INFORMATIQUE THÉORIQUE ET APPLICATIONS

NICOLA GALESI

A syntactic characterization of bounded-rank decision trees in terms of decision lists

Informatique théorique et applications, tome 31, nº 2 (1997), p. 149-158

<http://www.numdam.org/item?id=ITA_1997__31_2_149_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 31, n° 2, 1997, pp. 149-158)

A SYNTACTIC CHARACTERIZATION OF BOUNDED-RANK DECISION TREES IN TERMS OF DECISION LISTS (*)

by Nicola GALESI ([1])([†])

Abstract. – *We define syntactically a sub-class of decision lists* (tree-like decision lists) *and we show its equivalence with the class of bounded rank decision trees. As a by-product, the main theorem provides an alternate and easier proof of the Blum's containement Theorem* [1]. *Furthermore we give an inversion procedure for Blum's derivation of a decision list from a bounded rank decision tree.*

Résumé. – *Nous définissons syntactiquement une sous-classe de listes de decision* (tree-like decision lists) *et nous montrons son équivalence avec la classe des arbres de décision de rang borné. Comme sous-produit, le théorème principal fournit une preuve alternative et plus simple du Théorème d'inclusion de Blum* [1]. *En plus, nous donnnons une procédure d'inversion pour la dérivation de Blum d'une liste de décision à partir d'un arbre de décision de rang borné.*

1. INTRODUCTION

Decision lists have been introduced by Rivest in [3] as a representation of boolean functions. He showed that k-decision lists, *i.e.* decision lists in which any term has at most k literals, are (1) a generalization of k-CNF, k-DNF and of depth-k decision trees and (2) are polynomially learnable under PAC model. [2] showed that constant rank decision trees are also polynomially PAC learnable and [1] showed that rank-k decision trees are a sub-class of k-decision lists, thus providing to an improvement of the result of [2] since constant rank decision trees can be polynomially PAC-learned using Rivest's algorithm for k-decision lists as subroutine.

Here we define a sub-class of decision lists - the class of *tree-like decision lists*. For the lists of this class we define the *rank* measure and we show that the class of rank-k decision trees is equivalent to the class of rank-k tree-like

(*) Received March 1996.
([1]) *Author's Address:* Departament de Llenguatges i Sistemes Informatics. Universitat Politecnica de Catalunya, Jordi Salgado Girona 1-3, 08034 Barcelona, Spain. e-mail: galesi@goliat.upc.es
([†]) Supported by an EC grant under the TMR project.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/97/02/$ 7.00/© AFCET-Gauthier-Villars

decision lists. As a by-product of Theorem 3.1, we provide an alternate proof of Blum's containement theorem.

In the final section we give an algorithm such that given a decision list L it builds a corresponding decision tree. Also when L is the list that the main Theorem of [1] produces when applied to a rank-k *reduced* decision tree T, it allows us to recover exactly T.

2. PRELIMINARIES

Let $\mathcal{V}_n$ be a set of n boolean variables $v_1, v_2, \ldots, v_n$. A *literal* ℓ_i denotes a variable v_i or its negation. Boolean *constants* are denoted by $a, b, \ldots$ A *term* or *monomial* t is a *conjunction* of literals. Terms are supposed to be strings of literals and we refer to a *prefix* of length k of a term t as the term built from the conjunction of the first (from left to right) k literals of t, with $k \leq |t|$.

A *decision list* L on a family $\{F_i\}$ of boolean functions over n variables is a sequence $(F_1, b_1), \ldots, (F_{m-1}, b_{m-1}), (1, b_m)$, with $m > 0$. On input $\vec{x} \in \{0,1\}^n$, it computes the boolean function f_L defined as b_j where j is the least number less than $m-1$ such that $F_j(\vec{x}) = 1$, if such j exists, and b_m otherwise. Here we limit the boolean functions F_i to monomials on $\mathcal{V}_n$ like in [3]. L is a *k-decision list* if for each monomial t, $|t| \leq k$. The length $|L|$ of a decision list L is the number of monomials. A couple of the form (t, a) will be called *item* and by $(t, a)_i$ we denote the i-th item in L. Given $L = (t_1, b_1), \ldots, (t_{m-1}, b_{m-1}), (1, b_m)$ and a literal ℓ we denote by $(\ell \wedge L)$ the list $(\ell \wedge t_1, b_1), \ldots, (\ell \wedge t_{m-1}, b_{m-1}), (\ell, b_m)$.

A *decision tree* T is a binary tree such that the internal nodes are labelled with a variable of $\mathcal{V}_n$, the leaves are labelled with boolean constants and each right (respectively left) arc is labelled with 1 (respectively 0). Note that the same variable can label several internal nodes on the same path; if there is no such repetition, then the tree is said to be *reduced*. The boolean function f_T computed by T is defined in the following way: if T is a constant a then $f_T = a$, otherwise if $T = (v_i, T_1, T_2)$, then $f_T = (v_i \wedge f_{T_1}) \vee (\bar{v}_i \wedge f_{T_1})$.

The *rank* $r(T)$ of a decision tree T is the height of the largest complete binary tree that can be embedded in T. It is defined by:

$$r(T) = \begin{cases} 0 & \text{if } T = a \\ \max(r(T_1), r(T_2)) & \text{if } T = (v_i, T_1, T_2) \text{ and } r(T_1) \neq r(T_2) \\ r(T_1) + 1 & \text{if } T = (v_i, T_1, T_2) \text{ and } r(T_1) = r(T_2) \end{cases}$$

The size $|T|$ of a decision tree T is the number of its internal nodes. We refer to $\mathcal{T}_k$ as the class of rank k decision trees.

3. MAIN RESULT

First we define the class $\mathcal{L}_k$ of tree-like decision lists with rank k, proving some key properties they satisfy. Then we show the equivalence between $\mathcal{L}_k$ and $\mathcal{T}_k$.

3.1. Tree-like decision lists

To reader's convenience we state the Lemma (proved below) that guarantees the soundness of the definition of tree-like decision list.

LEMMA 3.1: *Given a tree-like decision list L, with* $|L| > 1$, *there exists a unique decomposition of L in* $(\ell \wedge L_1)$ *and* L_2 *such that* $L = (\ell \wedge L_1), L_2$, *and* L_1 *and* L_2 *are tree-like decision lists.*

DEFINITION 3.1: *A tree-like decision list (tdl) is defined inductively by:*

• $(1, a)$ *is a tdl for any* $a \in \{0, 1\}$;

• *given two tdl's* L_1 *and* L_2, *the decision list* $(\ell \wedge L_1), L_2$ *is a tdl for any literal* ℓ.

The rank $\rho(L)$ of a tdl L is 0 if $L = (1, a)$, and is obtained from $\rho(L_1)$ and $\rho(L_2)$, as for the decision trees, otherwise. $\mathcal{L}_k$ is the class of tdl's having rank k.

Observe that a rank-k tdl is not necessarily a k-decision list. For example, the 3-decision list $((v_1 \wedge v_2 \wedge v_3, 1), (v_1 \wedge v_2, 1), (v_1, 1), (1, 0))$ has rank 1.

It is easy to see that in a tdl L of length greater than 1 there is always a first item having a term t such that $|t| = 1$ (so $t = \ell$) and all t_i's in the previous items of L, if any, start with ℓ and have length at least 2. This observation allows us to prove the key property (Lemma 3.1) of the tdl's, namely: from a tdl L, there is a unique way to recover the two sub-tdl's L_1 and L_2 and the literal ℓ that define it.

Proof of Lemma 3.1: The decomposition of L is as follows:

• Starting from the leftmost item of L, search for the first term t such that $|t| = 1$;

• define $(\ell \wedge L_1)$ by taking all the items of L up to t, define L_2 as the remaining items of L.

Suppose that this decomposition is not unique so that L can be written as $(\ell' \wedge L_1'), L_2'$. By hypothesis, by the decomposition and by the previous observation we have that ℓ and ℓ' must be the same literal and they must be in the same item of L. Since the items in $(\ell' \wedge L_1'), L_2'$ and in $(\ell \wedge L_1), L_2$

are the same, it follows immediately that $L_2' = L_2$ and therefore $L_1' = L_1$. So the decomposition of L with respect to its sub-tdl's is unique. □

We define the boolean function ϕ_L associated with a tdl L in terms of the tree structure as follows: $\phi_L = a$ if $L = (1, a)$ and $\phi_L = (\ell \wedge \phi_{L_1}) \vee (\bar{\ell} \wedge \phi_{L_2})$ otherwise. Then the previous property allows us to show that the boolean function f_L computed by L is ϕ_L.

LEMMA 3.2: *For any tree-like decision list L,* $f_L = \phi_L$.

Proof: By induction on $|L|$. Suppose $|L| > 1$, since if $|L| = 1$ the result is trivial. By Lemma 3.1 we find uniquely ℓ, L_1 and L_2 such that $L = (\ell \wedge L_1), L_2$ and by inductive hypothesis $\phi_{L_i} = f_{L_i}$ for $i = 1, 2$. If $\ell = 1$, then $f_L = f_{L_1} = \phi_{L_1}$, since the last term of L_1 is the true term. On the other hand, if $\ell = 0$, then all terms in $(\ell \wedge L_1)$ are falsified and so $f_L = f_{L_2} = \phi_{L_2}$. So $f_L = (\ell \wedge \phi_{L_1}) \vee (\bar{\ell} \wedge \phi_{L_2}) = \phi_L$. □

3.2. Equivalence result

THEOREM 3.1: *For any decision tree* $T \in \mathcal{T}_k$, *there is an equivalent tdl* $L \in \mathcal{L}_k$, *moreover L is k-decision list and and the size of L is equal to the number of leaves of T.*

Proof: By double induction on the height and on the rank of T. If $r(T) = 0$ and $T = a$, then $L = (1, a)$ and the result is immediate. Now, Let $r(T) = k$ and suppose that ℓ is the literal at the root of T and that T_1 and T_2 are respectively the right and the left sub-trees of T. By definition of rank at least one between T_1 and T_2 has rank at most $k - 1$. Assume without loss of generality that T_1 has this property. Let L_1 and L_2 be the two tdl's associated respectively with T_1 and T_2, having their same rank and granted by the inductive hypothesis. The list L we associate with T is therefore $(\ell \wedge L_1), L_2$. Thus $r(T) = \rho(L)$ and $f_T = f_L$ since by inductive hypothesis we have $r(T_i) = \rho(L_i)$ and $f_{T_i} = f_{L_i}$ for $i = 1, 2$. Observe that the role of L_1 and L_2 is compulsory if we want to obtain a k-decision list. □

Observe that the proof of this Theorem, suggested by one of the Referees, implicitly defines another way to obtain Theorem 1 of [Bl]. Here we give a sketch of its original proof since it will be useful in the next section.

THEOREM 3.2 ([1]): *For any decision tree* $T \in \mathcal{T}_k$ *of* m *leaves there exists an equivalent k-decision list of size at most* m.

Proof: By induction on m. If $m = 1$ or $m = 2$ the result is easy. Suppose $m > 2$, observe that if $r(T) = k$, then there is a path of length at most k

ending in a leaf a. Consider the item (t, a) where t is the term associated to this path and consider the tree $\overline{T} = T - t$ obtained by by-passing T with respect to t, *i.e.* eliminating the node in T corresponding to the last variable in t and attaching the brother sub-tree of the leaf a to the node above it. Since $\overline{T}$ has at most $m - 1$ leaves, by inductive hypothesis we have that $L_{\overline{T}}$ is the list associated to $\overline{T}$. The list L is therefore $(t, a), L_{\overline{T}}$. Since the length of each term is bounded by the rank of T, L is a k-decision list; and, since we repeat the above procedure for each leaf in T, the length of L is at most m. □

The reverse inclusion is given by the following Theorem.

THEOREM 3.3: *For any tdl $L \in \mathcal{L}_k$, there is an equivalent decision tree $T \in \mathcal{T}_k$.*

Proof: By induction on $|L|$. If $|L| = 1$, then $L = (1, a)$ so $T = a$. If $|L| > 1$, then by Lemma 3.1 we can identify uniquely ℓ, L_1 and L_2 such that $L = (\ell \wedge L_1), L_2$. Given T_1 and T_2 associated respectively with L_1 and L_2, we build the tree $T = (\ell, T_1, T_2)$ according to the sign of ℓ. Then $r(T) = \rho(L)$ and $f_T = f_L$ since by inductive hypothesis we have $r(T_i) = \rho(L_i)$ and $f_{T_i} = f_{L_i}$, for $i = 1, 2$. □

Given $L \in \mathcal{L}_k$ the number of steps required to build $T \in \mathcal{T}_k$ is $O(|L| \log |L| + (|L| - 2^k)^2)$. To see this we first discuss the case in which T is a complete binary decision tree of depth k, then we consider the general case.

Consider the algorithm implicitly defined by the previous Theorem subdivided in phases as follows. At the *first phase* we search for the first term in L from the left having size 1, in $|L|$ items, using the decomposition algorithm of Lemma 3.1. We have thus identified the literal at the root of T and the two sub-tdl's L_1 and L_2 of L. At the *second phase* we search sequentially in L_1 and L_2 for two terms of size 1 in only $|L| - 1$ items, since $|L_1| + |L_2| = |L|$ and we can exclude from the search the term identified at the previous phase. In general, at the j-th *phase*, we search for 2^{j-1} terms of size 1 in $(|L| - (2^{j-1} - 1))$ items.

Observe that after j phases such that $\sum_{i=0}^{j} 2^i = |L|$ we have identified all the terms in L. Thus the number of phases is $j = O(\log |L|)$. The total number of steps required to build (a binary complete decision tree) T is $\sum_{i=1}^{j} (|L| - (2^{i-1} - 1))$ and this is $O(|L| \log |L|)$.

Observe that if $T \in \mathcal{T}_k$, then a complete binary tree T_c of depth k is always embedded in T. This means that in the general case of a not necessarily complete decision tree $T \in \mathcal{T}_k$, at some point the algorithm will recover

T_c. By previous observation, this part requires at most $O(|L| \log |L|)$ steps and eliminates 2^k items from L. For the remaining $|L| - 2^k$ items in L we can only say that in each phase the algorithm eliminates at least one item. So in the worst case this second part requires $O((|L| - 2^k)^2)$ steps. Therefore the total number of steps is $O(|L| \log |L| + (|L| - 2^k)^2)$. Observe that when L corresponds to a complete decision tree our algorithm runs in time $O(|L| \log |L|)$.

As remarked in Lemma 1 of [2], for any decision tree T, $r(T) \leq \log(|T| + 1)$ since the smallest decision tree of rank k is the complete binary tree of depth k. This means that a decision tree of size n can be represented by a tdl $L \in \mathcal{L}_{\lceil \log(n+1) \rceil}$. On the other hand, since the minimal decision tree computing the parity function over n variables requires a complete binary decision tree with 2^n leaves, the minimal tdl computing the parity function belongs to $\mathcal{L}_n$ but must have length no less than 2^n.

Moreover it is obvious that a k-rank tdl can be represented by a k-decision list (Theorems 3.3 and 3.2). For the reverse inclusion we can only say that, since a k-decision list L has a trivial representation as a decision tree of size $\leq k^{|L|}$, then L can be represented by an equivalent tdl $L' \in \mathcal{L}_{\lceil \frac{|L|}{\lfloor \log k \rfloor} \rceil}$ but of length $k^{|L|}$.

4. RECOVERING BOUNDED RANK REDUCED DECISION TREES

The procedure converting a rank k tdl into an equivalent rank-k decision tree is straightforward. On the other side recovering a rank-k decision tree from the k-decision list produced by Blum's procedure requires some more work. In this section we present an algorithm, *Rec-Tree*, to recover decision trees from decision lists. Moreover if L_T is the decision list produced by Theorem 3.2 when applied to the reduced decision tree T we have that *Rec-Tree*$(L_T) = T$.

Let $path(T)$ be the set of terms associated with paths of T. Let $t = \ell_1 \wedge \ldots \wedge \ell_k$ be a term in $path(T)$, ending with leaf a. In order to view T as in Part 1 of Figure 1, for each variable in t we define $+, - \in \{0, 1\}$ according to the sign (respectively the negated sign) of ℓ_i in t . Moreover if $T = (v_i, T_1, T_2)$ we denote T_1 by T^{i^+}, T_2 by T^{i^-} and $(T^{i^+})^{j^-}$ by $T^{i^+j^-}$ (with $+$ and $-$ submitted to the restrictions above). A simple relation between T and $\overline{T} = T - t$ is given by the following Remark.

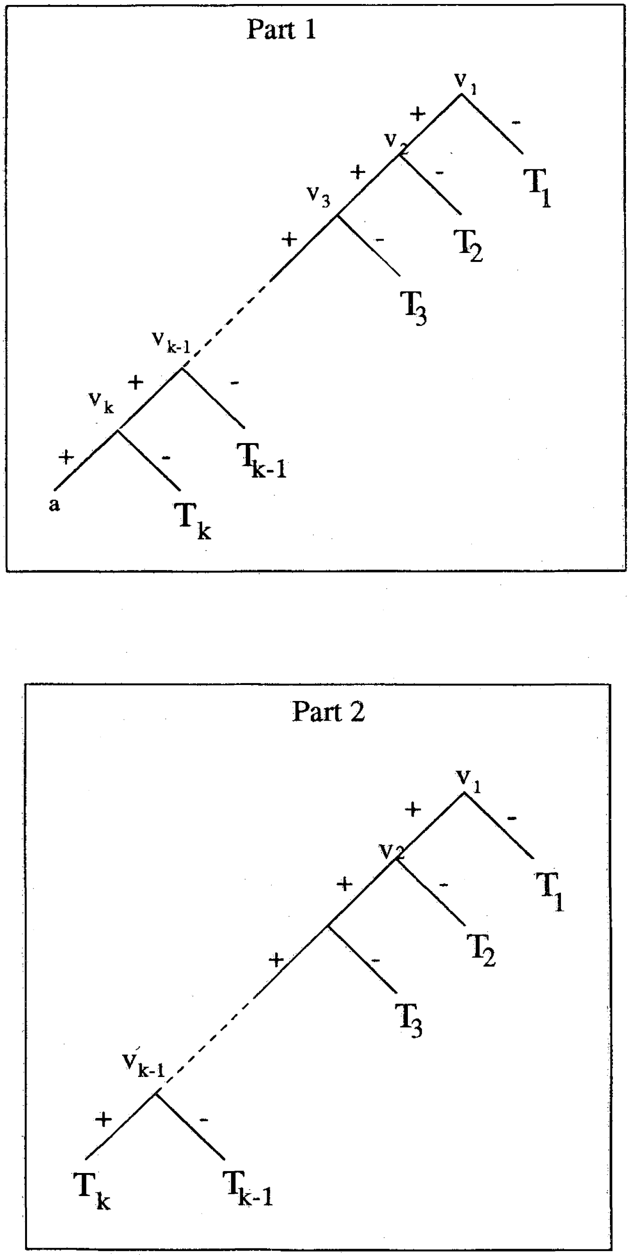

Figure 1. – Part 1: the decision tree T **wrt** $t = \ell_1 \wedge \ldots \wedge \ell_k$;
Part 2: the decision tree $\overline{T} = T - t$ **wrt** t.

REMARK 4.1: *Let T be a decision tree as in Part 1 of Figure 1 and let $t \in path(T)$ be a term $\ell_1 \wedge \ldots \wedge \ell_k$ with $1 \leq k \leq r(T)$ ending with leaf a:*

- *if $|t| = 1$, then $T_1 = \overline{T}$;*
- *if $|t| > 1$, then for any $1 \leq i \leq k-1$, $T_i = \overline{T}^{1^+2^+\ldots(i-1)^+i^-}$ and $T_k = \overline{T}^{1^+2^+\ldots(k-2)^+(k-1)^+}$.*

On a given decision list L, *Rec-Tree* works as follows: at the first step it recovers the constant decision tree T_1 from the default term of L; at the i-th step it recovers T_i by: (1) taking the $(|L|-i+1)$-th item $(t,a)_{|L|-i+1}$ of L; (2) building the trivial decision tree consistent with the term t and the constant a and putting the tree T_{i-1}, recovered at the previous step, at the unused nodes of this tree; (3) reducing each one of the T_{i-1}'s according to the path followed to reach it.

In what follows we provide more details about the algorithm. In order to have a more efficient reduction step and to simplify the proof of the theorem we merge the second and the third step, reducing the T_{i-1}'s as soon as they have to be attached to a node and working at each node on the previously reduced T_{i-1}.

Let $sgn(\ell,t)$ and $nsgn(\ell,t)$ be two functions computing respectively the sign and the negated sign of ℓ in t and let $root(T)$ be a function giving the variable at the root of T. Consider the following sub-routines:

1. BTV (Build a Tree wrt to a Variable), that takes as inputs a variable v_i, a term t and two decision trees T_1 and T_2 and outputs the tree $T = (\ell_i, T_1, T_2)$ according to the sign of v_i in t;
2. RT (Reduce Tree), that takes as inputs a variable v_i, $sg \in \{0,1\}$ and a decision tree T and outputs the decision tree T^* as follows:
   if $(T = a)$ **or** $(Root(T) \neq v_i)$
   then $T^* = T$;
   else T^* is the sub-tree of T chosen according to sg;
3. BTT (Build Tree wrt a Term), a recursive sub-routine that takes as input an item of the form (t,a) and a decision tree T, outputs the decision tree T^* as follows:
   if $|t| = 1$
   then $T^* = BTV(t^{=1}, t, a, T)$;
   else
   $T^+ = RT(t^{=1}, sgn(t^{=1},t), T)$;
   $T^- = RT(t^{=1}, nsgn(t^{=1},t), T)$;
   $T^* = BTV(t^{=1}, t, BTT((t^{>1},a),T^+), T^-)$;

4. Finally *Rec-Tree*, that takes as input a decision list L, outputs a decision tree T, defined recursively as follows

   if $L = (1, a)$

   then $T = a$;

   else $T = BTT((t, a)_1, Rec\text{-}tree(L - (t, a)_1))$;

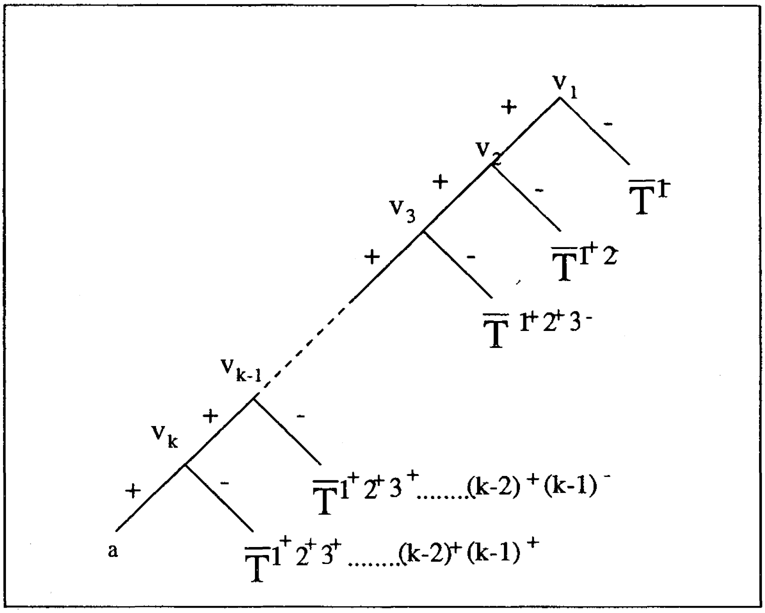

Figure 2. – The output of BTT on inputs $t = \ell_1 \wedge \ldots \wedge \ell_k$ and $\overline{T} = T - t$.

THEOREM 4.1: *For any reduced tree $T \in \mathcal{T}_k$, Rec-Tree(L_T) outputs T in $O(|L|k)$ steps.*

Proof: By induction on the number m of leaves of T. Let $m > 1$, since the case $m = 1$ is immediate by definition of *Rec-Tree*. Let $t = \ell_1 \wedge \ldots \wedge \ell_k \in path(T)$ be the term chosen ending with leaf a and let $\overline{T} = T - t$. By inductive hypothesis $Rec-Tree(L_{\overline{T}}) = \overline{T}$ and by Theorem 3.2 $L_T = (t, a), L_{\overline{T}}$. The theorem follows showing that $BTT((t, a), \overline{T}) = T$ and this is obtained by cases on $|t|$: if $|t| = 1$, then $t = \ell_k$ for some ℓ_k. Since v_k is the root label of T and T is a reduced tree, then v_k does not occur as label of any node of $\overline{T}$ (so we have no need to reduce it in BTT). By definition of BTV we obtain T. If, otherwise, $t = \ell_1 \wedge \ldots \wedge \ell_k$ with

$k > 1$, then the result follows by Remark 4.1 observing that, in this case, BTT outputs the tree of Figure 2.

Observe that if $T \in \mathcal{T}_k$, then every term in L_T has length bounded by k, so for each term in L, BTT calls itself at most k times. Since *Rec-tree* calls BTT $|L| - 1$ times, the total number of steps to output T is $O(|L|k)$. $\square$

Observe that *Rec-tree* can be used to recover decision trees from any decision list. Suppose that we modify *Rec-tree* by eliminating the reduction sub-routine, and that we run the modified algorithm on a k-decision list L. It is easy to see that in $O(|L|k)$ steps, *Rec-tree* outputs a decision tree T consistent with L of depth $\leq k|L|$ but of size $\leq k^{|L|}$. On the other hand, supposing that $k^{|L|} \gg |\mathcal{V}_n|$ and that the minimal decision tree consistent with L has size, for example, polynomial in $|L|$, it could be interesting to study under what kind of hypothesis and what kind of modifications of *Rec-tree*, such a decision tree can be obtained, using, for istance, a fully reducing subroutine that, for each variable in the currently analyzed term, always explores the whole tree produced at the previous step.

AKNOWLEDGEMENTS

I am greatly indebted to José Luis Balcázar for many reasons. He put my attention, during the Boolean Complexity course, on problems about recovering decision trees arising from Blum's paper, he always encouraged me to pursue my ideas holding with me some useful discussions and has revised partial versions of this work during the time. In particular I would underline that the idea of the algorithm of Section 4 to recover decision trees, come up to him during one of our discussions.

Also I would to thank Maria Luisa Bonet for reading with me the final version and Juan Luis Esteban.

Finally I would to thank the Referees for their very useful comments, in particular that one that suggested me a new vision of the tree-like decision lists arising from the new proof of the Blum's containement Theorem.

REFERENCES

1. A. Blum, Rank-r decision trees are a subclass of r-decision lists, *Information Processing Letters*, 1992, *42*, pp. 183-185.
2. A. Ehrenfeucht and D. Haussler, Learning decision trees from random examples, *Information and Computation*, 1989, *82*, pp. 231-246.
3. R. L. Rivest, Learning decision lists, *Machine Learning*, 1987, *2*, pp. 223-246.