The transitive closure is deterministic: $\phi^{(\ell)}_{\ell+1,\ell+1}$ is functional because Φ is deterministic and because Φ-paths ending in $\ell+1$ cannot be extended following the definition of $\Phi^{(\ell)}$. □

Lemma 4. *For ranked trees,* $\mathsf{DPW}^k\mathsf{A} \subseteq \mathsf{FO{+}DTC}^k$.

Proof. Consider a k-head twa $\mathcal{A}$ with n pebbles $x_n, \ldots, x_1$, used in the order given, i.e., x_n is always placed on the bottom of the pebble stack. View $\mathcal{A}$ as consisting of $n+1$ 'levels' $\mathcal{A}_n, \ldots, \mathcal{A}_1, \mathcal{A}_0$ such that $\mathcal{A}_\ell$ is a k-head twa with ℓ pebbles $x_\ell, \ldots, x_1$, available for dropping and retrieving, whereas pebbles $x_n, \ldots, x_{\ell+1}$ have a fixed position on the tree and $\mathcal{A}_\ell$ may test for their presence. Basically, $\mathcal{A}_\ell$ acts as a twa that drops pebble x_ℓ, then queries $\mathcal{A}_{\ell-1}$ where to go in the tree, moves there, and retrieves pebble x_ℓ (from a distance).

The number of pebbles dropped can be kept in the finite control of $\mathcal{A}$, so we can unambiguously partition its state set as $Q = Q_n \cup \cdots \cup Q_1 \cup Q_0$, where Q_ℓ consists of states where ℓ pebbles are still available. Automaton $\mathcal{A}_\ell$ is the restriction of $\mathcal{A}$ to the states in Q_ℓ.

For $\mathcal{A}_\ell$ a matrix $\Phi^{(\ell)}$ is constructed with predicates $\phi^{(\ell)}_{p,q}$ for $p, q \in Q_\ell$. These predicates represent the single steps of $\mathcal{A}_\ell$, so $t \models \phi^{(\ell)\#}_{p,q}(\overline{u}, \overline{v})$ iff $\mathcal{A}_\ell$ has a nonempty computation from configuration $[p, \overline{u}]$ to configuration $[q, \overline{v}]$. Note that $\Phi^{(\ell)}$ has additional free variables $x_n, \ldots, x_{\ell+1}$ that will hold the positions of the pebbles already placed on the tree.

First assume that pebble x_ℓ has not been dropped. For each of its heads, $\mathcal{A}_\ell$ may test the presence of pebbles $x_n, \ldots, x_{\ell+1}$, and the node label and child number of the current node, and then it may move each of its heads. These steps, relations between the current and next configurations $[p, \overline{u}]$ and $[q, \overline{v}]$, are easily expressed in first-order logic. E.g., if the automaton can move head 5 to the first child of a node with label σ, while the node under head 6 has child number 2 and does not contain pebble $x_{\ell+1}$, then $\phi^{(\ell)}_{p,q}(\overline{u}, \overline{v})$ is the conjunction of the formulas $\mathrm{lab}_\sigma(u[5])$, $(\exists u')\, \mathrm{edg}_2(u', u[6])$, $u[6] \neq x_{\ell+1}$, $\mathrm{edg}_1(u[5], v[5])$, and $u[j] = v[j]$ for $j \neq 5$ (where $u[i]$ denotes the ith component of $\overline{u}$).

Additionally when $\ell \geq 1$, $\mathcal{A}_\ell$ may drop pebble x_ℓ at the position of head i in state p, call $\mathcal{A}_{\ell-1}$, and retrieve pebble x_ℓ returning to state q. Such a 'macro step' from configuration $[p, \overline{u}]$ to $[q, \overline{v}]$ is only possible when there is a pair of pebble instructions $(p, \mathrm{drop}_i(x_\ell), p')$ and $(q', \mathrm{retrieve}(x_\ell), q)$, such that $\mathcal{A}_{\ell-1}$ has

a (nonempty) computation from $[p', \overline{u}]$ to $[q', \overline{v}]$, i.e., $t \models \phi^{(\ell-1)\#}_{p',q'}(\overline{u}, \overline{v})$. ([3]) Hence, $\mathcal{A}_\ell$ can take that step iff the disjunction of $\phi^{(\ell-1)\#}_{p',q'}(\overline{u}, \overline{v})$ over all such q' holds, where the free variable x_ℓ in that formula is replaced by $u[i]$, the position at which the pebble is dropped.

The resulting step matrix $\Phi^{(\ell)}$ is deterministic thanks to the determinism of $\mathcal{A}$ and $\Phi^{(\ell-1)}$. It is in FO+DTC^k by Lemma 3. The computational behaviour of $\mathcal{A}_\ell$ is expressed by $\Phi^{(\ell)\#}$, and that of $\mathcal{A}$ by the disjunction of all formulas $\phi^{(n)\#}_{p,q}(\overline{\mathrm{root}}, \overline{\mathrm{root}})$ with p initial and q accepting. □

Combining Lemmas 2 and 4, we immediately get the main result of this paper. Note that it includes the case of strings.

Theorem 5. *For ranked trees,* $\mathsf{DPW}^k\mathsf{A} = \mathsf{FO{+}DTC}^k$.

As a corollary we may transfer two obvious properties of $\mathsf{FO{+}DTC}^k$, closure under complement and union, to deterministic twa with nested pebbles, where the result is nontrivial. In the proof of Lemma 2 we have constructed automata that are always halting. As all our constructions are effective this means that 'always-halting' is a normal form for deterministic twa with nested pebbles. In fact, the two closure properties follow rather directly from this normal form. This is further studied with regard to the number of pebbles needed in [27].

When the twa is not deterministic we no longer can assure the determinism of the formulas $\Phi^{(\ell)}$ in the proof of Lemma 4. However, they are in FO+TC^k. The proof of Lemma 4 uses negation only on atomic predicates, to model negative tests of the automaton (to check there is no specific pebble on a node). Since negation is not used in the proof of Lemma 3, we obtain *positive formulas*, allowing transitive closure only within the scope of an even number of negations (see, e.g., [11, 21]).

Conversely, for positive formulas there is also a result similar to Lemma 2. For disjunction and existential quantification the automaton now uses nondeterminism in the obvious way. For transitive closure $\phi = \phi_1^*$ the Sipser technique is not needed: $\mathcal{A}$ checks nondeterministically the existence of a path from vertex $\overline{x}$ to vertex $\overline{y}$ in the directed graph determined by ϕ_1.

Denoting the positive restriction of $\mathsf{FO{+}TC}^k$ by $\mathsf{FO{+}posTC}^k$, we thus obtain a characterization for the nondeterministic case. We do not know whether $\mathsf{NPW}^k\mathsf{A}$ is closed under complement (i.e., whether 'pos' can be dropped from this result).

Theorem 6. *For ranked trees,* $\mathsf{NPW}^k\mathsf{A} = \mathsf{FO{+}posTC}^k$.

5 Single Head on Trees

Single-head tree-walking automata (with output) were introduced as a device for syntax-directed translation [1] (see [17]). Quite recently they came into fashion again as a model for translation of XML specifications [25, 28, 23, 7].

[3] Here we assume that instructions dropping and retrieving pebbles have no tests.

The control of a single-head tree-walking automaton is at a single node of the input tree, i.e., sequential. Thus it differs from the classic bottom-up/top-down tree automata, which are inherently parallel in the sense that the control is fused/split for every branching of the tree.

The power of the classic model is well known: it accepts the regular tree languages. For twa however, the situation was unclear for a long time. They recognize regular tree languages only, but it was conjectured in [12] (and later in [15, 13, 7]) that they cannot recognize them all. Recently this has been proved for deterministic and nondeterministic twa in [5] and [6], respectively.

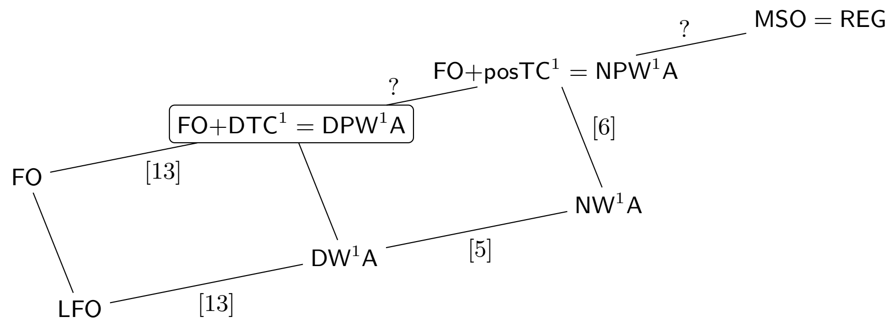
To strengthen the power of the single-head twa, keeping its sequential nature, in [13] the single-head twa was equipped with nested pebbles. We showed there that the tree languages accepted by such twa are still regular.

As observed before, $\mathsf{DSPACE}(\log n)$ is the class of languages accepted by single-head two-way automata with (nonnested) pebbles. Thus, for $k = 1$ (single-head automata vs. unary transitive closure), our main characterization for tree languages, Theorem 5, can be seen as a 'regular' restriction of the result of Immerman characterizing $\mathsf{DSPACE}(\log n)$; on the one hand only automata with *nested* pebbles are allowed, while on the other hand we consider only *unary* transitive closure, i.e., transitive closure for $\phi(x, y)$ where x, y are single variables. Note that unary transitive closure can be simulated in monadic second-order logic (MSO), which defines the family REG of regular tree languages [10, 33].

In the diagram we compare the family $\mathsf{FO{+}DTC}^1 = \mathsf{DPW}^1\mathsf{A}$ with several next of kin. Lines without question mark denote proper inclusion. By LFO we denote the family of languages definable in *local* first-order logic, i.e., dropping the atomic formula $x \leq y$. The regular language $(aa)^*$ shows that $\mathsf{DW}^1\mathsf{A} \not\subseteq \mathsf{FO}$.

Consider the binary trees that among their leaves have (exactly) three positions marked by a special symbol a in such a way that there is an internal node, the left subtree of which contains a single a, while its right subtree contains the other two. This example from [5] shows that $\mathsf{DW}^1\mathsf{A} \subset \mathsf{NW}^1\mathsf{A}$ and moreover that $\mathsf{FO} \not\subseteq \mathsf{DW}^1\mathsf{A}$.

The example to prove $\mathsf{REG} \not\subseteq \mathsf{NW}^1\mathsf{A}$ [6] shows even that $\mathsf{FO} \not\subseteq \mathsf{NW}^1\mathsf{A}$, cf. Example 1. Logical characterizations of $\mathsf{DW}^1\mathsf{A}$ and $\mathsf{NW}^1\mathsf{A}$ are given in [29], using transitive closure in a restricted way. In [31] several logics for regular



tree languages are studied; it is stated as an open problem whether all regular tree languages can be defined using monadic transitive closure, i.e., whether $\mathsf{FO{+}TC}^1 = \mathsf{REG}$.

Some of the inclusions between the families of trees we have studied are not known to be strict. In particular, can single-head twa with pebbles recognize all regular tree languages? If, instead of with pebbles, they are equipped with a synchronized pushdown or, equivalently, with 'marbles', then they do recognize all regular tree languages [22, 15]. There are several other open questions. Is there a hierarchy for tree languages accepted by (deterministic) twa with respect to the number of pebbles these automata use? Are our 'pointer' pebbles more powerful than the usual 'physical' ones? For nonnested pebbles the two types have the same power, even when the number of pebbles is fixed [3].

6 Walking on Graphs

We generalize our results on trees (and strings) to more general families of graphs (with both node and edge labels). To have a meaningful notion of graph-walking automaton we only consider (connected) graphs with a natural locality condition: a node cannot have two incident edges with the same label and the same direction. Trees over a ranked alphabet fall under this definition since we label the edge from a parent to its i-th child by i. Unranked trees satisfy this condition when represented with 'first child' and 'next sibling' edges. Two-dimensional grids satisfy it by distinguishing between horizontal and vertical edges.

A k-head *graph-walking automaton with nested pebbles* is like its relative for trees, but it may additionally check whether one of its current nodes has an incident incoming/outgoing edge with a specific label (generalizing the concepts of child number and rank). Generally graphs do not have a distinguished node (like the root for trees); thus for acceptance of an input graph we require that the automaton has an accepting computation when started with all its heads on *any* node of the input graph. Not all automata satisfy this requirement.

The first-order logic for graphs over the label alphabet Σ has atomic formulas $\mathrm{lab}_\sigma(x)$, $\sigma \in \Sigma$, for a node x with label σ, $\mathrm{edg}_\sigma(x, y)$, $\sigma \in \Sigma$, for an edge from x to y with label σ, and $x = y$. We do not allow the predicate $x \leq y$, although for trees that can be defined in first-order logic with deterministic transitive closure.

For arbitrary families of graphs the computation of an automaton can be specified in logic, like in Section 4. (We keep the notation for the families.)

Lemma 7. *For every family of graphs,* $\mathsf{DPW}^k\mathsf{A} \subseteq \mathsf{FO{+}DTC}^k$.

The other direction holds for all families of graphs for which there exists a (fixed) single-head deterministic graph-walking automaton (with nested pebbles) that can traverse each graph of the family, visiting each node at least once. Such a family is called *searchable*, and the fixed automaton a *guide*.

Unranked (ordered) trees, without bound on the number of children of a node, are a searchable graph family in their representation as binary trees. The (single-head) automata in this representation may move to the first child or to

the next sibling of a node (and back), exactly as customary in the literature [28, 30] (albeit without pebbles). Rectangular (directed) grids, edges pointing to the right or downwards, with edge labels distinguishing these two types of edges, form another example of a searchable family. This can be generalized to higher dimensions [2].

Cyclic grids, or toruses, where the last node of each row has an edge to the first node of that row, and similarly for columns, can be searched using two pebbles. We search the grid row-by-row: the first pebble marks the position we start with (in order to stop when all rows are visited; this pebble is not moved), the second pebble moves down in the first column to mark the position in which we started the row (in order to stop when we finish the row; we then move the pebble down to the next row until we meet the first pebble).

Theorem 8. *For every searchable family of graphs,* $\mathsf{DPW}^k\mathsf{A} = \mathsf{FO}{+}\mathsf{DTC}^k$.

The family of all graphs is not searchable, not even with nonnested pebbles or with several heads. This follows from results of Cook and Rackoff [9].

It is open whether we can search a maze (a connected subgraph of a grid) with a single head using nested pebbles. However with two heads we can search a maze [4]. To cover this family we need to extend the notion of searchability: a family of graphs is *k-searchable* if there is a deterministic guide as before, now having k heads. We have to extend our automaton model with a *new instruction* that moves a given head to a given pebble (like the 'jumping' instruction from [9]). This is quite natural if we see pebbles as pointers, storing the address of a node. With this assumption we get a result as above for k-searchable families.

Some unresolved questions were stated in Section 5. Another question is whether our results can be generalized to alternating automata and the alternating transitive closure operator of [20].

References

1. A.V. Aho, J.D. Ullman. Translations on a context free grammar, Inform. Control. 19, 439–475, 1971.
2. Y. Bargury, J.A. Makowsky. The expressive power of transitive closure and 2-way multihead automata, Proceedings CSL '91, LNCS 626, 1–14, 1992.
3. M. Blum, C. Hewitt. Automata on a 2-dimensional tape, Proceedings 8th IEEE SWAT, 155–160, 1967.
4. M. Blum, D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs), Proceedings 19th FOCS, 132–142, 1978.
5. M. Bojańczyk, T. Colcombet. Tree-walking automata cannot be determinized, Proceedings ICALP 2004 (J. Diaz et al., eds.), LNCS 3142, 246–256, 2004.
6. M. Bojańczyk, T. Colcombet. Tree-walking automata do not recognize all regular languages, Proceedings 37th STOC (H.N. Gabow, R. Fagin, eds.), 2005.
7. A. Brüggemann-Klein, D. Wood. Caterpillars: A context specification technique, Markup Languages 2, 81–106, 2000.
8. L. Budach. Automata and labyrinths, Math. Nachr. 86, 195–282, 1978.
9. S.A. Cook, C.W. Rackoff. Space lower bounds for maze threadability on restricted machines, SIAM J. Comput. 9, 636–652, 1980.

10. J. Doner. Tree acceptors and some of their applications, J. Comp. Syst. Sci. 4, 406–451, 1970.
11. H.-D. Ebbinghaus, J. Flum. *Finite Model Theory*, second edition, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1999.
12. J. Engelfriet. Context-free grammars with storage, Leiden University, Technical Report 86–11, 1986.
13. J. Engelfriet, H.J. Hoogeboom. Tree-walking pebble automata, *Jewels are forever* (J. Karhumäki et al., eds.), Springer-Verlag, 72–83, 1999.
14. J. Engelfriet, H.J. Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. LIACS Tech. Rep. 2005-02, Leiden University, 2005.
15. J. Engelfriet, H.J. Hoogeboom, J.-P. van Best. Trips on trees, Acta Cyb. 14, 51–64, 1999.
16. J. Engelfriet, S. Maneth. A comparison of pebble tree transducers with macro tree transducers, Acta Inf. 39, 613–698, 2003.
17. J. Engelfriet, G. Rozenberg, G. Slutzki. Tree transducers, L systems, and two-way machines, J. Comp. Syst. Sci. 20, 150–202, 1980.
18. D. Giammarresi, A. Restivo. Two-dimensional languages, *Handbook of Formal Languages*, Vol. 3 (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, 1997.
19. N. Globerman, D. Harel. Complexity results for two-way and multi-pebble automata and their logics, TCS 169, 161–184, 1996.
20. N. Immerman. Languages that capture complexity classes, SIAM J. Comput. 16, 760–778, 1987.
21. N. Immerman. *Descriptive Complexity*, Springer-Verlag, New York, 1999.
22. T. Kamimura, G. Slutzki. Parallel and two-way automata on directed ordered acyclic graphs, Inform. Control. 49, 10–51, 1981.
23. N. Klarlund, T. Schwentick, D. Suciu. XML: Model, Schemas, Types, Logics, and Queries, *Logics for Emerging Applications of Databases* (J. Chomicki et al., eds.), Springer-Verlag, 1–41, 2004.
24. O. Matz, N. Schweikardt, W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs, Inform. Comput. 179, 356–383, 2002.
25. T. Milo, D. Suciu, V. Vianu. Typechecking for XML transformers, J. Comp. Syst. Sci. 66, 66–97, 2003.
26. B. Monien. Two-way multihead automata over a one-letter alphabet, RAIRO – ITA 14, 67–82, 1980.
27. A. Muscholl, M. Samuelides, L. Segoufin. Complementing deterministic tree-walking automata, Manuscript, 2005. To appear in IPL.
28. F. Neven. Automata, logic, and XML, Proceedings CSL 2002 (J.C. Bradfield, ed.), LNCS 2471, 2–26, 2002.
29. F. Neven, T. Schwentick. On the power of tree-walking automata, Inform. Comput. 183, 86–103, 2003.
30. A. Okhotin, K. Salomaa, M. Domaratzki. One-visit caterpillar tree automata, Fund. Inf. 52, 2002, 361–375.
31. A. Potthoff. Logische Klassifizierung regulärer Baumsprachen, PhD thesis, Institut für Informatik und Praktische Mathematik, Universität Kiel, 1994.
32. M. Sipser. Halting space-bounded computations, TCS 10, 335–338, 1980.
33. J.W. Thatcher, J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic, MST 2, 57–81, 1968.
34. W. Thomas. Languages, automata, and logic, *Handbook of Formal Languages*, Vol. 3 (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, 1997.