Feature Extraction Languages for Propositionalized Relational Learning

Chad Cumby **Dan Roth**
Department of Computer Science
University of Illinois, Urbana, IL 61801
{*cumby,danr*}*@uiuc.edu*

Abstract

We study representations and relational learning over structured domains within a propositionalization framework that decouples feature construction and model construction.

We describe two complementary approaches that address three aspects of the problem: First, we develop and study a flexible knowledge representation for structured data, with an associated language that provides the syntax and a well defined equivalent semantics for expressing complex structured data succinctly. Second, we use this language to automate the process of feature construction by expressing 'types' of objects in the language, which are instantiated in the ground data, allowing us to determine the level at which learning is done. Finally, this process of re-representation of the domain allows general purpose learning schemes, such as feature efficient linear algorithms and probabilistic representations, to be defined over the resulting space, yielding efficient and expressive learning of relational functions over a structured domain using propositional means.

1 Introduction

In a variety of AI problems, such as natural language understanding related tasks and visual inference, there is a need to learn, represent and reason with respect to definitions over structured and relational data. Examples include learning to identify properties of text fragments such as functional phrases and named entities, identifying relations such as "A is the assassin of B" in text, learning to classify molecules for mutagenicity from atom-bond data in drug design, learning to identify 3D objects in their natural surrounding and learning a policy to map goals to actions in planning domains.

In all these cases it is necessary (1) to represent and reason with *structured* domain elements in the sense that their internal (hierarchical) structure can be encoded, and learning functions in these terms can be supported, and (2) it is essential to represent concepts and functions *relationally*, in the sense that different data instantiations may be abstracted to yield the same representation – so that evaluation of functions over different instantiations will produce the same output.

The challenge is to provide the expressivity necessary to deal with large scale and highly structured domains such as natural language and visual inference and at the same time meet the strong tractability requirements for these tasks. Propositional representations might be too large, could lose much of the inherent domain structure and consequently might not generalize well. This realization has renewed the interest in studying relational representations both in the knowledge representation and reasoning (KRR) community and in the learning community. As it turns out, both are relevant to our approach. The main effort in the knowledge representation and reasoning community has been to identify classes of representations that are expressive enough to allow reasoning in complex situations yet are limited enough as to support reasoning efficiently [Levesque and Brachman, 1985; Selman, 1990]. It has become clear that propositional representations are not sufficient, and effort has been devoted to studying languages that are subsets of first order logic, such as description logics and frame representation systems [Borgida and Patel-Schneider, 1994], as well as probabilistic augmentations of those [Koller *et al.*, 1997].

The expressivity vs. tractability issue has been addressed also from the learning perspective, and a similar tradeoff has been observed and studied. While, in principle, Inductive Logic Programming (ILP) methods provide the natural approach to these tasks in that they allow induction over relational structures and unbounded data structures, theoretical and practical considerations render the use of unrestricted ILP methods impossible. These methods have also been augmented with the ability to handle uncertainty [Kersting and Raedt, 2000] although, as expected, this makes some of the computational issues more severe - studies in ILP suggest that unless the rule representation is severely restricted the learning problem is intractable [Muggleton and De Raedt, 1994; Dzeroski *et al.*, 1992; Cohen, 1995a; 1995b].

The main way out of these computational difficulties has been via the use of propositionalization methods that attempt to learn classifiers for relational predicates via propositional algorithms, mapping complex structures to simple features [Lavrac *et al.*, 1991; Kramer *et al.*, 2001; Khardon *et al.*, 1999]. These approaches attempt to decouple feature construction and model construction (learning) by devising methods to produce propositional features from structured data.

This paper is best viewed in this context, as it describes

our work on feature extraction languages for propositionalized relational learning. The study of feature extraction languages in the context of learning relational representations over structured domains needs to address at least three aspects of the problem. First, we develop and study a flexible knowledge representation for structured data, with an associated language that provides the syntax and a well defined equivalent semantics for expressing complex structured data succinctly. Second, we use this language to automate the process of feature construction by expressing 'types' of objects in the language, which are instantiated in the ground data. In particular, this process can determine the level at which learning is done (between ground literals and full relational expressions) by choosing these types appropriately. Finally, this process of re-representation of the domain should allow general purpose learning schemes, such as feature efficient linear algorithms and probabilistic representation and algorithms, to be defined over the resulting space.

The paper describes two different but complementary extraction frameworks and discusses their equivalence. The first, "functional" framework, following [Cumby and Roth, 2000], defines a set of relational formulae $\mathcal{R}$, a subset of FOL, with a functional calculus composed of so-called "Relational Generation Functions" (RGFs). These functions serve to generate elements of $\mathcal{R}$ representing (properties of) ground input data elements. The formulae could then be treated as features for a propositional learner. In this framework ground data is codified in a graphical structure on which the RGF calculus operates, and elements in the language are defined operationally via this calculus. The second, "syntactic" framework, expanding on [Cumby and Roth, 2002], provides a unified language used both in expressing structured features and in generating them. It builds on the idea of Description Logics to give a concrete syntactic form to the graphical representation of ground data introduced in the first framework. We provide a formal syntax and semantics for a specific feature description language (FDL) - but this is only one member in a family of languages, deterministic or probabilistic, that could be used within our framework. Domain elements and properties of them are "concepts" which are described, as in other description logics, in terms of *individuals* possessing attributes and roles in relation to other individuals. The equivalence of descriptions in FDL to a class of concept graphs is used to show efficient subsumption between descriptions The importance of inference with relational representations becomes clear in this paradigm. The description logic is an intermediate step and the basic inference step, subsumption, is used as a means to transform a domain element, e.g., a natural language sentence, and represent it in terms of a richer vocabulary – descriptions in our Feature Description Logic (FDL). This representation, in turn, may serve as an input to any propositional learning algorithm, including probabilistic algorithms, to yield structures in which sought after predicates are represented as functions (or conditional probabilities) over the relational descriptions.

We then discuss the extent to which the flexible operational language and the better defined syntactic language are equivalent and provide a mapping between the two. The FDL language is shown to possess a semantics similar to the subset $\mathcal{R}$ of FOL introduced earlier; and, the parameterized Feature Generating Function is shown to duplicate the operation of the RGF calculus.

Both frameworks differ from standard ILP approaches and most propositionalization techniques. Features are generated up front before any learning stage, in a data-driven way, based on background knowledge (or pre-learned knowledge) in the "type" of feature defined. This allows us to dictate the level of complexity of our intermediate representation before learning occurs, and to bypass a potentially expensive search for good features. Thus particularly expressive features that would not necessarily be generated during a search are allowed to influence our final learned function in a significant way.

Our techniques are aimed at complicated large-scale relational learning problems in which ground features, in addition to quantified predicates, play an important role in any learned classifier. This is the case in many natural language applications [Roth and Yih, 2001; Khardon *et al.*, 1999] where lexical features are an important part of the learned concept. In this cases, the potential number of features is very large and choosing a suitable learning approach in conjunction with the feature extraction approach is essential.

While the learning approach is presented here as an approach to learn a definition for single predicates, we view this in a wider context. Learning definitions may be used to enrich vocabulary describing the input data; the feature extraction technique can then be used incrementally to produce useful features again and subsequently to build up new representations in terms of those in a manner similar to the one envisioned in [Valiant, 1999]. Such a system might integrate easily into a programming platform, allowing researchers to construct large scale learning-based architectures to solve complex AI problems in areas such as natural language processing. It then becomes even more crucial that the basic components of this system are articulated in a language whose structure and meaning are well understood.

The remainder of the paper is structured as follows: Sec. 2 surveys related work. Sec. 3 explains the machine learning setting in which our frameworks can be used. Sec. 4 presents our Functional Feature Extraction Framework and Sec. 5 the Syntactic Feature Extraction Framework. The relations between the two is discussed in Sec. 6, and Sec. 7 concludes.

2 Related Work

Our work is mostly related to the work in the ILP community on the topic of learning relational concepts by propositional means. Commonly known as "propositionalization" methods, these approaches reformulate data for relational problems in terms of attribute-value feature vectors. A hypothesis is then induced over the set of these new features.

However, our language allows us to generate expressive, relational formulae – "quantified propositions" – and place this within any model construction approach. Specifically, it is possible to learn probabilistic classifiers and models over quantified propositions extracted with the our approach [Punyakanok and Roth, 2001]. Thus, it can also be viewed and compared with probabilistic approaches. Moreover, defining the "type" of features so as to dictate the abstraction level

of our intermediate representation is conceptually similar to modeling approaches such as relational probabilistic models [Friedman *et al.*, 1999], where the modeler may determine the level of abstraction and dependencies between entities, attributes and relations.

The following brief survey, however, focuses on related propositionalization approaches, specifically those that utilize a graph-based knowledge representation. See [Kramer *et al.*, 2001] for a good survey of propositionalization methods.

The most similar formulation to the approach we present is Kramer's graph-based approach for feature construction in biochemical domains. This approach [Kramer and Frank, 2000; Kramer and Raedt, 2001], uses structural features produced by a molecular feature mining program called MolFea in conjunction with SVM to learn a classifier for predicting carcinogenicity in molecules. [Kramer and Raedt, 2001], uses a version-space approach to represent a set of fragments in the input data that is more general than, and more specific than a given fragment. This is somewhat similar to our notion of defining particular "types" of features which are instantiated in the input data, however we allow the programmer to constrain the specificity of the instantiated features in a way designed to reduce overfitting.

Other graphical techniques include the method of [Geibel and Wysotzki, 1996] which, like ours, utilizes properties of proximity in a graph-based representation of the input data to restrict the range of features produced during feature construction, and [Cook and Holder, 1994; Gonzalez *et al.*, 2002], which construct features from graphical instances but restrict the number of features produced (to "good" features), blurring the line between feature and model construction.

3 The Learning Framework

The propositionalization approach presented in this work consists of a feature extraction stage – structured data elements represented as labeled graphs are converted to features representing relational and grounded properties of it – along with a general purpose propositional model generation (learning) stage that makes use of the extracted vocabulary.

In this framework, as in ILP, each observation in the domain is mapped into a collection of predicates that hold over elements in the domain. The key difference from standard ILP is that our representation of an observation may contain quantified formulae. "Examples" of this form are then given as input to a learning algorithm, that is supposed to produce a classifier to predict whether a particular target predicate holds for some particular elements. For example, we may wish to predict that for some domain elements X and Y, the predicate $father(X, Y)$ holds. To accomplish this task using standard propositional learning algorithms, we must generate *examples* in the form of lists of active propositions (features) for each predicate to be learned. Propositions of this form may either be fully ground as in the predicate $father(john, jack)$, or existentially quantified as in the predicate $\exists X\ father(john, X) \wedge father(X, harry)$. In the supervised learning setting each example will contain a label feature, which corresponds to the true relation between X and Y. An example of this sort can also serve as a negative example for other possible relations between elements that do not hold in it. Our major task then becomes to re-represent the data in a manner conducive to producing features over which we can learn a good model or a good discriminant function. This re-representation is the subject of the work described in the rest of this paper.

The feature extraction methods presented operate under the closed-world assumption, generating only the features judged to be active in the observation. All other features are judged to be inactive, or false. As it may be inefficient or impossible to list all features for a particular interpretation, this is a performance boon. Thus our learning algorithm should be able to accept examples represented as variable length vectors of only positive features. In addition, our methods provide the flexibility to generate a large number of features by designating a smaller set of "types" of features, so our learning algorithm should be able to learn well in the presence of a large number of irrelevant features.

In most of the applications of our approach we have used as the learning component, the SNoW[1] learning system. This is a multi-class propositional classifier suited to a high dimensional but sparse representation of feature data of variable length that uses a network of linear functions to learn the target concept. It has been shown to be especially useful for large scale NLP and IE problems [Khardon *et al.*, 1999; Roth and Yih, 2001; Golding and Roth, 1999]. Unlike "traditional" ILP methods that typically learn concepts represented as conjunctive rules, SNoW employs a variation of a feature-efficient learning algorithm, Winnow [Littlestone, 1988] (or other linear learning algorithms), to learn a linear function over the feature space; consequently, these "generalized rules" are more expressive than simple rules, and are easier to learn.

4 Functional Feature Extraction Framework

This section introduces a feature extraction framework in which elements in a restricted subset of first-order logic, generated using a set of composable functions with an associated calculus, serve as features for learning.

4.1 The Relational Language $\mathcal{R}$

The relational language $\mathcal{R}$ is a restricted first order language. The *alphabet* 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 function-free first order language formulae in which there is only a single predicate in the scope of each variable.

Definition 4.1 *An* atomic formula *is defined inductively:*

1. *A* term *is either a variable or a constant.*
2. *Let p be a k-ary predicate, $t_1, \ldots, t_k$ terms. Then $p(t_1, \ldots, t_k)$ 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 4.2 *A* formula *is defined inductively as follows:*

[1] Available at `http://L2R.cs.uiuc.edu/~cogcomp/`

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 formulae constructed from the symbols of the alphabet. We call a variable-less atomic formula a *proposition* and a quantified atomic formula, a *quantified proposition* [Khardon *et al.*, 1999]. 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 formulae in $\mathcal{R}$ are *closed* since all formulae are composed from propositions or quantified propositions which are connected via $\sim$, $\wedge$ or $\vee$, thus variable occurrences are *bound*.

4.2 Interpretation

$\mathcal{R}$ is used as a language for representing knowledge with respect to a domain; we now define how formulae in $\mathcal{R}$ receive their truth values.

Definition 4.3 *A domain $\mathcal{D}$ for the language $\mathcal{R}$ is a collection D of elements along with*

(i) *An assignment for each constant in $\mathcal{R}$ to an element in D.*

(ii) *For each k-ary predicate in $\mathcal{R}$, the assignment of a mapping from D^k to $\{0,1\}$ ($\{$true,false$\}$).*

When there is no confusion, we will call D, the set of elements in the domain, the domain. We can always think of D as the Herbrand base, the collection of all ground atoms in $\mathcal{R}$ [Lloyd, 1987]. In this case any interpretation is a subset of the Herbrand base, so we can talk in terms of subsets of D.

Given $D' \subseteq D$, a formula F in $\mathcal{R}$ is given a unique truth value, which we call *the value of F on D'*. This value is defined inductively using the truth values of the predicates in F, and the semantics of the connectives. Notice that if F has the form $\exists p$ ($\forall p$, resp.), for some k-ary predicate, then its truth value is *true* (1) iff there exists (for all, resp.) $d_1, \ldots d_k \in D'$ such that $p(d_1, \ldots d_k)$ has truth value *true*. Since for formulae in $\mathcal{R}$ the scope of a quantifier is always the unique predicate that occurs with it in the atomic formula, we have:

Proposition 4.4 *Let F be a formula in $\mathcal{R}$, $D' \subseteq D$, and let t_p be the time to evaluate the truth value of an atom p in F. Then, the value of F on D' can be evaluated in time $\sum_{p \in F} t_p$.*

That is, F is evaluated simply by evaluating each of its atoms (ground or quantified) separately.

4.3 Relational Generation Functions

Definition 4.5 (features) *Let D be a domain, $D' \subseteq D$. We call D' an* instance, *and $X = 2^D$ an* instance space. *A* feature[2] *is a function $\chi : X \rightarrow \{0,1\}$. χ can be viewed as an indicator function over X, defining the subset of those elements in X that are mapped to 1 by χ.*

Formulae in $\mathcal{R}$ are viewed as features over the instance space 2^D. A formula F maps $D' \subseteq D$ to its truth value on D'. A formula is *active* in D' if it has truth value *true* on D'.

[2] In earlier versions of this work features were called "relations".

Given an instance, we would like to know what are the features (with corresponding formulae) that are *active* in it. We would like to do that, though, without the need to write down explicitly all possible formulae in the domain. This is important, in particular, over infinite domains or in on-line situations where the domain elements are not known in advance, and therefore it is simply impossible to write down all possible formulae. An efficient way to do that is given by the construct of *relational generation functions*. As will be clear later, this notion will also allow us to significantly extend the language of formulae by exploiting properties of the domain.

Definition 4.6 *Let $\mathcal{X}$ be an enumerable collection of features on X. A relational generation function (RGF) is a mapping $G : X \rightarrow 2^{\mathcal{X}}$ that maps $x \in X$ to a set of all elements in $\mathcal{X}$ that satisfy $\chi(x) = 1$. If there is no $\chi \in \mathcal{X}$ for which $\chi(x) = 1$, $G(x) = \phi$.*

RGFs can be thought of as a way to define "types" of formulae, or to parameterize over formulae. Only when an instance $D' \subseteq D$ is presented, concrete formulae are generated.

4.4 Relational Calculus

The family of relational generation functions for $\mathcal{R}$ are RGFs whose outputs are formulae in $\mathcal{R}$. Those are defined inductively, just like the definition of the language $\mathcal{R}$.

The relational calculus is a calculus of symbols that allows one to inductively compose relational generation functions. The alphabet for this calculus consists of (i) basic RGFs, called *sensors* and (ii) a set of connectives. While the connectives are the same for every alphabet the *sensors* vary from domain to domain. A sensor is a way to encode basic information one can extract from an instance. It can also be used as a uniform way to incorporate external knowledge sources that aid in extracting information from an instance.

Definition 4.7 *A* sensor *is a relational generation function that maps an instance D' into a set of atomic formulae in $\mathcal{R}$. When evaluated on an instance D' a sensor s outputs all atomic formulae in its range which are* active.

Definition 4.8 *Let C be a set of formulae. A* conditioning *operation $|C$ on an RGF r restricts r to output features for only formulae in C.*

Definition 4.9 *The operation of a* relational generation function *(RGF) for $\mathcal{R}$ is defined inductively as follows:*

1. *When evaluated on an instance D' the sensor s outputs features for all* active *atomic formulae in its range.*
2. *If s and r are RGFs for $\mathcal{R}$, then so are $(\neg s_{|F}), (s \& r), (s_{|F_1} | r_{|F_2})$.*
   - **i** *The feature output by $(\neg s_{|F})$ corresponds to the formula $\neg F$ given that F is in the range of s and is not active on D'.*
   - **ii** *The features in the output of $(s \& r)$ correspond to active formulae of the form $F_1 \wedge F_2$, where F_1 is in the range of s and F_2 is in the range of r (evaluated on D').*
   - **iii** *The features in the output of $(s_{|F_1} | r_{|F_2})$ correspond to formulae of the form $F_1 \vee F_2$, where either F_1 is active in D' or F_2 is active in D'.*

Notice that for negation and disjunction it is necessary to condition the argument RGFs with input formulae, as for many sensors the range of formulae which are not active in the current instance may be infinite. For conjunction and disjunction, it is possible to focus the range of formulae to those active for a particular subset of the current instance, based on structural information as described in Sec. 4.5.

4.5 Structural Instance Space

So far we have presented $\mathcal{R}$ and RGFs with respect to an abstract domain D. In most domains more information than just a list of objects and assignments is available. We abstract this using the notion of a *structural domain* that is defined below. Instances in a structural domain are augmented with some structural information and, as a result, it is possible to define more expressive RGFs in terms of the sensors provided along with the domain.

Structured Instances

Definition 4.10 *Let D be the set of elements in the domain. A* structured instance *O is a tuple $(V, E_1, E_2, ...E_k)$ where V is a set of nodes each associated with some subset $D' \subseteq D$ of elements in the domain, and E_i is a set of edges on V. The graph $G_i = (V, E_i)$, is called the ith* structure *of instance O.*

Structural Operations

We now augment the relational calculus of Sec 4.4 by adding structural operations. These operations exploit the structural properties of the domain as expressed in the graphs G_is in order to define RGFs, and thereby generate non-atomic formulae that may have special meaning in the domain.

Definition 4.11 *Let $V' \subseteq V$ be a set of nodes in the structured instance O. An RGF r is* focused *on V' if, given an instance D' it generates features only for formulae in its range that are active on those domain elements associated with V'. The focused RGF is denoted $r[V']$.*

Definition 4.12 *Let $s_1, s_2, \ldots s_k$ be RGFs for $\mathcal{R}$. $colloc_g(s_1, s_2, \ldots s_k)$ is a restricted conjunctive operator that is evaluated on a chain of length k in the gth structure of the given structured instance. Specifically, let $O = \{G_i\}_1^m$ be a structured instance and let $v_1, v_2, \ldots v_k$ be a chain in G_i. The features generated by $colloc_i(s_1, s_2, \ldots s_k)$ are those generated by $s_1[v_1]\&s_2[v_2]\&\ldots\&s_k[v_k]$, where by $s_j[v_j]$ we mean here the RGF s_j focused to $\{v_j\} \subseteq V$, and the $\&$ operator is defined as in Definition 4.9. Notice that each subRGF conjunctions may produce more than one feature.*

Focus-Word Centered Representation

The structural information also provides an easy way to *focus* the RGFs (Def 4.11). For example, defining a set of elements for the focus set V' in $s[V']$ can be done using some graph property. Specifically, we use the notion of a focus node, and define a focus set with respect to it using a radius length. In particular, in the $colloc$ operation, we can restrict the chains to *start* at a node $v' \in V$ at a certain length in edges from a focus node v or to *contain* it. Notice that if, for the given instance $O = (V, G)$, we have that $v' \notin V$, then the output is an empty set of features.

The next section describes an alternative to the functional extraction framework based on a syntactic construction.

5 Syntactic Feature Extraction Framework

This section presents a feature extraction framework based on a description logic-like language. Statements in this language are constructed to be equivalent to a restricted graphical representation for our relational data, and they serve as input for a parameterizable *Feature Generating Function*.

5.1 Feature Description Logic

The basic Feature Description Logic (FDL) is described below by providing its formal syntax and semantics.

As in most formal description logics, FDL descriptions are defined with respect to a set X of *individuals*. However, unlike most KL-ONE-like description logics, the basic alphabet for FDL descriptions includes *attribute*, *value*, and *role* symbols. We differentiate attribute from role descriptions and our basic primitive description is an attribute-value pair. We also allow a non-functional definition of attribute descriptions and role descriptions; thus an attribute describing an individual may take many values and a role describing an individual could take several different fillers.

This type of language is useful since, at a basic level, statements in the language abstract over sets of objects in different domain instances that have that have the same attributes and relationships to other objects present. These statements therefore serve as a useful basis for features for learning algorithms. At another level, by quantifying over the set of values that attributes can take, we can describe an even more general set of individuals. The procedure which we later introduce takes statements of this type and information about the current set of objects being considered, and rewrites the statements to contain the values seen in the current instance.

Definition 5.1 *A FDL* description *over the* attribute *alphabet $Attr = \{a_1, ..., a_n\}$, the* value *alphabet $Val = v_1, ..., v_n$, and the* role *alphabet $Role = \{r_1, ..., r_n\}$ is defined inductively as follows:*

1. *For an attribute symbol a_i, a_i is a description called a* sensor. *For some value symbol v_j, $a_i(v_j)$[3] is also a description, called a* ground sensor. *We also define a special* identity sensor *denoted $*$, which represents all individuals x.*
2. *If D is a description and r_i is a role symbol, then (r_i D)[4] is a* role *description.*
3. *If $D_1, ..., D_n$ are descriptions, then (***AND** *$D_1, ..., D_n$) is a description. (The conjunction of several descriptions.)*

We also define the *size* of a description $|D|$ as the number of conjunctive and role sub-descriptions present in D. Def. 5.1 allows the recursive construction of FDL descriptions.

We now turn to the semantics of FDL descriptions. This discussion follows a model-theoretic framework similar to that laid out in [Borgida and Patel-Schneider, 1994]. This definition uses the notion of an interpretation [Lloyd, 1987], and that of an *interpretation function* which can be viewed as the function that encodes the information about domain. For a domain element z we denote by z^I its image under the interpretation function.

[3] read: a_i takes value v_j

[4] read: relation r_i holds for current object and those in $ext(D)$

Definition 5.2 (FDL extension) *An interpretation I consists of a domain Δ, for which there exists an interpretation function I. The domain is divided into disjoint sets of individuals, X, and values, V. The interpretation function assigns an element $v^I \in V$ to each value v. It assigns a set of binary relations a^I over $X \times V$ to each symbol a in Attr, and a set of binary relations r^I over $X \times X$ to each symbol r in Role. The* extension *of a FDL description $ext(D)$ is defined as follows:*

1. *The extension of a sensor is defined as $ext(a(v)) = \{x \in X | (x, v^I) \in a^I\}$. The extension of an existential sensor $ext(a)$ is $\{x \in X | \exists v^I \in V$ s.t. $(x, v^I) \in a^I\}$.*
2. *The extension of a role is defined as $ext((r\ D))$ = $\{x \in X | (x, y) \in r^I \rightarrow y \in D^I\}$.*
3. *The extension of a conjunctive expression $ext($(**AND** D_1 D_2)) is defined as $ext(D_1) \bigcap ext(D_2)$.*

We can now define the *subsumption* of a FDL description D_1 by another description D_2. We say that D_1 *subsumes* D_2 iff the extension of D_2 is a subset of the extension of D_1. i.e. $D_1^I \supseteq D_2^I$ for all interpretations I. In our framework subsumption is used to transform a domain element, represented as a concept graph, into a feature set that can serve as an input to a propositional learning algorithm.

To show that FDL allows efficient subsumption, we use the notion of a concept graph that we define next.

5.2 Concept Graphs

The notion of concept graphs stems from work in the semantic network and frame-based representations. In many ways description logics were invented to provide a concrete semantics for the construction of such graph-based knowledge representations. Here they provide a tool for computing subsumption between descriptions and as a convenient representation for examples presented to algorithms in our learning framework.

FDL concept graphs are a variation on the type invented for [Borgida and Patel-Schneider, 1994] to explain "basic CLASSIC". A FDL concept graph is a rooted labeled directed graph $G = G(N, E, v_0, l_N)$, where N is a set of nodes, $n_0 \in N$ is the root of the graph, $E \subseteq (N \times N \times Role)$ a set of labeled edges (with role symbols as labels) and l_N is a function that maps each node in N to a set of sensor descriptions.

The semantics of FDL concept graphs is defined similarly to that of basic CLASSIC, minus those associated with equality constraints. The extension of a node in the graph is intended to be the set of individuals described by its corresponding description.

Definition 5.3 (Concept Graph extension) *Given a FDL concept graph $G = (N, E, n_0, l_N)$, a node $n \in N$, and an interpretation I in some domain Δ composed of elements X and values V, we say that an individual $x \in X$ is in the extension of n iff:*

1. *For each sensor $a_i(v) \in l_N(n)$, $a_i^I(x, v^I)$ is true. For each sensor $a_i \in l_N(n)$, $\exists v^I \in V$ s.t. $a_i^I(x, v^I)$ is true.*
2. *For each edge $(n, m, r_i) \in E$, $\forall y \in X$ if $r_i^I(x, y)$ then y is in the extension of m.*

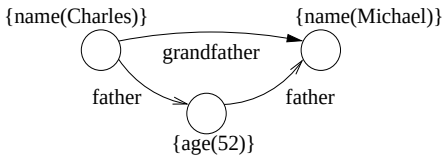

Figure 1: An example concept graph for the kinship domain.

As in earlier works on DL, an individual x is in the extension of G, iff it is in the extension of n_0. It will be clear later that in our paradigm we care about concept graph extension only as a clean way to define subsumption; the more basic notion here is the description itself. Two constructs over domain Δ are semantically equivalent if they have the same extensions given an interpretation I. The significance of concept graphs for our purposes stems from the following theorem.

Theorem 5.4 *Any FDL description D is semantically equivalent to an acyclic FDL concept-graph of size polynomial in $|D|$ that can be constructed in polynomial time.*

Thm 5.4 allows now to show that FDL supports efficient subsumption queries between descriptions and, moreover, that it supports checking subsumption of an arbitrary concept graph by a description.

Theorem 5.5 *For FDL descriptions D_1, D_2 the subsumption of D_2 by D_1 ($D_1 \supseteq D_2$) can be decided in polynomial time. Additionally for a description D_1 and an arbitrary FDL concept graph G_2, the subsumption of G_2 by D_1 can be decided in polynomial time.*

Given these definitions for FDL descriptions and their corresponding concept graph representations, it now becomes possible to describe a feature extraction framework where such representations play a major role. Efficient subsumption testing allows generation of expressive propositional features from arbitrarily complex data represented by concept graphs.

5.3 Feature Generating Functions

Up until this point, our treatment of FDL has closely mirrored that of similar CLASSIC-like DL's [Borgida and Patel-Schneider, 1994]. However, our usage of FDL descriptions is vastly different from the usage of descriptions in these other DL's. The most closely related usage may be that of P-CLASSIC [Koller *et al.*, 1997] descriptions, in which a probabilistic distribution over descriptions is used to perform probabilistic subsumption queries. Instead, in our paradigm, descriptions are used to generate propositional formulae, in a data-driven way via subsumption queries. We first describe the process of generating propositional formulae using FDL descriptions.

The essential construction of our method is a *Feature Generating Function*, closely related to the RGF of Sec. 4.3.

The constructions, however, differ in an important respect. Here we discuss a general Feature Generating Function, whose operation is constrained by the formal syntax of the generating descriptions themselves, having well defined structure and meaning. This therefore extends and unifies the "relational calculus" of Sec. 4.3 that procedurally composes different types of RGFs to produce complex features.

In fact, we claim that any operation of such a calculus may be pushed onto the syntax of an FDL, and therefore it is possible to define descriptions in our language that produce exactly the same features as produced there (as we explain later).

Definition 5.6 (features) *Let I be some interpretation with domain $\Delta = (X, V)$, and let $\mathcal{I}$ be the space of all interpretations. For a description D we define a* feature F_D *to be a function $F_D : \mathcal{I} \rightarrow \{0, 1\}$. F_D acts an indicator function over $\mathcal{I}$, denoting the interpretations for which the extension of the description D is not empty.*

Given an interpretation I we say that a feature F is *active* in I if it evaluates to true. Generating such features efficiently however is the topic of much debate, as such feature spaces could be prohibitively large or in some cases infinite, making manual generation impossible.

Our next step is to automate the construction of features of this sort. Luckily, the semantics of FDL descriptions and their equivalence to rooted concept graphs give rise to an efficient method of constructing *active* features, via the notion of the *feature generating function* (FGF). Let some interpretation I be represented as a concept graph G, in which all elements of I are in the extension of some node of G. The construction of this graph is efficient, following Thm 5.4.

Our FGF method takes G along with a set of input FDL descriptions $\mathcal{D}$, and outputs a set of active features over G. The basic method computes a *feature description* D_θ[5] for each attribute as described in Def. 5.8 for G with respect to each description $D \in \mathcal{D}$, and constructs a feature for each D_θ. The intuition is that each input description defines a "type" of feature, subsuming many possible (partially) ground descriptions over an interpretation. We say a description is *ground* if it is a description containing only sensors of the form $a_i(v_i)$.

Definition 5.7 (Feature Generating Function) *Let $\mathcal{F}$ denote an enumerable set of features over the space $\mathcal{I}$ of interpretations and let D be a description. A* feature generating function $\mathcal{X}$ *is a mapping $\mathcal{X} : \mathcal{I} \times D \rightarrow 2^{\mathcal{F}}$ that maps and interpretation I to a set of all features in $\mathcal{F}$ such that $F_D(I) = 1$.*

Thus, the image of I under $\mathcal{X}$ is a re-representation of I in terms of the (set of D_θ's subsumed by the) description D.

Definition 5.8 *The* feature description *of a rooted concept graph G with respect to an input description D is the unique ground description D_θ subsuming G and subsumed by D, containing only ground forms of the sensors in D.*

In the case that D is itself already ground, computing the feature description D_θ amounts to checking the subsumption of G by D.

As usual, the importance of features stems from the fact that they might provide some abstraction. That is, they describe some significant property of the input which may occur also in other, different, inputs.

Theorem 5.9 *Given any interpretation I represented as a concept graph and a description D, all active features over I with respect to D can be generated by $\mathcal{X}$ in polynomial time.*

[5]The θ here indicates the binding that occurs for each attribute.

6 Mapping the Two Formalisms

As previously stated, the two methods presented are both means to generate propositional features from structured or semi-structured input data. In the first case presented, a more functional approach is taken. Sensor RGFs may produce formulae inferred through some process and not explicitly provided as predicates in the domain. For example, in a visual processing problem, we might want a sensor such as the $I > 50$ sensor discussed earlier. The domain may provide predicates only of the form $intensity(60)$,

By contrast, the second method attempts to push all of the functionality of the RGF functions, and the logical structure implied by using graph operations, onto the syntax of our description language. The process of constructing features is then reduced to a single mechanical operation. In order to produce a feature from a particular domain instance, the information contained in that feature must be explicitly represented in the graphical structure, or else implied in the syntax of the language. For complicated learning problems involving several stages of classification, we thus update the domain instance with information gained from previous stages. In this manner we can, for example, simulate the learning of recursive concepts such as a $path$ in a graph. If a domain instance is given with $edge$ arcs represented between each node, we can first learn a classifier to predict that two nodes linked by an $edge$ define a $path$. After filling in all predicted $path$ arcs based on this classifier, a second classifier based on the existing $path$ arcs and the new $edge$ arcs can be learned, and iteratively applied to predict the remaining $path$ edges between any pair of nodes.

While the above discussion would give the impression that the formalism of RGFs could yield more expressivity than the FDL formalism, we claim that we can simulate most of the features output in the first framework with the second. The details of this mapping appear in [Cumby and Roth, 2003a].

Here we stress that, although the formalisms presented can be mapped to one another in terms of creating features to express the same concepts in the same situations, each has its own unique advantages. The RGF formalism serves as a more abstract foundation for feature extraction. It allows us to devise extraction functions that are independent of the underlying knowledge representation used for the data. The conjunctive $\&$ operator and the disjunctive $|$ operator exhibit this independence. The FDL based approach, by encoding particular graph properties explicitly in the syntax of the language and by directing the feature extraction process through syntactic operators, gives a more "implementation-level" understanding of that process. Additionally, with a cleanly established syntax for the description language we can substitute other functionality in place of explicit feature generation. For example, the language can be used as the basis of a parameterizable family of kernels for use with kernel learners as shown in [Cumby and Roth, 2003b].

7 Conclusion

This work presents two paradigms for efficient learning and inference with relational data. The first framework addressed feature extraction in a functional setting through the defini-

tion of Relational Generation Functions. This framework can be viewed as providing a general basis for the second framework, abstracting away many of the details of the feature construction process. For example, we introduced the abstract notion of a *sensor* RGFs, which we allowed to construct features by inferring predicates from input instances, through external functions or any other means.

The second paradigm defined the notion of feature description logics - a relational language with clear syntax and semantics that can be used, via feature generation functions, to efficiently re-represent world observations in a way that is suitable for general purpose learning algorithms. We have shown that both these formalisms allow one to efficiently learn complex relational representations, in a system in which the basic components are articulated in a language whose structure and meaning are well understood.

It is important to point out that a wide family of feature description logics can be used within our framework. In fact, other CLASSIC-like description logics, as well as their probabilistic variations, could be incorporated into the framework with the addition of a Feature Generating Function for each. They could then participate as building blocks in the process of learning relations and predicates. For example, features generated in this framework need not be defined as Boolean. They can be associated with a real number, indicating the probability the feature holds in the interpretation, allowing a immediate use of P-CLASSIC like languages. This approach provides a different view on ways to extend such description languages, orthogonal to the one suggested by existing extensions, such as PRMs [Friedman *et al.*, 1999]. Unlike those extensions, which are more suitable to relational database-like (probabilistic) inferences, we provide a natural solution to learning predicates and relational structure, as seen in the examples pointed to in [Cumby and Roth, 2003a]. Furthermore, the syntactic framework allows us to use the FDL language for tasks other than pure feature extraction. For example, in [Cumby and Roth, 2003b], a family of relational kernel functions parameterized by descriptions in the language is developed for use with the Kernel Perceptron algorithm.

Some future directions include the use of our formalism to determine in a data driven way the level of abstraction of feature 'types' required for a given application; the development of nested and hierarchical FDL-based knowledge representations; and integrating our framework into a programming platform, allowing researchers to construct large scale learning-based architectures to solve complex AI problems.

References

[Borgida and Patel-Schneider, 1994] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *JAIR*, 1:277–308, 1994.

[Cohen, 1995a] W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *JAIR*, 2:501–539, 1995.

[Cohen, 1995b] W. Cohen. PAC-learning recursive logic programs: Negative result. *JAIR*, 2:541–573, 1995.

[Cook and Holder, 1994] D.J. Cook and L.B. Holder. Substructure discovery using minimum description length and background knowledge. *JAIR*, 1:231–255, 1994.

[Cumby and Roth, 2000] C. Cumby and D. Roth. Relational representations that facilitate learning. In *Proc. of KR'2000*, 2000.

[Cumby and Roth, 2002] C. Cumby and D. Roth. Learning with feature description logics. In *Proc. of ILP'02*, 2002.

[Cumby and Roth, 2003a] C. Cumby and D. Roth. Feature extraction languages for propositionalized relational learning. Technical Report UIUCDCS-R-2003-2346, UIUC Computer Science Department, May 2003.

[Cumby and Roth, 2003b] C. Cumby and D. Roth. On kernel methods for relational learning. In *Submission*, 2003.

[Dzeroski *et al.*, 1992] S. Dzeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proc. of COLT*, pages 128–135, 1992.

[Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI'99*, pages 1300–1309, 1999.

[Geibel and Wysotzki, 1996] P. Geibel and F. Wysotzki. Relational learning with decision trees. In *ECAI'96*, pages 428–432, 1996.

[Golding and Roth, 1999] A. R. Golding and D. Roth. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999.

[Gonzalez *et al.*, 2002] J. Gonzalez, L.B. Holder, and D.J. Cook. Graph-based relational concept learning. In *ICML'02*, 2002.

[Kersting and Raedt, 2000] K. Kersting and L. De Raedt. Bayesian logic programs. In *Proc. of the ILP'2000 Work-in-Progress Track*, pages 138–155, 2000.

[Khardon *et al.*, 1999] R. Khardon, D. Roth, and L. G. Valiant. Relational learning for NLP using linear threshold elements. In *Proc. of IJCAI*, 1999.

[Koller *et al.*, 1997] D. Koller, A. Levy, and A. Pfeffer. P-classic: A tractable probabilistic description logic. In *Proc. of NCAI'97*, pages 360–397, 1997.

[Kramer and Frank, 2000] S. Kramer and E. Frank. Bottom-up propositionalization. In *Proc. of the ILP-2000 Work-In-Progress Track*, 2000.

[Kramer and Raedt, 2001] S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical applications. In *Proc. ICML'01*, 2001.

[Kramer *et al.*, 2001] S. Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. In *Relational Data Mining*. Springer Verlag, 2001.

[Lavrac *et al.*, 1991] N. Lavrac, S. Dzeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Proc. of the 5th European Working Session on Learning*, pages 265–281, 1991.

[Levesque and Brachman, 1985] H. Levesque and R. Brachman. A fundamental tradeoff in knowledge representation and reasoning. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*. Morgan Kaufman, 1985.

[Littlestone, 1988] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Progamming*. Springer-verlag, 1987.

[Muggleton and De Raedt, 1994] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 20:629–679, 1994.

[Punyakanok and Roth, 2001] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *NIPS'2000*, pages 995–1001, 2001.

[Roth and Yih, 2001] D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *Proc. of IJCAI*, pages 1257–1263, 2001.

[Selman, 1990] B. Selman. *Tractable Default Reasoning*. PhD thesis, Dept. of Computer Science, Univ. of Toronto, 1990.

[Valiant, 1999] L. G. Valiant. Robust logic. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, 1999.