The expressive power of first-order logical decision trees

Joris J.M. Gillis[⋆] and Jan Van den Bussche

Hasselt University and transnational University of Limburg

Abstract. This paper characterises the expressive power of first-order logical decision trees (FOLDTs) as a fragment of first-order logic. Specifically, using FOLDTs one can express precisely the boolean combinations of existential formulas. FOLDTs are also slightly generalized to formally allow for output variables.

1 Introduction

In logical and relational learning [4], the logical languages that can be learned most effectively offer rather limited expressiveness, typically not going beyond the existential fragment of first-order logic. Indeed, this is the standard balancing exercise between expressive power and efficiency that one faces everywhere in the fields of AI and computer science. First-order logical decision trees (FOLDTs) [1] are one of the few logical languages used in ILP that offer higher expressive power, yet can still be learned effectively (cf. the Tilde system, part of the ACE-ilProlog system [2]). FOLDTs allow the expression of certain properties involving universal quantification in a natural and direct manner. For example, consider the vocabulary with a binary relation symbol E, used to indicate the edges of a directed graph, and unary relation symbols R and B, used to indicate the "red" and the "blue" nodes in the graph. Then the very simple FOLDT shown in Fig. 1 expresses the property that every blue node has edges to all red nodes, expressed in first-order logic as

$$\forall x(B(x) \to \forall y(R(y) \to E(x,y))).$$

A natural question now, which has remained unanswered in the literature so far, is, *exactly* which properties can be expressed by FOLDTs? Blockeel and De Raedt [1] have given a translation of FOLDTs into first-order logic (FOL), but exactly which fragment of FOL do we cover by FOLDTs? In the present paper we answer the question and show the equivalence between FOLDTs and the fragment of FOL formed by all *boolean combinations of existential formulas*. For example, the above formula can be rewritten as the *negation* of an existential formula:

$$\neg\exists x\exists y(B(x) \land R(x) \land \neg E(x,y))$$

[⋆] Ph.D. Fellow of the Research Foundation Flanders (FWO).

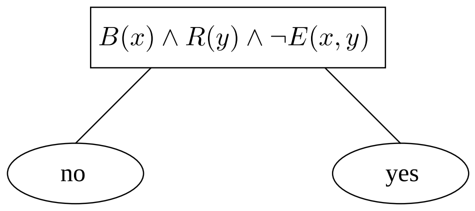

Fig. 1. Example of a FOLDT.

and indeed this way of expression closely matches the FOLDT of Fig. 1.

Our result implies that properties whose expression require the alternation of quantifiers in an essential way are not expressible as a FOLDT. A typical example of such a property is "there exists a blue node with edges to all red nodes", or in FOL,

$$\exists x(B(x) \wedge \forall y(R(y) \rightarrow E(x,y))).$$

We will work with a slight generalization of FOLDTs, in comparison to their original definition, so that not only boolean queries but also k-ary queries can be expressed for $k > 0$. In the context of full FOL, the distinction between boolean and k-ary queries is not essential. In contrast, we will see that a k-ary query may be expressible by a FOLDT, while the corresponding boolean nonemptiness query is not.

2 Preliminaries

To avoid misunderstanding, we fix terminology and notation for some well-known notions from logic. A *relational vocabulary* is a set τ of relation symbols, each with an associated arity (a natural number). A *τ-interpretation* I consists of a nonempty set $\mathrm{dom}(I)$, called the domain of I, and a k-ary relation R^I on $\mathrm{dom}(I)$, for each $R \in \tau$, with k the arity of R.

A *boolean query* over τ is a function Q from the set of τ-interpretations to the two-element set $\{\text{yes}, \text{no}\}$. In the most basic classification setting of learning from interpretations, the learner is provided with some yes- and some no-instances of a boolean query Q, and must infer a classifier, i.e., an expression for Q. Often this expression can be translated in a first-order logic (FOL) sentence; this is the case, for example, with classifiers in the form of recursion-free Prolog programs. A FOL sentence over τ is a FOL formula without free variables and involving only the relation symbols from τ, besides the equality symbol. The boolean query Q expressed by such a sentence φ is defined as follows: for every τ-interpretation I, we have that $Q(I) = \text{yes}$ if and only if $I \models \varphi$. Here, $I \models \varphi$ denotes that φ is true in I.

For any natural number k, a *k-ary query* over τ is a function Q that maps each τ-interpretation I to a k-ary relation on $\mathrm{dom}(I)$. A classical example would be where τ consists of the binary relation symbol *Parent*, and Q, given a concrete parent relation, outputs the corresponding grandparent relation. The relevance of k-ary queries for relational learning lies in the setting of predicate description. Here, the learner is provided with examples, where each example consists of a τ-interpretation I and some positive and negative Q-facts (e.g., a parent relation, some pairs in the grandparent relation; and some pairs not in the grandparent relation); again the task is to come up with an expression for Q.

For a FOL formula $\varphi(x_1, \dots, x_k)$ with free variables among $x_1, \dots, x_k$, the k-ary query Q expressed by the expression $\{(x_1, \dots, x_k) \mid \varphi\}$ is defined as follows: for every τ-interpretation I, we have that

$$Q(I) = \{(a_1, \dots, a_k) \in \mathrm{dom}(I)^k \mid I \models \varphi[x_1/a_1, \dots, x_k/a_k]\}$$

where $I \models \varphi[x_1/a_1, \dots, x_k/a_k]$ denotes that φ becomes true in I if we substitute free variable x_i by value a_i for $i = 1, \dots, k$. For example, the grandparent query can be expressed as

$$\{(x, y) \mid \exists z(\mathit{Parent}(x, z) \wedge \mathit{Parent}(z, y))\}.$$

Boolean queries are just a special kind of k-ary queries. Indeed, a boolean query can be thought of as a 0-ary query by identifying the nonempty nullary relation $\{()\}$ with 'yes' and the empty nullary relation with 'no'. The class of queries expressed by FOL formulas is called the class of *first-order queries.*

3 First-Order Logical Decision Trees

Fix a relational vocabulary τ. A *first-order logical decision tree (FOLDT)* over τ is a triple $(T, \lambda, \bar{x})$, such that

- T is a finite binary tree;
- λ is a labeling function on the nodes of T such that each internal node (including the root) is labeled with a conjunction of literals over τ, and each leaf node is labeled with 'yes' or 'no'; the label of node n is denoted by λ_n;
- $\bar{x}$ is a tuple of distinct variables, called the *output variables.*[1]

When no confusion can arise, we will refer to the FOLDT simply as T and leave λ and $\bar{x}$ implicit.

An example of a FOLDT over the vocabulary consisting of the two binary relation names R and S, is shown in Figure 2. In figures we use the convention of underlining the output variables; in this case, there is only one output variable, namely x.

[1] The original definition of FOLDTs did not have output variables; we will come back to this feature later. Also, in the original definition, leaf nodes can take labels from any finite set of class names rather than just 'yes' and 'no'. If we are interested in the expressive power of FOLDTs in their capacity of describing any one particular class, we can identify that class with 'yes' and the other classes with 'no'. Finally, the original definition requires that a variable introduced in some node must not be used in the right subtree of that node. This requirement guarantees that the first-order logic translation of the FOLDT does not reuse variables, and makes the FOLDT more easy to understand, but is otherwise inessential.

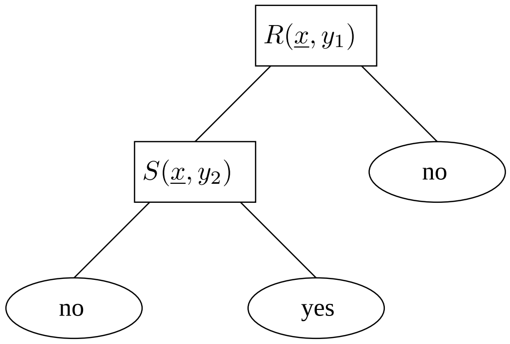

Fig. 2. FOLDT expressing the first-order query $\{x \mid \exists y\, R(x,y) \wedge \neg\exists y\, S(x,y)\}$.

A FOLDT T expresses a k-ary first-order query Q_T, where k equals the arity of the tuple $\bar{x}$ of output variables. The query Q_T is defined by translating T into a FOL formula $\Phi_T(\bar{x})$, then defining Q_T as the query expressed as $\{\bar{x} \mid \Phi_T\}$. So it remains to define Φ_T. We do this in three steps.

1. We first define formulas α_n for every node n of T:
$$\alpha_n := \begin{cases} \text{true} & \text{if } n \text{ is the root of } T; \\ \alpha_p \wedge \lambda_p & \text{if } n \text{ is the left child of node } p; \\ \alpha_p \wedge \neg\exists\bar{y}(\alpha_p \wedge \lambda_p) & \text{if } n \text{ is the right child of node } p. \end{cases}$$
where $\bar{y}$ is the set of free variables in $\alpha_p \wedge \lambda_p$ that are not output variables.
2. For each node n we next define β_n as the formula $\exists\bar{z}(\alpha_n)$, where $\bar{z}$ is the set of free variables in α_n that are not output variables.
3. Finally, we define
$$\Phi_T := \bigvee\{\beta_\ell \mid \ell \text{ is a leaf node labeled yes}\}.$$

It is readily verified that this definition of the semantics of FOLDTs conforms to the original definition given by Blockeel and De Raedt [1, Fig. 2], at least when the set of output variables is empty.

Example 1. Consider the FOLDT T of Fig. 2. Let us number the root, the left child of the root, and the only leaf node labeled yes, as 1, 2, and 3, respectively. Then

$$\begin{aligned} \alpha_1 &= \text{true} \\ \alpha_2 &= \text{true} \wedge R(x,y_1) \equiv R(x,y_1) \\ \alpha_3 &= R(x,y_1) \wedge \neg\exists y_1 \exists y_2 (R(x,y_1) \wedge S(x,y_2)) \end{aligned}$$

and we obtain $\Phi_T = \exists y_1(\alpha_3)$ which can be simplified to

$$\exists y_1\, R(x,y_1) \wedge \neg\exists y_2\, S(x,y_2).$$

Note that if we would modify the FOLDT by using the same variable y instead of y_1 and y_2, we would obtain a different meaning, namely, $\exists y\, R(x,y) \wedge \neg\exists y(R(x,y) \wedge S(x,y))$.

4 The expressive power of FOLDTs

Our main result characterizes the expressive power of FOLDTs as follows.

Theorem 1. *A k-ary query is expressible by a FOLDT if and only if it is expressible by a boolean combination of existential FOL formulas.*

Here, an *existential* FOL formula is of the form $\exists\bar{y}\, \psi(\bar{x},\bar{y})$, where ψ is quantifier-free. In other words, a formula is existential if it can be written such that all the quantifiers are in front (prenex normal form) and are existential. Boolean combinations are then built up from existential formulas using conjunction, disjunction, and negation, but no further quantification. The existential fragment of FOL is usually denoted by Σ_1, and we denote the class of boolean combinations of Σ_1-formulas by $BC(\Sigma_1)$.

We omit the proof of the theorem in this extended abstract, but provide some comments on the issues involved. First, the reader should not be lulled into interpreting our theorem as merely stating that the FOL translation Φ_T of a FOLDT T is in $BC(\Sigma_1)$; in fact, it is not. Indeed, the gist of the proof of the *only-if* direction consists of showing that Φ_T, for any FOLDT, can always be simplified into an equivalent $BC(\Sigma_1)$ formula. Example 1 already gave an example of this simplification.

The if-direction of the theorem follows from three basic constructions:

1. *Every Σ_1-expressible query is expressible by a FOLDT.* Indeed, consider a Σ_1-formula $\varphi(\bar{x})$ of the form $\exists\bar{y}\,\psi(\bar{x},\bar{y})$. We can put ψ in DNF as $\gamma_1 \vee \cdots \vee \gamma_\ell$. We construct a FOLDT for φ as follows. The root is labeled γ_1. From the root descends a chain of right children, labeled γ_2 until γ_ℓ. Every node on this linear chain, including the root, gets as left child a leaf labeled yes. Finally, the rightmost node on the chain (the one labeled with γ_ℓ) gets as right child a leaf labeled no. The set of output variables is $\bar{x}$.
2. *The conjunction of two FOLDT-expressible queries is FOLDT-expressible.* Indeed, if we have two FOLDTs T_1 and T_2 then we can form their conjunction by attaching a copy of T_2 at every leaf node of T_1 labeled yes. This construction is only correct if we make sure in advance (without loss of generality) that T_1 and T_2 have disjoint sets of non-output variables. The output variables of the resulting FOLDT are the union of those of T_1 and T_2.
3. *The negation of a FOLDT-expressible query is FOLDT-expressible.* Indeed, to negate a FOLDT it suffices to swap the leaf labels yes and no.

It can be proven that the above three constructions are correct (proof omitted).

5 Discussion

Our result places the expressive power of FOLDTs at a rather low position in the quantifier alternation hierarchy for first-order logic [3]. We have already seen Σ_1 as the existential fragment of first-order logic. The next level in this hierarchy is Σ_2, consisting of all formulas that can be put in prenex form with a quantifier prefix of the form $\exists^*\forall^*$. Similarly, Π_2 consists of the $\forall^*\exists^*$ formulas. It is easy to see that $BC(\Sigma_1)$ formulas can be put both in Σ_2 form and in Π_2 form. This places the FOLDT-expressible queries in the class known as Δ_2: the queries expressible both by a Σ_2-formula and by a Π_2-formula.

Now it is known that there are queries expressible in Π_2 but not in Σ_2, and vice versa, even in restriction to finite interpretations [3]. Any such queries are not FOLDT-expressible. For example, the boolean query mentioned in the Introduction "there exists a blue node with edges to all red nodes", $\exists x \forall y (B(x) \wedge (R(y) \rightarrow E(x,y)))$, is a typical example of a query expressible in Σ_2 but not in Π_2, and, consequently, not as a FOLDT. Likewise, the boolean query "all blue nodes have an edge to some red node", $\forall x \exists y (B(x) \rightarrow (R(y) \wedge E(x,y)))$ is in Π_2 but not in Σ_2 and hence again not FOLDT-expressible.

A somewhat remarkable consequence is that a k-ary query may be FOLDT-expressible, while the corresponding boolean nonemptiness query is not. For example, consider the unary query Q: "list all blue nodes that have edges to all red nodes". This query is Π_1-expressible as $\{x \mid B(x) \wedge \forall y (R(y) \rightarrow E(x,y))\}$ and hence expressible by a FOLDT with output variable x. The boolean query $Q \neq \emptyset$ however, amounts to $\exists x (B(x) \wedge \forall y (R(y) \rightarrow E(x,y)))$ which we have just seen to be not FOLDT-expressible.

Acknowledgment

We thank Hendrik Blockeel and Jan Struyf for interesting discussions and help with the Tilde system.

References

1. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1–2), 285–297 (1998)
2. Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., Vandecasteele, H.: Improving the efficiency of inductive logic programming through the use of query packs. Journal of Artificial Intelligence Research 16, 135–166 (2002)
3. Chandra, A., Harel, D.: Structure and complexity of relational queries. J. Comput. Syst. Sci. 25, 99–128 (1982)
4. De Raedt, L.: Logical and Relational Learning. Springer (2008)