

LEARNING FAMILY RELATIONSHIPS VIA
PROPOSITIONAL MEANS

BY

CHAD CUMBY

THESIS

for the degree of Bachelor of Science in Computer Science
in the Undergraduate College of the
University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois

1 Introduction

The problem of learning a set of relational concepts which may hold among a given set of objects has become increasingly important in the field of Machine Learning over the past several years. From Natural Language Processing, to Machine Vision, to Biological Computation, it has become apparent that some relational modeling of the problem area is necessary in order to perform well on most tasks. Thus, efficient algorithms for learning relations from examples are also necessary.

Traditionally, Inductive Logic Programming methods have been studied for learning relational rules. However, for large scale problems in areas such as Natural Language, ILP methods have been shown to be brittle and inefficient, while the subsumption problem of evaluating general first order logic rules over a data instance is intractable. Without greatly restricting the form of the rules learned, it becomes impossible to learn ILP programs in an efficient way. ILP systems utilizing many different heuristics have been successfully developed to get around the intractability problems associated with unrestricted inductive methods [?].

This work advances the approach introduced in [?, ?, ?], which aims to provide an alternative to traditional ILP methods for relational learning by utilizing an expressive knowledge representation language along with standard propositional learning algorithms. This knowledge representation language allows relational data to be converted to an intermediate form usable by any propositional algorithm, including probabilistic methods. The paradigm relies on a graphical structuring of data instances along with efficient algorithms to extract intermediate representations from this structuring by the means of Relation Generation Functions.

While significant results comparing this new propositional approach to learning relations have been shown for large scale problems in NLP and Information Extraction [?, ?], one of the major purported strengths of the ILP paradigm is its ability to learn simple rules in the presence of limited data sources for highly structured problem areas. Thus one of the most widely studied examples in ILP literature is the problem of learning family relations. This problem is especially attractive for supporters of the ILP paradigm as both the target concepts and the antecedents available for learning are relational, with little to no ground literals present in learned programs. Addi-

tionally, all of the concepts present should be learned in terms of a relatively small number of highly structured examples.

We show that in addition to performing well on large scale NLP and IE problems, our system learns these simple family relations as well the standard ILP systems, as exemplified by FOIL [?], while benefitting from a vast increase in efficiency.

2 Knowledge Representations

Here we introduce the knowledge representations used to extract the intermediate propositional form utilized in the learning. As in [?], we rely on: (1) the restricted first-order logic language \mathcal{R} with its connectives and operators, and (2) a set of structured instances in a domain which are represented as graphs, with well-defined graphical operations over these instances.

\mathcal{R} is a restricted FOL language with restrictions over the scope of the quantifiers allowed and the manner in which the formulae in \mathcal{R} may be generated. This allows the efficient generation of propositional representations of relational data. The *alphabet* Σ consists of (i) variables, (ii) constants, (iii) predicate symbols, (iv) quantifiers and (v) connectives. (ii) and (iii) vary from alphabet to alphabet while (i), (iv) and (v) are the same for every alphabet. Formulae in \mathcal{R} are defined to be restricted first order language formulae in which there is only a single predicate in the scope of each variable.

Definition 2.1 *An atomic formula is defined inductively as follows: (1) A term is either a variable or a constant. (2) Let p be a k -ary predicate, t_1, \dots, t_k terms. Then $p(t_1, \dots, t_k)$ is an atomic formula. (3) Let F be an atomic formula, x a variable. Then $(\forall x F)$ and $(\exists x F)$ are atomic formulae.*

Definition 2.2 *A formula is defined inductively as follows: (1) An atomic formula is a formula. (2) If F and G are formulae, then so are $(\sim F)$, $(F \wedge G)$, $(F \vee G)$.*

The relational language given by the alphabet consists of the set of all formulas constructed from the symbols of the alphabet. We call a variable-less atomic formula a *proposition* and a quantified atomic formula, a *quantified proposition* [?]. The informal semantics of the quantifiers and connectives is as usual.

For formulae in \mathcal{R} , the *scope* of a quantifier is always the unique predicate that occurs with it in the atomic formula. All the formulae in \mathcal{R} are *closed* since all the formulae are composed from propositions or quantified propositions which are connected via \sim , \wedge or \vee , and thus all occurrences of variables are *bound*.

The semantics of formulae in \mathcal{R} are derived from the *domain* over which they are defined.

Definition 2.3 *A domain \mathcal{D} for the language \mathcal{R} consists of a structured element $D = \langle \mathcal{V}, \mathcal{G} \rangle$ where \mathcal{V} is a collection of typed elements and \mathcal{G} is a set of partial orders over \mathcal{V} (specifically, each partial order $g_i \in \mathcal{G}$ is a graph (V_i, E_i) , where $V_i \subseteq \mathcal{V}$ and E_i is a set of edges on V_i). Along with it, for each constant there is an assignment of an element in V and for each k -ary predicate, an assignment of a mapping from V^k to $0, 1$ (true, false).*

We distinguish the set V as being typed in terms of *objects* and *attributes*. Then primitive formulae in \mathcal{R} are typed in terms of *attribute* predicates if they take an both an object and attribute argument (describing properties of objects in the domain). Otherwise they will only take object elements as arguments. For a graph $G \in \mathcal{G}$, the predicate $p_G(o_1, o_2)$ is typed G and is valued *true* if o_1, o_2 are nodes in G with an edge connecting them.

Definition 2.4 *An instance is an interpretation [?] which lists a set of domain elements and the truth values of all instantiations of the predicates on them.*

Example 2.1 *In family relations, the domain $\mathcal{D} = (\mathcal{V}, \mathcal{G})$, where \mathcal{V} corresponds to the set of people making up a family in an instance, and \mathcal{G} corresponds to the set of graphs defined over each family instance, where each graph is a different relation. All predicates with value true in an instance are of the form $p_G(o_1, o_2)$.*

Given an instance x with all predicates valued true over it, a formula F in \mathcal{R} is given a unique truth value using the truth values of the primitive formulae in F and the semantics of the connectives. It has been shown (see [?, ?]) that the evaluation and subsumption of formulae in \mathcal{R} is efficient.

2.1 Relation Generation Functions

With a definition for a domain \mathcal{D} and a set of instances over the domain, a mechanism is necessary to generate expressive formulae in \mathcal{R} in a restricted way respecting the structures of the domain. When a formula has truth value *true* on an instance, we say it is *active*. We would like an efficient way to generate active formulae over a particular instance. The notion of the *relation generation function* (RGF) has been introduced [?, ?] in order to address this problem.