Most machine models one studies in theoretical computer science, like Turing machines and finite automata, work on inputs that are strings over some finite alphabet. However, often problem descriptions have inputs that more structured than strings. For example, the input could be a graph, or a tree, or a partial order, etc. We will now introduce an automaton model that computes on inputs that are trees.

Trees arise in a variety of contexts. Often strings semantically denote trees. Examples of this phenomena is shown in Figure 1. Arithmetic expressions or logical formulas maybe written as strings, but the scoping of the arithmetic and logical operators are best understood when one looks at the parse tree corresponding to the expression. Documents like XML and HTML are also textual representation of a tree, identified through the use of tags.

Trees are special kinds of graphs. Here we will consider rooted trees, i.e., trees where a special vertex has been designated as the root. There is a way to name vertices in a tree that makes explicit the parent/child and ancestor/descendent relation explicit. Recall that in a rooted tree, every vertex has a unique path from the root. This unique path can be taken to be the name of vertices. For example, consider the tree shown in Figure 1a. The root is labeled $\sqrt{}$; its child is 0 (labeled ·), and the other vertices are 00 and 01 (children of 0 and labeled $x$ and +, respectively), 010 (labeled $y$), and 011 (labeled 3). We formalize this naming scheme and define labeled trees.

**Definition 1.** An n-ary tree domain, dom_t, is a prefix closed subseteq of {0, 1, 2, ..., n − 1}∗ such that if $u_i \in \text{dom}_t$ then $u_j \in \text{dom}_t$ for every $j < i$.

A $\Gamma$-labeled n-ary tree is a pair $t = (\text{dom}_t, \text{val}_t)$, where dom_t is an n-ary tree domain and val_t : $t \rightarrow \Gamma$ is a labeling function.

**Example 2.** Let define each of the trees in Figure 1 formally, as a labeled tree. Tree in Figure 1a is a
\{\sqrt{\cdot}, +, x, y, 3\}-labeled tree \( t_1 = (\{\varepsilon, 0, 00, 01, 010, 011\}, \text{val}_{t_1}) \), where
\[
\begin{align*}
\text{val}_{t_1}(\varepsilon) &= \sqrt{\cdot} & \text{val}_{t_1}(0) &= 0 & \text{val}_{t_1}(00) &= x \\
\text{val}_{t_1}(01) &= + & \text{val}_{t_1}(010) &= y & \text{val}_{t_1}(011) &= 3
\end{align*}
\]
Similarly, tree shown in Figure 1b is \( t_2 = (\{\varepsilon, 0, 1, 10, 100, 101\}, \text{val}_{t_2}) \) with
\[
\begin{align*}
\text{val}_{t_2}(\varepsilon) &= \land & \text{val}_{t_2}(0) &= x & \text{val}_{t_2}(1) &= \lor \\
\text{val}_{t_2}(10) &= \lor & \text{val}_{t_2}(100) &= y & \text{val}_{t_2}(101) &= z
\end{align*}
\]

It is useful to introduce two operations on trees: the operation of building larger trees from smaller trees, and the operation of identifying subtrees of trees. Suppose \( t_0 \) and \( t_1 \) are \( \Gamma \)-labeled trees. Then, for any \( A \in \Gamma \), \( A(t_0, t_1) \) denotes tree with root labeled \( A \), with left child of the root being the tree \( t_0 \), and the right child of the tree being \( t_1 \). This can be generalized to a tree of the form \( A(t_0, t_1, \ldots, t_k) \) formed from \( k \) trees \( t_0 \ldots t_k \) with root labeled \( A \). For a vertex/position \( p \) in a tree \( t \), the subtree rooted at that vertex will be denoted by \( t|_p \). We define these two operations precisely.

**Definition 3.** Given \( \Gamma \)-labeled trees \( t_0 = (\text{dom}_{t_0}, \text{val}_{t_0}) \) and \( t_1 = (\text{dom}_{t_1}, \text{val}_{t_1}) \), and \( A \in \Gamma \), the tree \( A(t_0, t_1) \) is given by \( t = (\text{dom}_t, \text{val}_t) \) where
\[
\text{dom}_t = \{\varepsilon\} \cup \{0u \mid u \in \text{dom}_{t_0}\} \cup \{1u \mid u \in \text{dom}_{t_1}\}
\]
\[
\text{val}_t(u) = \begin{cases} 
A & \text{if } u = \varepsilon \\
\text{val}_{t_0}(v) & \text{if } u = 0v \\
\text{val}_{t_1}(v) & \text{if } u = 1v
\end{cases}
\]

The above construction can be naturally generalized to the tree \( A(t_0, \ldots, t_k) \), \( k \geq 0 \), formed from \( k + 1 \) trees \( t_0, \ldots, t_k \).

Given a \( \Gamma \)-labeled tree \( t = (\text{dom}_t, \text{val}_t) \) and vertex/position \( p \in \text{dom}_t \), subtree rooted at position \( p \), is the tree \( t|_p = (\text{dom}_{t|_p}, \text{val}_{t|_p}) \) given by
\[
\text{dom}_{t|_p} = \{u \mid pu \in \text{dom}_t\}
\]
\[
\text{val}_{t|_p}(u) = \text{val}_t(pu)
\]

Add example.

## 1 Deterministic Tree Automata

We will now define deterministic tree automata (DTA). Like a DFA, a DTA has finitely many transitions but it processes a tree. Intuitively, a DTA starts with some states at the leaves depending on the label of the leaves. At each step, based on the states of the children and the label of a node, the transition of the DTA determines the state of the node. When finally, the state of the root is determined, the input is accepted if this state is an accepting state; otherwise it is rejected. Let us define this automaton and its computation precisely.

**Definition 4.** A **deterministic tree automaton** (DTA) on \( \Sigma \)-labeled \( n \)-ary trees is \( M = (Q, \Sigma, \delta, \mathcal{F}) \) where \( Q \) is a finite set of states, \( \mathcal{F} \subseteq Q \) is a set of final/accepting states, and \( \delta = \bigcup_{i=0}^n \delta_i \) is the transition function, where \( \delta_i : Q^i \times \Sigma \to Q \).

We will now define the execution of a DTA \( M \) on a \( \Sigma \)-labeled tree \( t = (\text{dom}_t, \text{val}_t) \). To do that let us recall some terminology. Recall that as per our convention of tree domains, a vertex \( u \in \text{dom}_t \) has \( i \) children if \( uj \in \text{dom}_t \) for all \( j < i \) but \( ui \notin \text{dom}_t \). In particular, a vertex \( u \) has 0 children or is a leaf, if \( u0 \notin \text{dom}_t \). We are now ready to define a run of DTA, an accepting run of a DTA, and the language recognized by a DTA.
Definition 5. The run of a DTA \( M = (Q, \Sigma, \delta, \cup_{r=0}^{\infty}, F) \) on a tree \( t = (\text{dom}_t, \text{val}_t) \) is a Q-labeled tree \( \rho = (\text{dom}_\rho, \text{val}_\rho) \) where \( \text{dom}_\rho = \text{dom}_t \) and for any vertex \( u \in \text{dom}_t \) with \( i \) children,
\[
\text{val}_\rho(u) = \delta_i(\text{val}_\rho(u0), \ldots, \text{val}_\rho(u(i-1)), \text{val}_t(u)).
\]

A run \( \rho = (\text{dom}_\rho, \text{val}_\rho) \) of \( M \) on \( t \) is accepting if \( \text{val}_\rho(\varepsilon) \in F \). A tree \( t \) is accepted by \( M \) if \( M \) has an accepting run on \( t \). Finally the language recognized by \( M \) is the set of all \( \Sigma \)-labeled \( n \)-ary trees it accepts, i.e.,
\[
L(M) = \{ t \mid M \text{ accepts } t \}.
\]

Let us look at some examples to understand DTAs.

Example 6. Let \( \Sigma \) be the set \( \{0, 1, \neg, \wedge, \vee\} \). Consider the DTA \( M_\rho = (\{q_0, q_1, q_2\}, \Sigma, \delta, \{q_1\}) \) where the transition function is given as
\[
\begin{align*}
\delta_0(0) &= q_0 \\
\delta_1(q_0, \neg) &= q_1 \\
\delta_2(q_1, q_j, \wedge) &= \begin{cases} 
q_0 & \text{if } q_i = q_j = q_1 \\
q_1 & \text{if } \{q_i, q_j\} \cap \{q_1\} = \emptyset \\
q_2 & \text{if } \{q_i, q_j\} \cap \{q_1\} \neq \emptyset 
\end{cases} \\
\delta_0(1) &= q_1 \\
\delta_1(q_1, \neg) &= q_0 \\
\delta_2(q_1, q_j, \vee) &= \begin{cases} 
q_0 & \text{if } q_i = q_j = q_0 \\
q_1 & \text{if } \{q_i, q_j\} \cap \{q_1\} = \emptyset \\
q_2 & \text{if } \{q_i, q_j\} \cap \{q_1\} \neq \emptyset 
\end{cases}
\end{align*}
\]

For any of the cases not considered above \( M_\rho \) transitions to the state \( q_r \).

![Figure 2](image)

Figure 2: (a) Input tree \( \neg((0 \wedge 1) \vee (0 \wedge 0)) \) (left); (b) Run of DTA \( M_\rho \) on input \( \neg((0 \wedge 1) \vee (0 \wedge 0)) \) (right).

Recall that the string \( \neg((0 \wedge 1) \vee (0 \wedge 0)) \) represents the tree shown in Figure 2a. The run of \( M_\rho \) is shown in Figure 2b. Since the label of the root in the run is \( q_1 \), this input is accepted by \( M_\rho \).

Trees over the alphabet \( \Sigma \) can be thought of as boolean expressions, provided the labeling of the tree is consistent with the arity of the logical operators. If a tree represents a syntactically incorrect expression, like say a vertex labeled \( \neg \) is either leaf or has two children, then \( M_\rho \) on such an input has a run where the state labeling the root is \( q_r \). If the input tree corresponds to a syntactically correct boolean expression, then the state labeling the root in the run of \( M_\rho \) is \( q_0 \) if the boolean expression is false, and is \( q_1 \) if it is true. Therefore, the language recognized by \( M_\rho \) is the set of syntactically correct boolean expressions that evaluate to true.

Example 7. Consider the alphabet \( \Sigma = \{0, 1, +, \cdot\} \). Consider the DTA \( M_\alpha = (\{q_0, q_1, q_2, q_r\}, \Sigma, \delta, \{q_0\}) \) where
\[
\begin{align*}
\delta_0(0) &= q_0 \\
\delta_2(q_i, q_j, +) &= q_{i+j \mod 3} \text{ if } \{q_i, q_j\} \cap \{q_r\} = \emptyset \\
\delta_0(1) &= q_1 \\
\delta_2(q_i, q_j, \cdot) &= q_{i \cdot j \mod 3} \text{ if } \{q_i, q_j\} \cap \{q_r\} = \emptyset
\end{align*}
\]

In all other cases not considered above, \( \delta \) returns \( q_r \).

Trees over \( \Sigma \) represent arithmetic expressions. If they are syntactically incorrect, then \( M_\alpha \) will have a run whose root is labeled \( q_0 \). If the input is a syntactically correct arithmetic expression, \( M_\alpha \)’s run will have root labels \( q_i \), where \( i \) is the remainder when the value of the expression is divided by 3. Since the final state is \( q_0 \), the set of trees accepted by \( M_\alpha \) are all those representing syntactically correct arithmetic expressions that evaluate to a value that is a multiple of 3.
Let us look at some examples of designing DTAs to recognize languages of trees.

**Example 8.** Consider the alphabet $\Sigma = \{0, 1, g, f\}$. The collection of all trees containing an even number of $f$s can be recognized by a DTA as follows. The DTA will track the parity of the number of $f$s seen in the subtree read so far. So the DTA has two states $q_0$ and $q_1$, and it will be in state $q_i$ in subtree at position $p$ is of the number of $f$s in the subtree at position $p$ modulo 2 is $i$. This can be ensured by the following transition rules.

$$
\begin{align*}
\delta_0(0) &= \delta_0(1) = \delta_0(g) = q_0 & \delta_0(f) &= q_1 \\
\delta_1(q_i, 0) &= \delta_1(q_i, 1) &= \delta_1(q_i, g) = q_i & \delta_1(q_i, f) &= q_{i+1} \ mod \ 2 \\
\delta_2(q_i, q_j, 0) &= \delta_2(q_i, q_j, 1) &= \delta_2(q_i, q_j, g) = q_{i+j} \ mod \ 2 & \delta_2(q_i, q_j, f) &= q_{i+j+1} \ mod \ 2
\end{align*}
$$

Since we need to accept trees with even $f$s, the set of final states will be $\{q_0\}$.

**Example 9.** Consider the context-free grammar with rules $S \rightarrow AA$, $A \rightarrow a|c|AB$, and $B \rightarrow b$. The set of derivation trees/parse trees of this grammar can be recognized by a DTA. Intuitively, the DTA’s state after reading a subtree will track if the subtree is consistent with the grammar rules, and the state will record the nonterminal labeling the root of the subtree.

Formally, the states are $Q = \{q_a, q_b, q_c, q_A, q_B, q_C, q_s\}$, with $F = \{q_S\}$. Here the state $q_s$ is reached if the subtree is not consistent with the rules of the grammar, and it is one of the other states $q_X$ if it is consistent and the root of the subtree is labeled $X$. The transitions are as follows.

$$
\begin{align*}
\delta_0(a) &= q_a & \delta_0(b) &= q_b & \delta_0(c) &= q_c \\
\delta_1(q_a, A) &= q_A & \delta_1(q_c, A) &= q_A & \delta_1(q_b, B) &= q_B \\
\delta_1(q_A, q_B, A) &= q_A & \delta_2(q_A, q_A, S) &= q_S
\end{align*}
$$

All “other” transitions go to $q_s$.

# 2 Regularity and Nonregularity

As in the case of words, these automata define an interesting class of tree languages that we will call *regular*.

**Definition 10.** A set $A$ of $\Sigma$-labeled $n$-ary trees is a *regular* if there is a DTA $M$ such that $A = L(M)$.

Example 9 identifies an important class of tree regular languages — parse trees of a context-free grammar.

**Theorem 11.** The set of derivation trees/parse trees of a context-free grammar $G = (N, T, P, S)$ is regular.

**Proof.** The proof generalizes the construction in Example 9 and is based on the same intuition. Take $\Sigma = N \cup T$. The automaton recognizing the parse trees is as follows

1. $Q = T \cup N \cup \{\ast\}$
2. $F = \{S\}$
3. $\delta_0(a) = a$ for $a \in T$, and for $i > 0$, $\delta_i(\alpha_1, \alpha_2, \ldots, \alpha_i, X) = X$ if $X \rightarrow \alpha_1 \alpha_2 \cdots \alpha_i \in P$. In all other cases, $\delta_i(\alpha_1, \alpha_2, \ldots, \alpha_i, X) = \ast$.

The regularity of parse trees of a context-free grammar is an interesting connection between context-free languages and regular tree languages. However, Theorem 11 does not provide a characterization of regular tree languages, i.e., it is not the case that all regular tree languages are collections of parse trees of some grammar. Consider the collection $B$ of all full binary trees labeled by alphabet $\Sigma = \{b, A\}$, with the property that (a) all leaves are labeled $b$ and all internal vertices are labeled $A$, and (b) every path has at most 2 $A$s and there is at least one path with exactly 2 $A$s. The language $B$ is, in fact, a finite set of trees that we can write as the following set of terms.

$$
B = \{A(A(b, b), b), A(b, A(b, b)), A(A(b, b), A(b, b))\}.
$$
Proposition 12. If \( A \) is a finite set of \( \Sigma \)-labeled trees then \( A \) is regular.

Not all tree languages are regular; we could easily argue this through a counting argument as the number of DTAs is countable but the number tree languages is uncountable. However, we can give explicit examples of languages that are not regular. Such examples help understand the computational limits of DTAs.

Theorem 13. There are tree languages that are not regular.

Theorem 14. Let \( \Sigma = \{b, A\} \). Let \( T(\Sigma) \) be the collection of all full binary trees where leaves are labeled by \( b \) and internal vertices are labeled by \( A \). Consider

\[
L = \{ A(t, t) \mid t \in T(\Sigma) \}
\]

That is, \( L \) consists of tree whose root is labeled by \( A \), and its left and right subtrees are identical.

We will prove the non-regularity of \( L \) through a proof by contradiction. Assume that \( L \) is recognized by DTA \( M \). Since \( M \) has finitely many states, there must be two trees \( t_1, t_2 \) with \( t_1 \neq t_2 \) such that the runs of \( M \) on \( t_1 \) and \( t_2 \) end in the same state. That is, if \( \rho_1 = (\text{dom}_{\rho_1}, \text{val}_{\rho_1}) \) is the run of \( M \) on \( t_1 \), and \( \rho_2 = (\text{dom}_{\rho_2}, \text{val}_{\rho_2}) \) is the run of \( M \) on \( t_2 \), then \( \text{val}_{\rho_1}(\varepsilon) = \text{val}_{\rho_2}(\varepsilon) \). Since \( M \) accepts \( A(t_1, t_1) \) and \( A(t_2, t_2) \), it also accepts \( A(t_1, t_2) \). But \( A(t_1, t_2) \notin L \) which contradicts our assumption that \( M \) recognizes \( L \).

3 Nondeterminism and Closure Properties

As for any machine model, we can consider a nondeterministic version of tree automata. The definition is identical to that for DTAs, except that the transition function returns a set of possible states for a vertex in the tree, given its label and the states of its children. Formally it can be defined as follows.

Definition 15. A nondeterministic tree automaton (NTA) on \( \Sigma \)-labeled \( n \)-ary trees is \( M = (Q, \Sigma, \delta, F) \) where \( Q \) is a finite set of states, \( F \subseteq Q \) is a set of final/accepting states, and \( \delta = \cup_{i=0}^{n} \delta_i \) is the transition function, where \( \delta_i : Q^i \times \Sigma \rightarrow 2^Q \).

Runs, acceptance and language recognized are natural generalization of Definition 5

Definition 16. The run of a NTA \( M = (Q, \Sigma, \delta, F) \) on a tree \( t = (\text{dom}_t, \text{val}_t) \) is a \( \Sigma \)-labeled tree \( \rho = (\text{dom}_\rho, \text{val}_\rho) \) where \( \text{dom}_\rho = \text{dom}_t \) and for any vertex \( u \in \text{dom}_i \) with \( i \) children,

\[
\text{val}_\rho(u) \in \delta_i(\text{val}_\rho(u0), \ldots, \text{val}_\rho(u(i-1)), \text{val}_t(u)).
\]

A run \( \rho = (\text{dom}_\rho, \text{val}_\rho) \) of \( M \) on \( t \) is accepting if \( \text{val}_\rho(\varepsilon) \in F \). A tree \( t \) is accepted by \( M \) if \( M \) has an accepting run on \( t \). Finally the language recognized by \( M \) is the set of all \( \Sigma \)-labeled \( n \)-ary trees it accepts, i.e.,

\[
L(M) = \{ t \mid M \text{ accepts } t \}.
\]

Like in the case of finite automata, nondeterminism does not add anything to the expressive power of tree automata.
Theorem 17. Let $A$ be a tree language recognized by an NTA. Then $A$ is regular.

Proof. The standard subset construction extends to this case. Let $N = (Q, \Sigma, \delta, F)$ be an NTA recognizing $A$. Consider the DTA $D = (2^Q, \Sigma, \delta', F')$ where

- $F' = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$
- $\delta'$ is as follows: $\delta'_0 = \delta_0$ and $\delta'_k(Q_1, Q_2, \ldots, Q_k, f) = \{q \mid \exists q_1, q_2, \ldots, q_k, q_i \in Q_i \text{ and } q \in \delta_k(q_1, q_2, \ldots, q_k, f)\}$.

One can prove inductively the following observation for any $\Sigma$-labeled tree $t$. Let $\rho = (\text{dom}_\rho, \text{val}_\rho)$ be the run of $D$ on $t$ with $\text{val}_\rho(\varepsilon) = P$. We can show that

$$q \in P \text{ iff there is a run } \rho' = (\text{dom}_{\rho'}, \text{val}_{\rho'}) \text{ of } N \text{ such that } \text{val}_{\rho'}(\varepsilon) = q.$$  

Using this observation we have $L(D) = L(N)$, given the definition of $F'$.

Regular tree language enjoy similar closure properties to regular languages — they are closed un Boolean operations.

Theorem 18. Regular tree languages are closed under all boolean operations.

Proof. Standard constructions for DFAs like cross product and flipping final/non-final states, extend to DTAs. Details are left to the reader to work out.

The notion of projection can be extended to trees. Let $\Sigma$ and $\Gamma$ be finite alphabets such that $|\Gamma| \leq |\Sigma|$. A projection is any function between such alphabets, i.e., $\pi : \Sigma \rightarrow \Gamma$. Given such a projection function, one can extend it to function from $\Sigma$-labeled trees to $\Gamma$-labeled trees in the natural way: for $\Sigma$-labeled tree $t = (\text{dom}_t, \text{val}_t)$, $\pi(t)$ is the $\Gamma$-labeled tree given by $(\text{dom}_{\pi(t)}, \text{val}_{\pi(t)})$ where $\text{dom}_{\pi(t)} = \text{dom}_t$ and $\text{val}_{\pi(t)}(u) = \pi(\text{val}_t(u))$. Extending projection to languages we have $\pi(A) = \{\pi(t) \mid t \in A\}$. Like regular (word) languages, regular tree languages are closed under projections.

Theorem 19. If $A$ is a regular tree language and $\pi$ is a projection, then $\pi(A)$ is also regular.

Proof. The construction of an NTA recognizing $\pi(A)$ is very similar to the way one constructs an NFA to recognize the projection of a word language. The details are left tot he reader to work out.

4 Decision Problems and MSO

5 Applications

5.1 Solving Games

5.2 Rewrite Theories