The algorithms RT and k-TTI: A first comparison.*

José Ruiz and Pedro García
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

I.-INTRODUCTION
Recognizable tree sets are a generalization of regular languages [Gécse and Steinby, 84]. The property that defines them is that they can be accepted by tree automata. Many properties of regular languages can be extended to recognizable tree sets in a very natural way. In particular, it is possible to obtain certain subfamilies of the class of recognizable tree sets as a generalization of subclasses of the class of regular languages. This is the case of two families that can be learned using positive data, the 0-reversible languages [Angluin, 82] and k-testable languages in strict sense [McNaughton, 74].

Learning recognizable tree sets or some of its subclasses is interesting for its relation to context-free languages. For any CFL, the set of skeletons of its derivation trees is a recognizable tree set, so CF languages can be learned from skeletons of their derivation trees with algorithms very similar to those used for regular languages.

We present here a first experimental comparison between two algorithms -RT and k-TTI algorithms- that learn recognizable tree sets. Both have the property that they identify in the limit a subfamily of recognizable sets from positive structural descriptions: the class of 0-reversible sets [Sakakibara, 92], and the class of k-testable sets [García, 92]. The comparison faces the possibility of approximating languages generated from arbitrary context-free grammars using skeletons of their derivation trees.

II.-PRELIMINARIES AND NOTATION

Let \( N \) be the set of natural numbers and \((N^*, \cdot)\) the free monoid generated by \( N \) with \( \lambda \) as the identity. We define \( u \leq w \) for \( u, w \in N^* \) iff there exists \( v \in N^* \) such that \( w = u \cdot v \) (\( u \leq w \) if \( u \leq w \) and \( u \neq w \)). For \( x \in N^* \), we define the length of \( x \) denoted by \( |x| \) as follows:

\[
|\lambda| = 0 \\
|x \cdot n| = |x| + 1 \text{ for } n \in N
\]

\( D \subseteq N^* \) is a tree domain iff it satisfies: a) \( v \in D \) and \( u < v \) implies \( u \in D \) b) if \( u \cdot i \in D, \ i \in \mathbb{N}, \) then \( u \cdot f \in D \) for \( 1 \leq i \leq l \).

A ranked alphabet \( V \) is a finite set associated with a finite relation \( r \subseteq (V \times N) \). \( V_t \) denotes the subset of \( V \): \( s \in V \mid (s, n) \in r \).

A tree \( t \) over a ranked alphabet \( V \) is a mapping \( t : D \rightarrow V \) with \( D \) being a tree domain called domain of \( t \) and denoted by \( \text{dom}(t) \). The set of finite trees over \( V \) will be called \( V_T \). The alphabet can be seen as a set of function symbols having different arities in the way that \( V_T \) can be considered as the set of terms over \( V \). For example, the tree shown in Figure 1 (left) can be represented as \( S(A(a, b), B(c, C(c))) \).

---

*Work supported in part by the Spanish CICYT under grants TIC93-0533-C02 and IIC102692-C02.
Let $t \in V^T$ and $x \in \text{dom}(t)$. The depth of $x$ is defined as $\text{depth}(x) = |x|$ and the depth of $t$ as $\text{depth}(t) = \max \{ \text{depth}(x) \mid x \in \text{dom}(t) \}$. The subtree of $t$ rooted at $x$, denoted as $t[x]$, is defined as:

$$\text{dom}(t[x]) = \{ y \mid y \in \text{dom}(t) \text{ and } (t[x])(y) = t(x,y) \} \forall y \in \text{dom}(t[x])$$

If $t \in V^T$, then $ST(t)$ is the set of subtrees of $t$, that is, $ST(t) = \{ t[x] \mid x \in \text{dom}(t) \}$ and for the set $T \subseteq V^T$, $ST(T) = \bigcup_{t \in T} ST(t)$. The replacement of the subtree $t[x]$ with $s \in V^T$ is defined as:

$$t(s \leftarrow x)(y) = \begin{cases} t(y) & \text{if } |y| \leq |x|; y \neq x; y \in \text{dom}(t); \\ s(z) & \text{if } y = x, z \in \text{dom}(s) \end{cases}$$

A skeletal alphabet is a ranked alphabet with exactly one symbol that has arities greater or equal to one. If $Sk$ denotes such an alphabet and $V_0$ is an alphabet of symbols whose arity is zero, a tree over $Sk \cup V_0$ is called a skeleton (all its inner nodes are labelled by $\sigma$ and the leaves are symbols from $V_0$). If $t \in V^T$, $sk(t)$ denotes the skeleton of $t$, which is obtained labelling with $\sigma$ all the inner nodes of $t$. For the set $T \subseteq V^T$, the set of skeletons associated to the trees in $T$ is $sk(T) = \bigcup_{t \in T} sk(t)$.

![Figure 1. Example of a derivation tree and its associated skeleton](image)

Let $V$ be a ranked alphabet and $m$ the greatest arity of the symbols in $V$. A deterministic tree automaton (DTA) is defined as the four-tuple $A = (Q, V, \delta, F)$ where $Q$ is a finite set (states), $F \subseteq Q$ is the set of final states and $\delta = (\delta_0, \delta_1, \delta_m)$ the set of state transition functions defined by:

$$\delta_k : (V_0 \times Q^k) \to Q, k = 1, 2, \ldots, m$$

$$\delta_0(a) = a, \forall a \in V_0$$

$\delta$ can be extended to operate on trees as follows:

$$\delta(\sigma(t_1, \ldots, t_n)) = \begin{cases} \delta_{\sigma}(\delta(t_1), \ldots, \delta(t_n)) & \text{if } n > 0 \\ \delta_0(\sigma) & \text{if } n = 0 \end{cases}$$

A tree $t \in V^T$ is accepted by $A$ if $\delta(t) \in F$. The set of trees accepted by $A$ is defined as $T(A) = \{ t \in V^T \mid \delta(t) \in F \}$.

For a context-free grammar $G = (N, \Sigma, P, S)$ and for each symbol $X \in N \cup \Sigma$ we define the set of trees from $G$ rooted at $X$ as:

$$D_X(G) = \{ \{ a \} \text{ if } X = a \in \Sigma \\ \{ X(t_1, \ldots, t_k) \mid X \to B_1 \ldots B_k, t_i \in D_{B_i}(G) \} \text{ if } A \in N \}$$

$D_S(G)$ denotes the set of derivation trees of $G$. If $Sk = \{ \sigma \}$, the set of trees over $Sk \cup \Sigma$ which are skeletons associated to trees in $D_S(G)$ is denoted by $sk(D(G))$. Both $D_S(G)$ and $sk(D(G))$ are regular tree sets.
Let $A = (Q, S_k, \Sigma, \delta, F)$ be a DITA for a set of skeletons. There exists a context-free grammar $G = (N, \Sigma, S, P, S)$ such that $sk(D(Q)) = T(A)$ which can be obtained as follows:

$N = Q \cup \{S\}$;

$P = \{\delta_\sigma(q_i, \ldots, q_k) \rightarrow q_1, \ldots, q_k : \sigma \in S_k, q_1, \ldots, q_k \in Q\}$

$\cup \{S \rightarrow q_1, \ldots, q_k : n \geq 1, \delta_\sigma(q_1, \ldots, q_k) \in F\}$

III. K-TESTABLE TREE SETS IN THE STRICT SENSE

Let $(V, r)$ be a ranked alphabet, $k \geq 2$ and let $V^T$ be the set of finite trees over $V$. For every $t \in V^T$, the $k$-test vector of $t$ is defined as: $Test_k(t) = (r_{k-1}(t), l_{k-1}(t), p_k(t))$ where:

$$r_{k-1}(t) = \begin{cases} t & \text{if depth}(t) \leq k - 2 \\
 x & \text{if depth}(t) > k - 2
\end{cases}$$

$$l_{k-1}(t) = \{t' : \text{depth}(t') = k - 2, \text{dom}(t') = \text{dom}_{k-2}(t), t'(y) = t(y) \forall y' \in \text{dom}(t') \}$$

where $\text{dom}(t) = \{x \in \text{dom}(t) : |x| \leq k\}$.

$$p_k(t) = \begin{cases} \varnothing & \text{if depth}(t) < k - 1 \\
 \{r_k(t') : t' \in SI(t), \text{depth}(t') \geq k - 1\} & \text{if depth}(t) \geq k - 1
\end{cases}$$

Example 1. Let

$$t = \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array}$$

Then:

$$r_3(t) = \begin{array}{c}
\text{a} \\
\text{b}
\end{array}$$

$$p_3(t) = \{ \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array}, \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array} \}$$

$$l_2(t) = \{ \begin{array}{c}
\text{a}, \text{b}, \text{c}
\end{array}, \begin{array}{c}
\text{a}, \text{b}, \text{c}
\end{array} \}$$
Let $\equiv_k$ be the equivalence relation in $V^I$ defined as:
\[ \forall s, t \in V^I, \quad s \equiv_k t \iff T_{T_{k+1}}(s) = T_{T_{k+1}}(t). \] Obviously $\equiv_k$ is a subtree invariant relation of finite index. A set $T \subseteq V^I$ is $k$-Testable ($k$-T) iff it is the union of some of the equivalence classes defined by $\equiv_k$. (so $k$-T sets are also regular sets). A set is Locally Testable (LT) if $t$ is $k$-T for some $k$. A tree language is $k$-Testable in a Strict sense ($k$-TS) if there exist three finite sets $R, L, P$ such that for every $t \in T$:
\[ r_{k+1}(t) \in R, \quad l_{k+1}(t) \subseteq L, \quad p_{k+1}(t) \subseteq P. \]

The family of the $k$-T languages is the boolean closure of the $k$-TS family.

**Remark:** An equivalence relation $=_{\equiv_k}$ defined over $V^I$ is called subtree invariant if $t_1 =_{\equiv_k} t_2$ implies $\forall t \in V^I, \quad \forall x \in D_t, \quad t(x \leftarrow t_x) =_{\equiv_k} t(x \leftarrow t_x)$.

**IV THE k-TTI INFERENCE ALGORITHM**

We can associate to the finite subset $S \subseteq V^I$ the four-tuple $Z_k(S) = (V(S), R_k(S), L_k(S), P_k(S))$ being $V(S)$ the ranked alphabet from $S$ and
\[ R_k(S) = \{ r_{k+1}(t) : t \in S \}, \quad L_k(S) = \bigcup_{i=1}^{k} L_{k+1}(t), \quad P_k(S) = \bigcup_{i=1}^{k} P_{k+1}(t) \]
The language defined by $Z_k(S)$ will be denoted as $T_k(S)$

**Properties:** Let $S, S'$ be two finite tree sets, $k \geq 2$ and $T_k(S)$ and $T_k(S')$. Then:

a) $S \subseteq T_k(S)$
b) $T_k(S)$ is the smallest $k$-TS tree set containing $S$
c) $S \subseteq S' \Rightarrow T_k(S) \subseteq T_k(S')$
d) $T_{k+1}(S) \subseteq T_k(S)$
e) If $k > 1 + \max_{i \in S} \{ \text{depth}(t) \}$ then $T_k(S) = S$

**k-TTI Inference Algorithm**

<table>
<thead>
<tr>
<th>INPUT:</th>
<th>$k \geq 2$, $S$ finite set of skeletons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT:</td>
<td>$A_k = (Q, V, \delta, F)$</td>
</tr>
<tr>
<td>METHOD:</td>
<td>$(V, R, L, P) := (V(S), R_k(S), L_k(S), P_k(S))$</td>
</tr>
<tr>
<td></td>
<td>$Q := R \cup L \cup P_{k-1}(P)$</td>
</tr>
<tr>
<td></td>
<td>$F := R$</td>
</tr>
<tr>
<td></td>
<td>$\forall t \in L \quad \delta(t) := t$</td>
</tr>
<tr>
<td></td>
<td>$\forall \sigma(t_1, \ldots, t_n) \in P \quad \delta_{k}(\sigma(t_1, \ldots, t_n)) = r_{k+1}(\sigma(t_1, \ldots, t_n))$</td>
</tr>
<tr>
<td></td>
<td>$A_k = (Q, V, \delta, F)$</td>
</tr>
</tbody>
</table>

*Fig. 2. Inference algorithm*
Example 2. Let \( k = 3 \) and \( S \) the positive structural sample:

\[
S = \{ \text{a, b, c} \}.
\]

\( A_3 = (Q, V, \delta, F) \), with:

\[
Q = \{ \text{a, b, c} \}.
\]

\[
F = \{ \}
\]

and the transition functions:

\[
\delta_\emptyset(a) = a, \quad \delta_\emptyset(b) = b, \quad \delta_\emptyset(c) = c,
\]

\[
\delta_\{a\}(a, b) = \sigma(a, b), \quad \delta_\{a\}(a, c) = \sigma(a, c), \quad \delta_\{a\}(b, c) = \sigma(b, c),
\]

\[
\delta_\{b\}(a, b) = \sigma(a, b), \quad \delta_\{b\}(a, c) = \sigma(a, c), \quad \delta_\{b\}(b, c) = \sigma(b, c),
\]

\[
\delta_\{c\}(a, b) = \sigma(a, b), \quad \delta_\{c\}(a, c) = \sigma(a, c), \quad \delta_\{c\}(b, c) = \sigma(b, c),
\]

Example 4. Let \( k = 3 \) and \( S \) be considered as a set of skeletons of a context-free grammar the same as Example 2. From the automaton \( A_3 \) obtained in Example 3, we obtain a context-free grammar \( G = (N, \Sigma, P, S) \) such that \( s_h(D(G)) = I(A_3) \).

By renaming the states in \( Q - \Sigma \) in order to simplify the notation: \( \sigma(\emptyset, \emptyset) = A, \sigma(a, b) = A_2, \sigma(c) = B_2, \sigma(a, c) = A_1, \sigma(c, c) = B_1 \), we obtain the following grammar:

\[
\Sigma = \{ a, b, c \}; \quad N = \{ A, B, A_1, B_1, A_2, B_2 \};
\]

\[
P = \{ S \to A_1 B_1, A_2 B_2, A_1 \to aA_1 b, aA_2 b, B_1 \to cB_1, cB_2, A_2 \to ab, B_2 \to c \}.
\]

V. REVERSIBLE CONTEXT FREE GRAMMARS

DEFINITION A context-free grammar \( G = (N, \Sigma, P, S) \) is called invertible iff for any two productions of the form \( A \to \alpha \) and \( B \to \alpha \), then \( A = B \). Invertible grammars constitute a normal form for CFGs.

DEFINITION A context-free grammar \( G = (N, \Sigma, P, S) \) is said to be reset-free if \( \forall B, C \in N, \alpha, \beta \in (N \cup \Sigma)^* \), \( (A \to \alpha \beta \beta \) and \( A \to \alpha C \beta) \Rightarrow B = C \).

DEFINITION A context-free grammar \( G = (N, \Sigma, P, S) \) is reversible iff \( G \) is invertible and reset-free. Reversible grammars constitute a normal form for CFGs [Sakakibara, 92].

VI. RT LEARNING ALGORITHM FOR TREE AUTOMATA [Sakakibara, 92]

a) DESCRIPTION The input to the algorithm is a finite set \( S \) of skeletons. It begins constructing the base tree automaton \( A \) for \( S \) and then generalizes by merging up states. Once it constructs the finest partition \( \pi_1 \) of the set \( Q \) of states of \( A \) with the property that \( A/\pi_1 \) is reversible, outputs \( A/\pi_1 \).

Beginning with the trivial partition of \( Q \), \( \pi_1 \), two distinct blocks \( B_1 \) and \( B_2 \) are merged in one of the stages of the process if any of the following conditions is satisfied:

1. \( \beta \) and \( \beta' \) contain final states of \( A \). (A reversible tree automaton must contain just one final state).
2. There exist \( q \in B_i \) and \( q' \in B_j \) with \( q = \sigma(u_1, \ldots, u_k) \) and \( q' = \sigma(u'_1, \ldots, u'_k) \) such that for \( 1 \leq j \leq k \), \( u_j \) and \( u'_j \) are in the same block or the same terminal symbols (Fig. 3)

![Diagram illustrating two states merging](image)

Fig. 3. In a situation like this, \( q \) and \( q' \) have to be merged.

3. There exist two states \( q, q' \) with \( q = \sigma(u_1, \ldots, u_k) \) and \( q' = \sigma(u_1', \ldots, u_k') \) in the same block and an integer \( 1 \leq l \leq k \) such that \( u_l \in B_i \) and \( u'_l \in B_j \) and for \( 1 \leq j \leq k \) (\( j \neq l \)), \( u_j \) and \( u'_j \) are in the same block or the same terminal symbols (Fig. 4)

![Diagram illustrating state merging](image)

Fig. 4. If \( q \) and \( q' \) are in the same block, the blocks containing \( u \) and \( u' \) are merged.

Next figure shows the algorithm RT as it is described in Sakakibara's article:

```
INPUT: a positive sample \( S \) of skeletons.
OUTPUT: a reversible skeletal tree automaton \( A \).

METHOD:
   Let \( A = (Q, \Sigma, S, F) \) the automaton \( SA(S) \);
   Let \( \pi_0 \) the trivial partition of \( Q \);
   Choose \( q \in F \);
   \( \text{LIST} = \{(q, q'), \forall q' \in F - \{q\}\} \);
   \( i = 0 \);
   While \( \text{LIST} \neq \emptyset \) do
      Remove the first element \((q_1, q_2)\) from \( \text{LIST} \);
      Let \( B_1 = B(q_1, \pi_i) \) and \( B_2 = B(q_2, \pi_i) \);
      If \( B_1 \neq B_2 \) then
         \( \pi_{i+1} = \text{merge}(\pi_i, B_1, B_2) \);
         \( p\text{-update}(\pi_{i+1}) \) and \( s\text{-update}(\pi_{i+1}, B_1, B_2) \);
         \( i = i + 1 \);
      end (if)
   end (while)
   \( i = i \); Outputs \( A/\pi_{i} \);
end.
```
The two routines used in the algorithm are:

\[ \text{- update}(\pi_{s_1}) \]

For all pairs of states \( \sigma(u_1, ..., u_k) \) and \( \sigma(u'_1, ..., u'_k) \) in \( Q \) with
\[
B(u_j, \pi_{s_1}) = B(u'_j, \pi_{s_1}) \text{ or } u_j = u'_j, \quad \forall 1 \leq j \leq k \text{ and } \\
B(\sigma(u_1, ..., u_k), \pi_{s_1}) \neq B(\sigma(u'_1, ..., u'_k), \pi_{s_1})
\]
Add \( \sigma(u_1, ..., u_k), \sigma(u'_1, ..., u'_k) \) a LIST.

\[ \text{s-update}(\pi_{s_1}, B_1, B_2) \]

For all pair of states \( \sigma(u_1, ..., u_k) \in B_1 \) and \( \sigma(u'_1, ..., u'_k) \in B_2 \) such that
\[
u_1, u'_1 \in Q \text{ and } B(u_1, \pi_{s_1}) \neq B(u'_1, \pi_{s_1}) \text{ for } 1 \leq l \leq k \text{ and } \\
u_1 = u'_1 \in Q, \quad \forall 1 \leq j \leq k, \quad j \neq l
\]
Add the pair \( (u_1, u_1) \) a LIST.

VII EXPERIMENTAL RESULTS

It has been theoretically proved that the algorithm k-TII identifies the family of k-TS tree sets from positive structural information in the limit and so does the algorithm RT with CF reversible languages.

The experiments carried out so far are a first attempt to compare the behaviour of both algorithms when they are used to identify CFLs from structural descriptions of arbitrary (meaning that they need not be neither k-TS nor reversible) CF grammars. Those experiments are to be considered preliminary and we are currently doing a better comparison using more CFG grammars chosen in a random way.

We first chose 10 CFG in an arbitrary way from several "Formal Language Theory" books. For each of these grammars, appropriate sets of structural training and validation data were randomly generated. The training data consisted in a set of 100 skeletal trees obtained randomly from the grammar. The validation data - used to observe the evolution of the algorithm - consisted in a set of 400 of skeletal trees, also obtained from the grammar. These experiments were repeated 10 times for each grammar. The average of training sentences required for convergence of both algorithms is shown in Table 1 (1).

For each automata learned in this way, we obtained the corresponding CF grammar, as in II. Then we used Earley's algorithm to classify the first 1000 sentences of \( \Sigma^* \) (in lexicographic order) with the target grammar and with the grammars obtained with the algorithms. The percentage of well classified sentences for different grammars is shown in Table 1.

The experiments have been done using Mathematica language [Wolfram, 91]. While RT has correctly classified about 5% more words than k-TII (for k=3), the former seems to run much slower than the latter. For instance, training the fifth grammar of Table 1 with the same 50 skeletons took 18,89 seconds for the algorithm k-TII and 795,49 seconds for the algorithm RT on a Next Station.
<table>
<thead>
<tr>
<th>GRAMMARS</th>
<th>TSFC (1)</th>
<th>CCTS (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k-TII</td>
<td>RT</td>
</tr>
<tr>
<td>S-&gt;AB A-&gt;aAb</td>
<td>ab, B-&gt;cB, B-&gt;c</td>
<td>7 5.6</td>
</tr>
<tr>
<td>S-&gt;aa</td>
<td>AbB</td>
<td>bbA, A-&gt;ab</td>
</tr>
<tr>
<td>S-&gt;aAa</td>
<td>bSa</td>
<td>sb, A-&gt;aaB, B-&gt;B</td>
</tr>
<tr>
<td>S-&gt;ASa</td>
<td>b</td>
<td>Sa</td>
</tr>
<tr>
<td>S-&gt;Sa</td>
<td>b</td>
<td>Sa</td>
</tr>
<tr>
<td>S-&gt;aA</td>
<td>bAb</td>
<td>a</td>
</tr>
<tr>
<td>S-&gt;AB</td>
<td>BC, A-&gt;AB</td>
<td>a, B-&gt;AA</td>
</tr>
<tr>
<td>S-&gt;AbAb</td>
<td>AbAb</td>
<td>A</td>
</tr>
<tr>
<td>S-&gt;aAbAb</td>
<td>AbAb</td>
<td>a</td>
</tr>
<tr>
<td>S-&gt;aB</td>
<td>bA, A-&gt;b</td>
<td>a</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>12.5 7.4</td>
<td>85.13 90.61</td>
</tr>
</tbody>
</table>

Table 1: Rate of learning of the algorithms when tested with all the words of $Σ^*$ up to length 10.
(1) Training sentences required for convergence
(2) Correct classification of the test set (%).

Some other features shown by the experiments done so far are:

(1) While the time needed by k-TII is always increasing with the amount of data, RT may decrease some times (the time curves present some "saddle points").

(2) The grammars obtained with both algorithms classify correctly all the positive words of the language, they only may fail in overgeneralizing if the grammar is not of the correct type.

(3) The algorithm k-TII achieves better classification rates for greater values of $k$, at the cost of less generalization (which means longer time of computation to get the definitive automata). The chosen value $k = 3$ is the one that gives better ratio: classification rate / learning-time.

VIII. CONCLUSIONS
As it has been discussed, both the RT and k-TII algorithms obtain similar results. They will hopefully work even better with CF grammars chosen randomly.

Our future experiments will try to measure in a more accurate way the classification rate and especially, the evolution of learning.

IX. REFERENCES


[Mc Naughton, 74] MC NAUGHTON, R. "Algebraic decision procedures for local testability"
Mathematics Systems Theory, 8(1), 60-76, 1974
