Expressive Power of Safe First-Order Logical Decision Trees

Joris J.M. Gillis* and Jan Van den Bussche

Hasselt University and transnational University of Limburg

Abstract. This paper characterizes the expressive power of a subclass of first-order logical decision trees (FOLDTs) as a fragment of first-order logic. Specifically, using *safe FOLDTs* one can express precisely the *boolean combinations of safe existential sentences.*

1 Introduction

In Logical and Relational Learning [5], 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 [2,3]). 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

$$(\forall x)(B(x) \rightarrow (\forall y)(R(y) \rightarrow E(x, y))). \qquad (*)$$

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 recursion-free Prolog which can be equivalently expressed in first-order logic (FOL). However, exactly which fragment of FOL do we cover by FOLDTs? In the present paper we answer the question for a subclass of FOLDTs, called the *safe FOLDTs* and show the equivalence between safe FOLDTs and the fragment of FOL formed by all *safe boolean combinations of existential sentences.* For example, the above formula can be rewritten as the *negation* of an existential sentence:

$$\neg(\exists x, y)(B(x) \wedge R(y) \wedge \neg E(x, y)) \qquad (\dagger)$$

* Ph.D. Fellow of the Research Foundation Flanders (FWO).

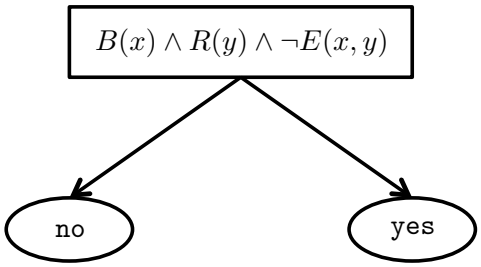

Fig. 1. Example of a FOLDT.

This existential sentence is safe because variables x and y, which occur in the negative literal $\neg E(x, y)$, also occur in a positive literal ($B(x)$ and $R(y)$ respectively). A formal definition of safety will be provided later in the paper.

Note that formula (†) 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 safe FOLDT. A typical example of such a property is "there exists a blue node with edges to all red nodes", or in FOL,

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

It is interesting to put our result in the perspective of the logical learning model by Osherson and Weinstein [7,8,9]; see also [6]. Indeed, there it has been shown that a query is learnable in the limit if and only if it is Δ_2-expressible. Since our result places safe FOLDTs in Δ_2, as we will see in section 5, our result may help to explain why FOLDTs are effectively learnable (in the limit).

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 $\{\mathrm{yes}, \mathrm{no}\}$. 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. 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. Recall that a *literal* is an atomic formula with relation name from τ, or the negation of such an atomic formula. A *first-order logical decision tree (FOLDT)* over τ is a couple (T, λ), such that

- T is a finite binary tree; and
- 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.

When no confusion can arise, we will refer to the FOLDT simply as T and leave λ implicit.

An example of a FOLDT over the vocabulary consisting of the two binary relation names R and S, is shown in Figure 2.

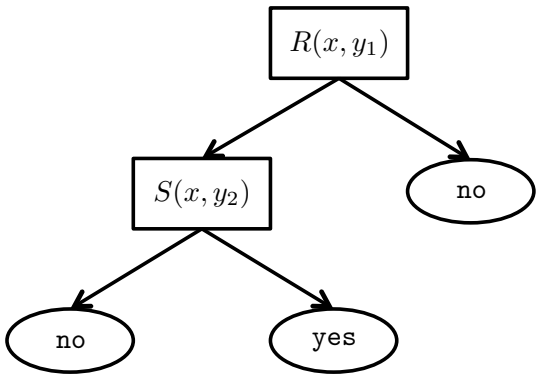

Fig. 2. FOLDT expressing the first-order query $\exists x\big(\exists y_1\, R(x, y) \wedge \neg\exists y_2\, S(x, y_2)\big)$.

The semantics of FOLDTs has been defined by a translation into recursion-free Prolog [1]. The resulting Prolog programs use negation by failure which sometimes gives non-declarative results. Consider, for example, the simple FOLDT consisting of a root with two leaves as children. The left child is labeled `yes`, the right child is labeled `no`. The root is labeled $\neg R(x)$. Then the semantics of this FOLDT is equivalent to $\neg(\exists x)R(x)$. In contrast, if the root would be labeled $S(x) \wedge \neg R(x)$, the semantics would become $(\exists x)(S(x) \wedge \neg R(x))$. Moreover, if the root would be labeled $\neg R(x) \wedge S(x)$, the semantics would be $\neg(\exists x)(R(x)) \wedge (\exists x)(S(x))$.

We next define a more declarative semantics for FOLDTs, which is equivalent to the original Prolog semantics in the case the FOLDTs are safe. We will define safety below, but first we already give the semantics.

A FOLDT T expresses a Boolean first-order query Q_T. The query Q_T is defined by translating T into a FOL sentence Φ_T.

1. We first define formulas α_n for every node n of T:

$$\alpha_n := \begin{cases} \texttt{true} & \text{if } n \text{ is the root of } T; \\ \alpha_p \wedge \lambda_p & \text{if } n \text{ is the left child of node } p; \\ \alpha_p \wedge \neg(\exists \overline{y})(\alpha_p \wedge \lambda_p) & \text{if } n \text{ is the right child of node } p. \end{cases}$$

   where $\overline{y}$ is the set of free variables in $\alpha_p \wedge \lambda_p$.
2. For each node n we next define β_n as the formula $(\exists \overline{z})(\alpha_n)$, where $\overline{z}$ is the set of free variables in α_n.
3. Finally, we define

$$\Phi_T := \bigvee \{\beta_\ell \mid \ell \text{ is a leaf node labeled yes}\}.$$

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

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

and we obtain $\Phi_T = (\exists x, y_1)(\alpha_3)$:

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

3.1 Safety

Let n be a node of a FOLDT T. Let $A_n = (m_1, \ldots, m_k)$ be the sequence of nodes from the root $r = m_1$ to node n (m_k is the parent of n and m_{k+1} denotes n). The *leftish ancestors* of n are the nodes m_i, $1 \leq i \leq k$, in A_n such that m_{i+1} is the left child of m_i.

Example 2. The fragment of a FOLDT depicted in Fig. 3 has three nodes of interest. The root, node 1, has no leftish ancestors, because it is the root. Node 2 has the root as sole leftish ancestors. The root is the sole leftish ancestor of node 3. Indeed, $A_3 = (1, 2, 3)$, node 2 is a left-child of 1 and thus is 1 a leftish ancestor.

The conjunction λ_n, labeling node n, is a sequence of literals. The *local predecessors* of the occurrence of a literal ξ in λ_n is the set of literals syntactically preceding ξ in λ_n. The *global predecessors* of ξ are the literals of the leftish

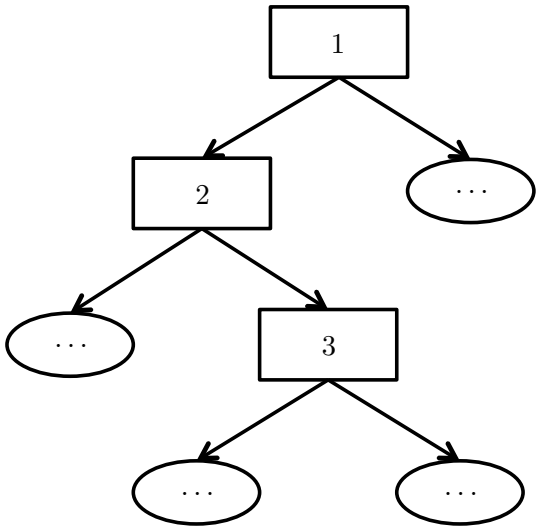

Fig. 3. Fragment of a FOLDT with a complex left-ancestor relation.

ancestors. The union of the local and global predecessors is called the set of *predecessors* of ξ. The *positive predecessors* of ξ are the positive literals in the set of predecessors of the literal. A variable in an occurrence of a negative literal ξ is *covered* if it occurs in a positive predecessor of ξ, otherwise it is *open*. Variables in an occurrence of a positive literal are always covered. The basic building blocks of the node labelings are relation symbols, which are inherently safe.

Example 3. In the FOLDT in Fig. 4 variable y in node 2 is covered by the occurrence of x in $R(x, y)$ in node 1. The variable x in $\neg T(x, y)$, in node 3, is covered locally by $S(x)$ in this same conjunction. On the other hand, the variable y is not covered in node 3, because the root is not a leftish ancestor of node 3 and y does not occur locally.

The definition of a FOLDT allows both covered and open variables. A *FOLDT T is safe*, if all occurrences of variables are covered. It is readily verified that our definition of the semantics of FOLDTs conforms to the original definition given by Blockeel and De Raedt [1, Fig. 2], at least when the FOLDT is safe.

4 The expressive power of safe FOLDTs

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

Theorem 1. *A Boolean query is expressible by a* safe *FOLDT if and only if it is expressible by a boolean combination of safe existential FOL sentences.*

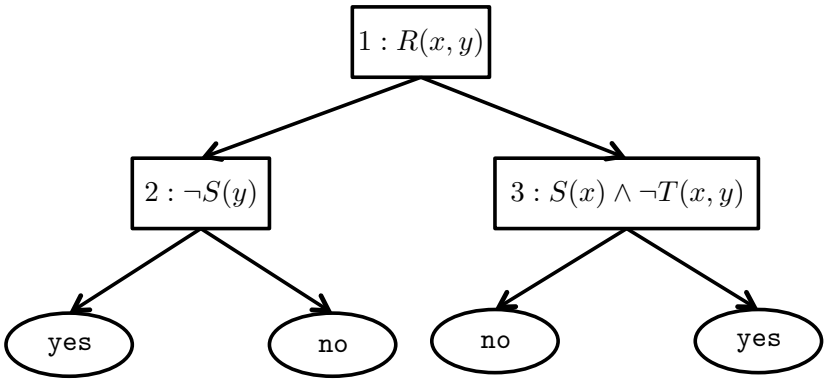

Fig. 4. The occurrences of y in nodes 2 and 3 are not covered.

Here, an *existential FOL formula* is of the form $(\exists \overline{y})\psi(\overline{x}, \overline{y})$, where ψ is quantifier-free and $\overline{x}$ and $\overline{y}$ are sets of variables. 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. An *existential FOL sentence* is an existential FOL formula without free variables. Boolean combinations are then built up from existential sentences using conjunction, disjunction, and negation. The existential fragment of FOL is usually denoted by Σ_1 [4]. We denote the class of boolean combinations of Σ_1*-sentences* by $BC(\Sigma_1)$. The class $BC(\Sigma_1)$ is thus a subset of the class of boolean combinations of Σ_1*-formulas*.

The notion of safety for $BC(\Sigma_1)$-sentences is defined in the following standard way [10]. Let χ be a $BC(\Sigma_1)$-sentence. For χ to be safe every Σ_1-sentence in χ must be safe, defined as follows. Let $\varphi \equiv (\exists \overline{x})\psi(\overline{x})$ be an existential sentence where ψ is quantifier-free. For φ to be safe, we require ψ to be in DNF, and furthermore we require that in every disjunct of ψ, every variable is "covered". Here, a variable is covered if it occurs in a positive literal.

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, shown in Lemma 1, consists of showing that Φ_T, for any safe FOLDT, can always be simplified into an equivalent $BC(\Sigma_1)$-sentence. Example 1 already gave an example of this simplification. The *if* direction of the theorem is proven by Lemmas 2, 3, 4, 5 and 6.

Lemma 1. *Let T be a safe FOLDT. Then Φ_T can be equivalently expressed as a boolean combination of safe Σ_1-sentences.*

Proof. Firstly, we show the conversion of Φ_T into a $BC(\Sigma_1)$ sentence. Secondly, it is proven that if T is safe, then Φ_T is also safe.

Φ_T can be equivalently expressed as a $BC(\Sigma_1)$ sentence: First, we prove that for any node n in T, α_n can be written in the form:

$$\alpha_n(\overline{x}) \equiv \psi_n(\overline{x}) \wedge \phi_n$$

where ψ_n is a conjunction of literals (in particular, ψ_n is quantifier-free) and ϕ_n is a conjunction of negated Σ_1-sentences (in particular, ϕ_n is in $BC(\Sigma_1)$). We prove this by induction on the position of n in T.

Base case: n is the root of T Trivial, because $\alpha_n \equiv$ `true`, so we can choose `true` both for ψ_n and ϕ_n.

Induction: n is a descendant of the root Let p be the parent node of n. The induction hypothesis states that $\alpha_p(\overline{x}) \equiv \psi_p(\overline{x}) \wedge \phi_p$. Node n is either the left or the right child of p. Let $\overline{y}$ be the variables in λ_p that do not yet belong to $\overline{x}$. Let $\overline{x}' = \overline{x} \cup \overline{y}$.

If n is the left child of p, we have:

$$\begin{aligned}\alpha_n(\overline{x}') &\equiv \alpha_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y}) \\ &\equiv [\lambda_p(\overline{x}, \overline{y}) \wedge \psi_p(\overline{x})] \wedge \phi_p\end{aligned}$$

So we can choose $\psi_n(\overline{x}') \equiv \lambda_p(\overline{x}, \overline{y}) \wedge \psi_p(\overline{x})$ and $\phi_n \equiv \phi_p$.

If n is the right child of p, we have:

$$\begin{aligned}\alpha_n(\overline{x}) &\equiv \alpha_p(\overline{x}) \wedge \neg(\exists \overline{x}, \overline{y})\big[\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y})\big] \\ &\equiv \psi_p(\overline{x}) \wedge \phi_p \wedge \neg(\exists \overline{x}, \overline{y})\big[\psi_p(\overline{x}) \wedge \phi_p \wedge \lambda_p(\overline{x}, \overline{y})\big] \\ &\equiv \psi_p(\overline{x}) \wedge \Big(\phi_p \wedge \big(\neg\phi_p \vee \neg(\exists \overline{x}, \overline{y})\big[\psi_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y})\big]\big)\Big) \\ &\equiv \psi_p(\overline{x}) \wedge \Big(\phi_p \wedge \neg(\exists \overline{x}, \overline{y})\big[\psi_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y})\big]\Big)\end{aligned}$$

So we can choose $\psi_n(\overline{x}) \equiv \psi_p(\overline{x})$ and $\phi_n \equiv \phi_p \wedge \neg(\exists \overline{x}, \overline{y})\big[\psi_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y})\big]$, because both ψ_p and λ_p are quantifier-free.

Secondly, the β formula of any node n in FOLDT T can now be converted into a $BC(\Sigma_1)$ formula as follows:

$$\begin{aligned}\beta_n &\equiv (\exists \overline{x})\alpha_n(\overline{x}) \\ &\equiv \big((\exists \overline{x})\psi_n(\overline{x})\big) \wedge \phi_n\end{aligned}$$

Finally, we know that Φ_T is the disjunction of the β sentences of the `yes`-labeled leaves. Thus Φ_T is a $BC(\Sigma_1)$ sentence; note that Φ_T is in DNF.

T is safe, then Φ_T is safe: If T is safe, each occurrence of a variable if covered, either locally or by a leftish ancestor. Note in the recursive definition of the α formula, that the conjunctions of all leftish ancestors of a node are combined into a single conjunction, thus constructing a safe existential sentence.

□

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

1. *The conjunction of two safe FOLDT-expressible queries is safe FOLDT-expressible.* Indeed, if we have two safe 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.
2. *The negation of a safe FOLDT-expressible query is safe FOLDT-expressible.* Indeed, to negate a safe FOLDT it suffices to swap the leaf labels `yes` and `no`.
3. *Every safe Σ_1-expressible query is expressible by a safe FOLDT.* Indeed, consider a safe Σ_1-sentence σ of the form $(\exists\overline{x})\psi(\overline{x})$. We have ψ in DNF: $\gamma_1 \vee \cdots \vee \gamma_\ell$. We construct a FOLDT for σ as depicted in Fig. 5. 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`.

We prove in Lemmas 4, 5 and 6 that the above constructions are correct. First we prove two auxiliary lemmas.

Lemma 2. *Let T be a FOLDT, let p be a node in T and let n and m be the respective left- and right child of p. Then:*

$$\beta_n \vee \beta_m \equiv \beta_p$$

Proof.

$$\begin{aligned}
& \beta_n \vee \beta_m \\
\equiv\ & (\exists\overline{x},\overline{y})\Big(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x},\overline{y}))\Big) \vee \\
& \Big((\exists\overline{x})\alpha_p(\overline{x}) \wedge \neg(\exists\overline{x},\overline{y})\big(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x},\overline{y})\big)\Big) && (1)\\
\equiv\ & \Big((\exists\overline{x})\alpha_p(\overline{x}) \vee (\exists\overline{x},\overline{y})(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x},\overline{y}))\Big) \wedge \\
& \Big((\exists\overline{x},\overline{y})(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x},\overline{y})) \vee \neg(\exists\overline{x},\overline{y})(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x},\overline{y}))\Big) && (2)\\
\equiv\ & (\exists\overline{x})\alpha_p(\overline{x}) && (3)
\end{aligned}$$

Here, $\overline{y}$ is the set of free variables in λ_p. We take the following steps in formula:

1. Replacing the β-formulas by their definition.
2. Distributing the first disjunct over the two parts of the second disjunct.
3. In the first conjunct of 2, the second part is a specialization of the first part, thus we may drop the second part. In the second conjunct, a tautology has been created.

□

Lemma 3. *Let T be a FOLDT, let p be a node in T and let n and m be the respective left- and right child of p. Then:*

$$\beta_n \wedge \beta_m \equiv \texttt{false}$$

Proof. Let $\overline{x}$ be the set of free variables in α_p and let $\overline{y}$ be the free variables in λ_p that are not in $\overline{x}$.

$$\begin{aligned} & \beta_n \wedge \beta_m \\ \equiv \quad & (\exists \overline{x}, \overline{y})\Big(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y})\Big) \wedge \\ & (\exists \overline{x})\alpha_p(\overline{x}) \wedge \neg(\exists \overline{x}, \overline{y})\Big(\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x}, \overline{y})\Big) \\ \equiv \quad & \texttt{false} \end{aligned}$$

□

Lemma 4. *Let T_1 and T_2 be two safe FOLDTs. Suppose, without loss of generality, that the sets of variables in T_1 and T_2 are disjoint. The conjunction construction is safe and correct.*

Proof. Let T be the FOLDT constructed to express the conjunction of FOLDTs T_1 and T_2.

If both T_1 and T_2 are safe, then so is T. Let n be an internal node of T. It is easily verified that all occurrences of variables in the conjunction labeling n, λ_n, are covered. If an occurrence of a variable is not covered, it would also be "uncovered" in T_1 or T_2, because the structure and labeling are preserved.

To prove the correctness, we need to show that:

$$\begin{aligned} \bigvee_{\texttt{yes}-\text{leaf } \ell \text{ in T}} \beta_\ell &\equiv \Phi_{T_1} \wedge \Phi_{T_2} \\ &\equiv \bigvee_{\texttt{yes}-\text{leaf } \ell_1 \text{ in } T_1} \beta_{\ell_1}^{T_1} \wedge \bigvee_{\texttt{yes}-\text{leaf } \ell_2 \text{ in } T_2} \beta_{\ell_2}^{T_2} \\ &\equiv \bigvee_{\texttt{yes}-\text{leaf } \ell_1 \text{ in } T_1,\ \ell_2 \text{ in } T_2} \beta_{\ell_1}^{T_1} \wedge \beta_{\ell_2}^{T_2} \end{aligned}$$

Let ℓ be a `yes`-leaf of T. Leaf ℓ corresponds to a `yes`-leaf ℓ_2 from a copy of T_2. The copy of T_2 replaces a `yes`-leaf ℓ_1 from T_1. Thus we denote ℓ as the couple (ℓ_1, ℓ_2). We have that $\beta_{\ell_1}^{T_1} \equiv (\exists \overline{y}_1)\alpha_{\ell_1}^{T_1}(\overline{x}_1, \overline{y}_1)$, where $\overline{y}_1$ is the set of free variables in $\alpha_{\ell_1}^{T_1}$. We need to prove that

$$\beta_\ell^T \equiv \beta_{\ell_1}^{T_1} \wedge \beta_{\ell_2}^{T_2} \ .$$

To this end, we show that the α-formula of each node n, equal to ℓ_1 or a descendent of ℓ_1, can be converted to the form:

$$\alpha_n(\overline{x}) \equiv \alpha_{\ell_1}^{T_1}(\overline{x}_1) \wedge \big(\psi_n(\overline{x}_2) \wedge \phi_n\big)$$

Here, $\overline{x}_1$ is the set of free variables in $\alpha^{T_1}_{\ell_1}$, $\overline{x}_2$ is the set of free variables in ψ_n, $\overline{x}$ is the union of $\overline{x}_1$ and $\overline{x}_2$, ψ_n is a conjunction of literals (in particular, ψ_n is quantifier-free), ϕ_n is a conjunction of negated Σ_1-formulas (in particular, ϕ_n is in $BC(\Sigma_1)$) and $\psi_n \wedge \phi_n \equiv \alpha^{T_2}_{\ell_2}$. This last property can be verified in the same way as in the proof of Lemma 1.

If $n = \ell_1$, the claim holds trivially, as we can choose $\psi_n \equiv \texttt{true} \equiv \phi_n$.

If $n \neq \ell_1$, let p be the parent node of n, by the induction hypothesis we know:

$$\alpha_p(\overline{x}) \equiv \alpha^{T_1}_{\ell_1}(\overline{x}_1) \wedge \big(\psi_n(\overline{x}_2) \wedge \phi_n\big)$$

There are two cases:

1. Node n is the left child of p: Let $\overline{y}$ be the set of free variables in λ_p that are not in $\overline{x}_2$

$$\begin{aligned}\alpha_n(\overline{x}) &\equiv \alpha_p(\overline{x}') \wedge \lambda_p(\overline{x}_2, \overline{y}) \\ &\equiv \alpha^{T_1}_{\ell_1}(\overline{x}_1) \wedge \Big(\big(\psi_p(\overline{x}_2) \wedge \lambda_p(\overline{x}_2, \overline{y})\big) \wedge \phi_p\Big)\end{aligned}$$

   So we can choose $\psi_n(\overline{x}'_2) \equiv \psi_p(\overline{x}_2) \wedge \lambda_p(\overline{x}_2, \overline{y})$ with $\overline{x}'_2 = \overline{x}_2 \cup \overline{y}$ and $\phi_n \equiv \phi_p$.
2. Node n is the right child of p:

$$\begin{aligned}\alpha_n(\overline{x}) &\equiv \alpha_p(\overline{x}') \wedge \neg(\exists \overline{x}', \overline{y})\big[\alpha_p(\overline{x}) \wedge \lambda_p(\overline{x}_2, \overline{y})\big] \\ &\equiv \alpha^{T_1}_{\ell_1}(\overline{x}_1) \wedge \psi_p(\overline{x}_2) \wedge \phi_p \wedge \\ &\quad \neg\Big((\exists \overline{x}_1)\big[\alpha^{T_1}_{\ell_1}(\overline{x}_1)\big] \wedge (\exists \overline{x}_2, \overline{y})\big[\psi_p(\overline{x}_2) \wedge \lambda_p(\overline{x}_2, \overline{y})\big] \wedge \phi_p\Big) \\ &\equiv \alpha^{T_1}_{\ell_1}(\overline{x}_1) \wedge \psi_p(\overline{x}_2) \wedge \phi_p \wedge \\ &\quad \Big(\neg\phi_p \vee \neg(\exists \overline{x}_1)\alpha^{T_1}_{\ell_1}(\overline{x}_1) \vee \neg(\exists \overline{x}_2, \overline{y})\big[\psi_p(\overline{x}_2) \wedge \lambda_p(\overline{x}_2, \overline{y})\big]\Big) \\ &\equiv \alpha^{T_1}_{\ell_1}(\overline{x}_1) \wedge \big(\psi_p(\overline{x}_2) \wedge \phi_p\big) \wedge \\ &\quad \Big(\neg(\exists \overline{x}_1)\alpha^{T_1}_{\ell_1}(\overline{x}_1) \vee \neg(\exists \overline{x}_2, \overline{y})\big[\psi_p(\overline{x}_2) \wedge \lambda_p(\overline{x}_2, \overline{y})\big]\Big) \qquad (4)\end{aligned}$$

   In formula 4 we encounter an interesting situation. If $\neg(\exists \overline{x}_1)\alpha^{T_1}_{\ell_1}(\overline{x}_1)$ evaluates to `true`, the first conjunct $(\alpha^{T_1}_{\ell_1}(\overline{x}_1))$ will never evaluate to `true` and thereby the formula will evaluate to `false`. However, if $\neg(\exists \overline{x}_1)\alpha^{T_1}_{\ell_1}(\overline{x}_1)$ would evaluate to `true`, we could not have arrived at ℓ_1. Obviously, as we are now in a descendant of ℓ_1, this is a contradiction. The subformula can thus be dropped from the disjunction, so we can choose:

$$\begin{aligned}\psi_n(\overline{x}_2) &\equiv \psi_p(\overline{x}_2) \\ \phi_n &\equiv \phi_p \wedge \neg(\exists \overline{x}_2, \overline{y})\big[\psi_p(\overline{x}_2) \wedge \lambda_p(\overline{x}_2, \overline{y})\big]\end{aligned}$$

   Hereby the statement is proven correct.

□

Lemma 5. *Let T be a safe FOLDT. The negation construction is safe and correct.*

Proof. A FOLDT is safe if all occurrences of variables are covered. The difference between T and $T_\neg$ are the leafs labels. The leafs are not labeled with conjunctions of literals and the structure is not changed. Therefore, $T_\neg$ is also safe.

Lemmas 2 and 3 state that if a certain node evaluates to `true`, *exactly one* one leaf descending from that node also evaluates to `true`. By definition, we know that $\beta_r \equiv$ `true`, where r is the root node of T. As a result, for any interpretation, exactly one leaf evaluates to `true` whereas all others will evaluate to `false`. Clearly, by inverting the leaf labels, all interpretations accepted by T are rejected by $T_\neg$ and vice versa. □

Lemma 6. *Let T_σ be the tree constructed as above to express the safe Σ_1-sentence $\sigma \equiv (\exists \overline{x})(\gamma_1(\overline{x}_1) \vee \ldots \vee \gamma_\ell(\overline{x}_\ell))$. Then T_σ is a safe FOLDT and $\Phi_{T_\sigma} \equiv \sigma$.*

Proof. Because σ is a safe existential sentence and each conjunct labels one node, each variable is locally covered.

We prove the equivalence by induction on ℓ.

$\ell = 1$**:**

$$\begin{aligned} \Phi_{T_\sigma} &\equiv (\exists \overline{x}_1)\gamma_1(\overline{x}_1) \\ &\equiv \sigma \end{aligned}$$

$\ell > 1$**:** Let σ' be σ without the last disjunct. Then by the induction hypothesis we know that:

$$\sigma' \equiv (\exists \overline{x})[\gamma_1(\overline{x}_1) \vee \ldots \vee \gamma_{\ell-1}(\overline{x}_{\ell-1})] \equiv \Phi_{T_{\sigma'}}$$

By construction, T_σ is obtained from $T_{\sigma'}$ by replacing the `no`-labeled leaf by an internal node m labeled with $\gamma_\ell(\overline{x}_\ell)$, and with the left child labeled `yes` and the right child labeled `no`. From Lemmas 2 and 3 it readily follows that the α-formula of node m is equivalent to $\neg\Phi_{T_{\sigma'}}$. Hence, the sentence describing the semantics of T_σ is:

$$\begin{aligned} \Phi_{T_\sigma} &\equiv \Phi_{T_{\sigma'}} \vee \left[\neg\Phi_{T_{\sigma'}} \wedge \gamma_\ell(\overline{x}_\ell)\right] \\ &\equiv \Phi_{T_{\sigma'}} \vee \gamma_\ell(\overline{x}_\ell) \\ &\equiv \sigma \end{aligned}$$

□

5 Discussion

Our result places the expressive power of FOLDTs at a rather low position in the quantifier alternation hierarchy for first-order logic [4]. The safe existential sentences are a subset of Σ_1, 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

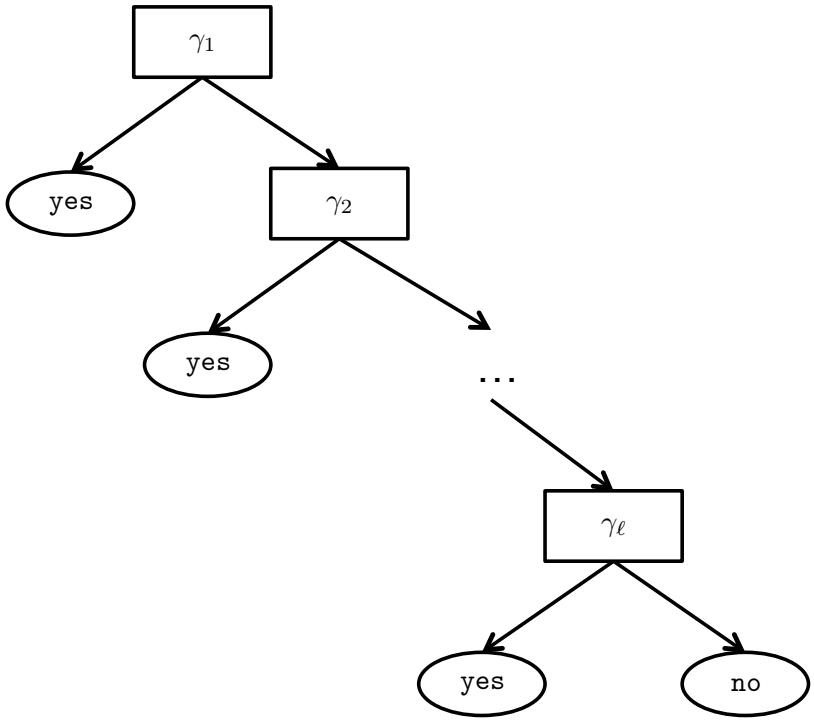

Fig. 5. Illustration of a FOLDT expressing a Σ_1-formula.

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 [4]. 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.

In the present work we have focused on *safe FOLDTs* and the safe fragment of existential formulas. This is due to the nature of Prolog evaluation. In future work, we would like to dig deeper into the "full" semantics of FOLDTs, i.e., we would like to incorporate unsafe FOLDTs in our definition of the semantics.

Most commonly, ILP learners induce *clausal theories*, i.e., conjunctions of universally quantified clauses (disjunctions of literals). Since a univerally quantified disjunction of literals amounts to a negated Σ_1-formula, clausal theories

thus correspond to the fragment of $BC(\Sigma_1)$ formed by all conjunctions of negated Σ_1-formulas. (A safety notion appropriate for clauses an be analogously defined.)

Acknowledgment

We thank Hendrik Blockeel and Jan Struyf for interesting discussions and help with the Tilde system. We also thank the anonymous referees for their helpful comments on an extended abstract of this paper.

References

1. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101, 285–297 (1998)
2. Blockeel, H., Dehaspe, L., Ramon, J., Struyf, J., Van Assche, A., Vens, C., Fierens, D.: The ace data mining system (2008)
3. 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), https://lirias.kuleuven.be/handle/123456789/123799
4. Chandra, A., Harel, D.: Structure and complexity of relational queries. Journal of Computer and System Sciences 25(1), 99–128 (1982)
5. De Raedt, L.: Logical and Relational Learning. Cognitive Technologies, Springer (2008)
6. Martin, E., Sharma, A., Stephan, F.: A general theory of deduction, induction, and learning. In: Proceedings of the 4th International Conference on Discovery Science. pp. 228–242. DS '01, Springer-Verlag, London, UK (2001)
7. Ohsherson, D., Weinstein, S.: Identification in the limit of first order structures. Journal of Philiosophical Logic 15, 55–81 (1986)
8. Osherson, D., Stob, M., Weinstein, S.: A universal inductive inference machine. The Journal of Symbolic Logic 56, 661–672 (1991)
9. Terwijn, S.: Learning and computing in the limit. In: Logic Colloquium '02, Lect. Notes Log., vol. 27, pp. 349–359. Assoc. Symbol. Logic, La Jolla, CA (2006)
10. Ullman, J.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press (1988)