

Relational Representations that Facilitate Learning

Chad Cumby

Dan Roth

Department of Computer Science
 University of Illinois at Urbana-Champaign
 {cumby,danr}@uiuc.edu

Abstract

Given a collection of objects in the world, along with some relations that hold among them, a fundamental problem is how to learn definitions of some relations and concepts of interest in terms of the given relations. These definitions might be quite complex and, inevitably, might require the use of quantified expressions. Attempts to use first order languages for these purposes are hampered by the fact that relational inference is intractable and, consequently, so is the problem of learning relational definitions.

This work develops an expressive relational representation language that allows the use of propositional learning algorithms when learning relational definitions.

The representation serves as an intermediate level between a raw description of observations in the world and a propositional learning system that attempts to learn definitions for concepts and relations. It allows for hierarchical composition of relational expressions that can be evaluated efficiently on the observations and thus supports learning complex definitions by learning simple functions of the intermediate representations. The approach is illustrated using examples from natural language and visual processing.

1 Introduction

Given a collection of objects in the world, along with some relations that hold among them, a fundamental problem is how to learn definitions for some relations or concepts of interest in terms of the given

relations. This situation arises in a variety of AI problems such as natural language understanding related tasks, visual interpretation and planning. Examples include the problem of identifying noun or subject-verb phrases in a sentence in terms of the lower level information provided in the sentence, detecting faces in an image (phrased as “finding a definition for a *face* in terms of the information provided in the image”), or defining a policy that maps states and goals to actions in a planning situation.

One of the main challenges in these tasks is that they are knowledge intensive; a lot of information about the world and the domain (language, three dimensional world, faces, etc.) may be required to make a decision with respect to a given input. Consequently, a computational solution seems to require a significant learning component; learning is necessary both for the large scale knowledge acquisition required and in order to ensure robust behavior – given that a decision needs to integrate a large number of information sources evaluated on a naturally occurring input such as a sentence or an image. This realization is a prime reason for the gradual but dramatic shift in paradigm in AI. Recent works on natural language and visual processing inferences emphasize statistical learning methods and shy away from the previously dominant knowledge based paradigm; an approach which suggested [15] that decisions of this sort are to be inferred from knowledge that is programmed into the system in some uniform well defined language.

Statistical learning methods, which have been quite successful for low level identification tasks, have significant drawbacks as a comprehensive solution for higher level tasks that may require manipulating more expressive representations. Expressing complex concepts, we believe, will require representing relations over the input. In terms of these intermediate representations (e.g., the concepts of “part of speech” or “proper noun” for phrases, the concept of an “edge”

for vision processing, or that of “eye locations” for face recognition) the complex concepts will have simpler representations and learning it might be more manageable. Representing the intermediate notions, in turn, may rely on external information sources or be acquired in different ways, possibly unrelated to the current task, and might require some sort of inference.

We believe that the study of computational processes that interact with raw data and acquire the ability to make knowledge intensive inferences would benefit if learning, knowledge representation and inference processes could be studied within a unified framework. This research continues an effort to develop an integrated framework for the study of learning, knowledge representation and reasoning [10, 11, 24] and investigates a crucial stage in the computational process, studied in the context of a concrete collection of large scale problems.

We are interested in learning a definition for some relations or concepts of interest, a definition which can be evaluated efficiently given an instance in the domain. For this purpose we study intermediate knowledge representations, between a raw description of observations in the world and a learning system that attempts to learn the target definition. A search for representations of this sort need to address, as a minimum, the following:

- (i) Expressivity: A target definition should have a simple functional form (e.g., a rule, a linear function), and thus be learnable, in terms of the intermediate representation. It might be too complex and therefore unlearnable in terms of the raw input. The use of quantified representations is important in this respect.
- (ii) Evaluating the intermediate representation on the raw input should be done efficiently. This requires an efficient solution to the *subsumption* problem: given an input instance and a quantified intermediate representation, decide whether the quantified expression is satisfied by the input instance.
- (iii) Uniformity and well defined semantics. An intermediate knowledge representation might be acquired in a variety of ways. Fragments of it may be learned, programmed into the system, or inferred. These should be abstracted away to allow its use by a learning system in a uniform way.
- (iv) Finally, the intermediate representation, which serves as the input representation for the learning stage should be made available to it in a way that facilitates learning.

For knowledge intensive tasks, the number of potential information sources affecting each decision may be huge (e.g, any English word may be in the input sentence) – although only a small subset of these may be relevant to a decision. We therefore interpret the last requirement in a specific way – the intermediate representation should be available to propositional learning algorithms. The reason is that no other computationally viable approach can learn definitions that are non-trivial in size and structure, and do it in the presence of thousands of information sources or more.

This paper develops a intermediate representation language that addresses the above issues. After describing the learning scenario, the context in which the intermediate representations are to be used (Section 2) and the representation language - a restricted first order language (Section 3), Section 4 introduces the key notion of a *relation-generation function* (RGF). RGFs can be viewed as defining “types” of relations that may be of interest as intermediate representations. Given a raw description of observations in the worlds, relations of these “types” that actually *occur* in the input are generated and used as input “features” to the learning stage. The relations are elements in the representation language defined in Section 3 and, as we show, map efficiently to an input of propositional learning algorithms.

The simplest form of an RGF is a *sensor*, which is used to abstract the interaction with naturally occurring data. Sensors extract the primitive information available on the input instance. A relational calculus defined allows the composition of sensors into more expressive RGFs. Section 5 formalizes the notion of structural information. E.g., the structural information in an instance may simply describe the linear structure of a sentence or the (bottom-up) \times (left-right) structure of an image. RGFs utilize this information to efficiently generate expressive representations. For example, a construct of a “conjunction of consecutive elements” may give rise to an *edge* if applied to an appropriate “thresholded intensity” sensor; a structural construct applied to “adjacency” may give rise to recursive notions like “above”, or “before”. Computational issues involved in evaluating RGFs and generating the input for learning are also discussed.

The main contribution of this work is in providing a way to use expressive representations in the context of learning. Specifically, it

- defines an expressive language so that further learning stages can view it propositionally,
- formalizes an interface with naturally occurring

data,

- introduces the notion of RGFs which allows a for non-explicit, data-driven generation of relations, and
- formalizes restrictions which allow efficient evaluation.

The approach described here has been implemented and used in several large scale problems [17, 21, 22]. In Section 6 we briefly discuss implementation issues and a detailed example from the natural language domain that exhibits some of the notions introduced.

2 Learning Expressive Representations

This section describes the learning scenario at the basis of this work to provide the context in which relation-generation-functions and the relational representations they produced are being used. We would like to learn programs of the form¹

$$\text{subject}(x) \doteq f(\text{after}(x, \text{verb}), \\ \text{before}(x, \text{determiner}), \text{noun}(x), \dots)$$

so that when the program is evaluated on a given sentence its consequent has truth value “true” iff x is bound to a *subject* in the given sentence. The function f can be, in principle, any efficiently evaluable Boolean function. In particular, if f is a conjunction, this program is a single clause logic program.

We present this work in the context of a *supervised learning* paradigm, although nothing in the rest of the paper is restricted to this learning protocol. The main concern would be the representation of a domain instance, its mapping into a convenient representation for the learning procedure and, consequently, the representation of the learned program. The input to the learning algorithm is given in terms of an input sentence S along with an atom p that describes a property that holds in the sentence (a “label”); we would like to learn a definition (program) for this property in terms of the input sentence.

Inductive logic programming (ILP) [16, 3] is a natural paradigm for this learning problem. Computational limitations, however, render this approach inadequate for large scale knowledge intensive problems, like natural language inferences in a real world context. The

¹The example is meant only to illustrate the ideas and may not be accurate. Indeed, one reason learning is crucial in this domain is that it is hard to come up with concise rules that perform well.

limiting computational issues include both learnability [5, 4] (see [12] for a discussion) and the problem of *subsumption*² – deciding whether the premises of a rule cover a domain instance.

Two additional issues motivate the computational approach presented here. First, the need to learn in very large domains or in domains in which not all elements are known in advance; thus, it is impossible, or impractical, for a learning approach to write explicitly, in advance, all possible “features”. Second, for knowledge intensive tasks, the number of potential information sources that may affect each decision is very large but, typically, only a small number of them is actually relevant to a decision. A realistic learning approach needs therefore to be feature efficient [13] in that its complexity depends on the number of relevant features and not the global number of those in the domain.

2.1 The Learning Approach

The learning approach that serves as context to this work is a propositional learning approach in an infinite attribute domain [1], such as the one presented in [19, 2]. The learning procedure in this case learns a mapping $h : \{0, 1\}^\infty \rightarrow \{0, 1\}$ (or another discrete range). As input, the algorithm receives labeled instances $\langle (x, l) \rangle$, where an instance $x \in \{0, 1\}^\infty$ is presented as a list of all the *active* (of value 1) features in it. We typically assume that the learning scenario is *on-line*, although this is not required for the current setting, and do not assume that labels l are explicitly given; these can simply be designated features in the representation of $x \in \{0, 1\}^\infty$ (as done, for example, in the SNoW learning architecture [2]).

In the learning scenario suggested here, an instance $x \in \{0, 1\}^\infty$ indicates a collection of formulae in the relational language \mathcal{R} (Section 3) that have a truth value “true” in the domain instance. Given a domain instance (e.g., a sentence) a set of pre-existing relation generation functions (RGFs, Section 4) are evaluated on it and generate a collection of formulae that are *active* (have truth value “true”) in this sentence. The names of these formulae are the identifiers of the features presented to the propositional learning procedure.

Basic RGFs may be viewed themselves as programs of type listed above for *subject*(x) (with some restrictions), but for the purpose of this exposition the source of the RGF is irrelevant; they can be computed explicitly, learned, or inferred. The paradigm presented here

²Subsumption is NP-complete [8] and its cost may be practically implausible in complex domains [23].

is made possible due to:

- The restricted formulae generated by the RGFs (Sec. 3, [12]) allow the efficient mapping of quantified expressions describing a given instance to a propositional instance.
- The consistency of the RGFs – identifiers (formulae) for properties satisfied by new instances are generated consistently thus enabling generalization.

Finally, the success of this paradigm depends on that (i) RGFs are given the flexibility to generate a large number of formulae, which are active in observed instance, but many of which may not be relevant to the decision of interest and (ii) a feature efficient learning algorithm receiving this input can tolerate this without a significant additional cost.

Programs learned by the propositional learning algorithms can then be translated to quantified expressions, by substituting for each proposition its corresponding quantified formula. While the basic formulae are not as expressive as in the ILP paradigm, the structure of the rules learned is, in general, more expressive than those used in ILP, since the function f in the *subject* definition above need not be a conjunction³. The learning scenario studied here is that of learning one “rule” f at a time. Chaining of these programs [17] will not be discussed here.

3 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 first order language formulae in which there is only a single predicate in the scope of each variable.

Definition 3.1 *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, z a variable. Then $(\forall zF)$ and $(\exists zF)$ are atomic formulae.

³The paradigm has been applied with f as a linear function [17, 12] although others can be used.

Definition 3.2 *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* [12]. The informal semantics of the quantifiers and connectives is as usual.

Example 3.1 *An example of a bound atomic formula in \mathcal{R} in an NLP scenario could be a relation $\text{word}(\text{dog})$, which evaluates to true if there exists a word “dog” in the input instance. An unbound atomic formula could be a relation $\text{object}(x)$, which would evaluate to true if a word exists in the input instance which is an object in it. The truth value of this formula is to be determined by a RGF, based on information given in the input instance (Section 4).*

For formulae in \mathcal{R} , the *scope* of a quantifier is always the unique predicate that occurs with it in the atomic formula.

Definition 3.3 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function of n variables, and let F_1, F_2, \dots, F_n be formulae in \mathcal{R} . A clause is a formula of the form $f(F_1, F_2, \dots, F_n)$.*

This can be used, in particular, to define disjunctive, conjunctive and implication clauses.

3.1 Interpretation

\mathcal{R} is used as a language for representing knowledge with respect to a domain.

Definition 3.4 *A domain \mathcal{D} for the language \mathcal{R} is a collection D of elements along with*

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

Relational variables in \mathcal{R} receive their “truth values” in a data driven way, with respect to an observation.

Definition 3.5 *An instance is an interpretation [14] which lists a set of domain elements and the truth values of all instantiations of the predicates on them.*

Given an instance x , a formula F in \mathcal{R} is given a unique truth value, *the value of F on x* , defined inductively using the truth values of the predicates in F , and the semantics of the connectives. 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 3.1 *Let F be a formula in \mathcal{R} , x an instance, and let t_p be the time to evaluate the truth value of an atom p in F . Then, the value of F on x 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. This holds, similarly, for the following version of subsumption for formulae in \mathcal{R} .

Proposition 3.2 (subsumption) *Let x be an instance and let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function of n variables that can be evaluated in time t_f . Then the value of the clause $f(F_1, \dots, F_n)$ on x can be evaluated in time $t_f + \sum_{F_i} t_{F_i}$, where the sum is over all n formulae that are the arguments of f .*

4 Relation Generation Functions

Definition 4.1 *A formula in \mathcal{R} maps an instance x to its truth value in x . A formula is active in x if it has truth value true in this instance. We denote by X the set of all instances, the instance space. A formula $F \in \mathcal{R}$ is thus a relation over X , $F : X \rightarrow \{0, 1\}$.*

Example 4.1 (NLP) *An instance x could be an unordered collection of words: he, ball, the kick, would. Some active relations on this instance are $\text{word}(\text{he})$, $\text{word}(\text{ball})$, and $\text{tag}(\text{DET})$.*

Given an instance, we would like to know what are the relations (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 *relation generating 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 4.2 *Let \mathcal{X} be an enumerable collection of relations on X . A relation 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 “kinds” of formulae, or to parameterize over a large space of formulae. Only when an instance x is presented, a concrete formulae (or a collection of) is generated. An RGF can be thought of as having its own range \mathcal{X} of relations.

4.1 Relational Calculus

The family of relation generation functions for \mathcal{R} are RGFs whose output 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 relation 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 4.3 *A sensor is a relation generation function that maps an instance x into a set of atomic formulae in \mathcal{R} . When evaluated on an instance x a sensor s outputs all atomic formulae in its range which are active.*

Example 4.2 (NLP) *Given the instance x from EX. 4.1, two reasonable sensors to be given may be a sensor that observes the “word” relations and one for “tag” relations. The first could output the grounded relations: $\text{word}(\text{he})$, $\text{word}(\text{would})$, etc., and the existential relation $\text{word}(z)$. The tag sensor would need to be able to extract part-of-speech information from the instance to generate relations over it. In this case it could generate the grounded relations: $\text{tag}(\text{DET})$, $\text{tag}(V)$, $\text{tag}(\text{MOD})$, etc. and the existential relation $\text{tag}(z)$. An ISA sensor could be a sensor which returns a semantic class for each word (e.g., for the word “kick”, it might return the relation $\text{ISA}(\text{action})$). It could be implemented, say, by accessing an external knowledge source.*

Example 4.3 (VP) *Interesting sensors for visual applications could include an (intensity > n) RGF, which could generate relations such as $I > 50(\text{pixel5})$.*

Several mechanisms are used in the relational calculus to define the operations of RGFs. The following two restrict the range and the domain of an RGF, respectively.

Definition 4.4 Let C be a set of formulae. A conditioning operation $|C$ on an RGF r restricts r to output only formulae in C .

Example 4.4 There are several notations used to define commonly used subsets of formulae. For example, for a sensor s , the notation $s|_z$ ($s|_a$, resp.) restricts that range of s to include only the corresponding existential atomic formulae (ground atomic formulae, resp.)

Definition 4.5 Let E be a set of elements in the domain D . An RGF r is focused on E if, given an instance x it generates only formulae in its range that are active in x due to elements in E . The focused RGF is denoted $r[E]$.

Definition 4.6 The operation of a relation generation function (RGF) for \mathcal{R} is defined inductively as follows:

1. When evaluated on an instance x the sensor s outputs all active atomic formulae in its range.
2. If s and r are RGFs for \mathcal{R} , then so are $(\neg s)$, $(s \& r)$, $(s|r)$.
 - i The formulae in the output of $(\neg s)$ are active formulae of the form $\{\sim F\}$, where F is in the range of s (evaluated on x).
 - ii The formulae in the output of $(s \& r)$ are active formulae of the form $F \wedge G$, where F is in the range of s and G is in the range of r (evaluated on x).
 - iii The formulae in the output of $(s|r)$ are active formulae of the form $F \vee G$, where F is in the range of $s|_C$ and G is in the range of $r|_{C'}$, and C , $(C'$, resp.) are formulas in $s[D]$ ($r[D]$, resp.)

Notice that for disjunctions it is natural to define C, C' so that they condition the RGFs to be either existential (e.g., $s|_z$) or a collection of ground formulae. The necessity to condition prevents the generation of trivial disjunctive formulae. For conjunction and negation it is also possible to focus first on each element in x separately.

4.1.1 Nesting

The inductive definition 4.6 indicates that operands in the relational calculus may be RGFs themselves. This is a “symbolic” calculus that is always “compiled” into formulae in \mathcal{R} , however, and not a recursive operation. True recursion will be introduced in the next section.

4.1.2 Naming

Naming is a mechanism that allows the definition of new RGFs in terms of existing RGFs. The syntax of the operation is

$$A \doteq f(s_1, s_2, \dots, s_n),$$

where f is any operation in the relational calculus, including nesting. Given an instance x , A outputs identical formulae to those produced by $f(s_1, s_2, \dots, s_n)$. The use of mnemonics contributes only to simplify “programming” and analysis.

Example 4.5 A relation that designates that a certain (or any) word is a modal, can be generated using an RGF that makes use of the word sensor and a conditioned version of the tag sensor. E.g., $word \& tag|_{tag(MOD)}$ would produce active formulae such as $word(can) \wedge tag(MOD)$ and $word(x) \wedge tag(MOD)$. Then, we may want to name it using a useful mnemonic: $MODAL \doteq word \& tag|_{tag(MOD)}$

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 is available. We abstract this using the notion of a *structural domain* that we define 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.

5.1 Structured Instances

Definition 5.1 Let D be the set of elements in the domain. A structured instance O is a tuple $(V, E_1, E_2, \dots, E_k)$ where $V \subseteq D$ is a set 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 i th structure of the instance O and is restricted to be an acyclic graph on V .

Example 5.1 (NLP) Let D be the set of all words in English. A structured instance can correspond to a sentence, with V , the set of words in the sentence and $E = E_1$ describes the linear structure of the sentence. That is, $(v_i, v_j) \in E$ iff the word v_i occurs immediately before v_j in the sentence.

Example 5.2 (VP) Let D be the set of all positions in a 100×100 gray level image. A structured instance

can correspond to a sub-image, such that V is the set of pixels in the sub-image and $E = E_1$ describes the top-down and left-right adjacencies in it. That is, $(v_j, v_k) \in E$ iff the pixel v_j is either immediately to the left or immediately above v_k .

The interaction of the system with the World is done via a collection of structured instances along with a set of sensors which can operate on them. Sensors may extract information directly from the input, such as word, tag, or intensity, but can also utilize outside knowledge in processing the input. More complex RGFs are then defined as functions of the sensors, and can be used to extract further information from the instances.

5.2 Structural Operations

We now augment the relational calculus of Section 4.1 by adding structural operations. These operations exploit the structural properties of the domain as expressed in the graphs G_i s in order to define RGFs, and thereby generate non-atomic formulae that may have special meaning in the domain.

Definition 5.2 Let s_1, s_2, \dots, s_k be RGFs for \mathcal{R} . $\text{colloc}_g(s_1, s_2, \dots, s_k)$ is a restricted conjunctive operator that is evaluated on a chain of length k in the g th structure of the given structured instance. Specifically, let $O = \{G_i\}_1^m$ be a structured instance and let v_1, v_2, \dots, v_k be a chain in G_i . The formulae generated by $\text{colloc}_i(s_1, s_2, \dots, s_k)$ are those generated by $s_1[v_1] \& s_2[v_2] \& \dots \& s_k[v_k]$, where by $s_j[v_j]$ we mean here the RGF s_j focused to $\{v_j\} \subseteq D$, and the $\&$ operator is defined as in Definition 4.6. Notice that each RGF in the conjunctions may produce more than one formulae.

Example 5.3 (NLP) Say we are interested in a Subject-Verb relation, and assume that the structured instance consists of, in addition to the linear structure of the sentence (G_1), a graph G_2 encoding functional relationships among the words. Using a role sensor we can produce the Subject-Verb relation using the RGF $\text{colloc}_2(\text{role}_{|\text{role}(\text{Subject})}, \text{role}_{|\text{role}(\text{Verb})})$.

Example 5.4 (VP) To find an edge relation in an image, we might define an Edge RGF thus: $\text{EDGE} \doteq \text{colloc}_1(s, s, s, s, s)$ where s is, say, a sensor producing active relations for pixels with intensity value above 50.

Definition 5.3 Let s_1, s_2, \dots, s_k be RGFs for \mathcal{R} . $\text{scolloc}_g(s_1, s_2, \dots, s_k)$ (sparse colloc) is a restricted conjunctive operator that is evaluated on a chain of

length n ($k \leq n$) in the g th structure of the given structured instance. Specifically, let $O = \{G_i\}_1^m$ be a structured instance and let v_1, v_2, \dots, v_n be a chain in G_i . Formulae are generated by $\text{scolloc}(s_1, s_2, \dots, s_k)$ as follows: For each subset $v_{i_1}, v_{i_2}, \dots, v_{i_k}$ of elements in v , such that $i_j < i_l$ when $j < l$, all the formulae: $s_1[v_{i_1}] \& s_2[v_{i_2}] \& \dots \& s_k[v_{i_k}]$, are generated, where by v_{i_i} we mean here the instance $\{v_{i_j}\} \subseteq D$, and the $\&$ operator is defined as in Definition 4.6.

5.3 Focus-Word Centered Representation

The structural information also provides an easy way to *focus* the RGFs (Def. 4.5). For example, defining a set of elements for the focus set E in $s[E]$ 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*, *scolloc* operations, we can restrict the chains to *start* at a specific element $d \in D$ or to *contain* it. Notice that if, for the given instance $O = (V, G)$, we have that $d \notin V$, then the output is an empty set of formulae.

Example 5.5 The power of the focus device can be shown in the following example for defining an RGF for gender agreement within an observation. Say we have a gender sensor, which will either output the relations *gender(male)* or *gender(female)* for gendered words or null otherwise. We can then define the RGF: $\text{gender}[\text{focus}]_{|\text{gender}[\sim \text{focus}]}$. That means that we are evaluating the gender RGF taking the focus element as the instance x , but that we are conditioning the output of this RGF to be active only when it produces a formula in the set of formulae produced by $\text{gender}[\sim \text{focus}]$. This essentially means that the RGF produces an active formulae of the form *gender(male)* or *gender(female)* iff the gender of the focus element matches the gender of another word in the sentence.

5.4 Complexity of Evaluating RGFs

Notice that while all the formulae generated are in \mathcal{R} and the structural operations only provides a way to specify which atomic formulas should be evaluated on which object, they actually provide a powerful mechanism that goes beyond \mathcal{R} . While variables in formulae in \mathcal{R} have a single predicate in their scope, the structural properties provides a way to go beyond that (as shown in Ex. 5.5), but only in a restricted way that is efficiently evaluated.

This, of course, can be done within the language of first order logic by, say, adding predicates that encode the structural properties, e.g., an *edge(x, y)* predicate.

This way, however, formulae will not be in \mathcal{R} anymore, and the problem of evaluating a clause given an instance would become intractable. Structural operations defined on \mathcal{R} allow us to define RGFs the constrain formulae evaluated on different objects without incurring the cost usually associated with enlarging the scope of free variables. This is done by enlarging the scope only as required by the structure of the domain, modeled by the G_i s. Exploiting the structure allows for efficient evaluation of the formulae with the only additional cost being that of finding chain in the graph.

Proposition 5.1 *Let O be a structured instance, with $n = |V|$. Let t be an upper bound on the time to evaluate sensors in O .*

(i) *The time to evaluate a conjunctive RGF of size k is bounded by $T_k = k \cdot n \cdot t$.*

(ii) *The time to evaluate a colloc RGF of size k is bounded by $c \cdot T_k$, where c is the number of chains of size k in the given graph.*

(iii) *The time to evaluate an scolloc RGF of size k is bounded by $c \cdot \binom{n}{k} \cdot T_k$, where c is the number of chains of size n in the graph.*

Structural instances also provide the ability to deal with recursive RGFs. Although recursion is not supported in \mathcal{R} , it is inherited from the graph. Since a *path* relation in a graph is recursive over the *edge* relation, *colloc* and *scolloc* inherit recursion at no additional cost, providing, for example, the ability to generate “above” RGF from the “on” sensor [9].

6 An Example

The following example serves to illustrate the language and knowledge representations described in this paper. It utilizes basic sensors, structural operations with recursion, implicit and explicit focus definition, and conditioning. It is based on an implemented system that has been used to support learning for knowledge intensive tasks [19, 22, 17, 21].

The problem considered is learning *verb representations*. Once a system has learned a “definition” for when to use each verb it could decide, for example, to use *throw* rather than *remove* in a specific context [7, 18]. This problem is subtle, because the context for many verbs is similar enough that it may be hard to learn good representations for them only in terms of surrounding words and part of speech information. Verb representations are likely to depend on complicated relations that hold in the sentence and on

semantic information [6].

6.1 Input

The input consists of a structured instance O that is given along with the sensors that can act on it. Let $O = \{V, G_1, G_2\}$ be the structured instance, with V - the set of all words in the sentence. G_1 corresponds to the linear structure of the sentence and G_2 to a structure encoding *functional dependencies* in the sentence⁴. The sensors given to operate on O consist of:

word - Outputs active relations of the form $word(a)$ where a is the spelling of a word $v \in V$. e.g. $word(dog)$, $word(locomotion)$, or $word(\mathbf{z})$.

suffix - Outputs active relations of the form $suffix(a)$ where a is the suffix of a word $v \in V$, or $suffix(\mathbf{z})$, when any one of a fixed set of suffixes is active.

role - Outputs active relations of the form $role(a)$ where a is the function of a word $v \in V$ in a functional dependency grammar. e.g. $role(Subject)$, $role(Main_Verb)$, or $role(\mathbf{z})$.

6.2 Useful Relations

We now devise a set of “kinds” of relations that may be useful for learning verb representations. It is important to note that we only need to define “kinds” of relations, and not to explicitly define the relations themselves; it is clear that different instantiations of the “kinds” (RGFs) defined would be relevant for different verbs. Also, one can suggest a superset of “kinds”, and allow the data driven learning process decide which are indeed relevant.

The following presentation is not complete, but would be enough to illustrate the approach. We denote by *target* the verb for which a representation is sought. Naturally only sentences in which *target* occurs are considered. Useful relations could include:

1. There exists a target word a .
2. There exists some suffix for the target word.

⁴This can be produced by a functional dependency grammar (FDG) of a sentence, which gives each word a specific function, and then structures the sentence hierarchically based on it. The graph is a tree that is rooted at the main verb of the sentence. This structure could be generated by an external rule-based parser or a learned one [17]; this issue is orthogonal to the current discussion which abstracts these issues into the sensors.

3. There exists a word a such that the subject of the target verb is a .
4. There exists a word a such that the direct object of the target verb is a .

Written explicitly, these relations are not in the language \mathcal{R} . However, they can all be defined as structural conjunctions of atomic formulae applied over a restricted domain. These conjunctions *will* be in \mathcal{R} , and allow the mapping to Boolean variables. We now present RGFs that produce relations of these forms.

6.3 Relation Generation Functions

Relations of the type (1) above, will be produced using the *word* sensor restricted to the *target* focus, written as $word[target]$.

The relations in (2) can be generated using the RGF

$$suffix_{|z}[target].$$

This will create the existential relation $suffix(\mathbf{z})$ when the target word has been suffixed, but will not indicate what suffix it has. (It is possible to generate the ground version as well.)

For (3) we can use the RGF

$$colloc_2(role_{|role(Subject)}\&word, word[target]).$$

This RGF attempts to find a chain $v', v'' \in V$ in the second structure where the conjunction:

$$role_{|role(Subject)}[v']\&word[v']\&word[v''],$$

is active and $v'' = target$. If it does this, then it outputs an active formula of the form

$$role(Subject)\wedge word(a)\wedge word(b)$$

where a, b correspond to two words in V . Due to the colloc operation, a is the subject word, and b is the target word. Note that we can assume that the subject word will always precede the target word in G_2 due to the structure in the domain (i.e., the graph representing the dependency grammar).

The RGF for (4) is

$$colloc_2(role_{|role(Object)}\&word, word[target]),$$

which behaves as the one for (3), producing active formulae for all chains with objects pointing to the target.

Finally, let O correspond to the sentence:

The boy batted the ball,

with $target = batted$. The above RGFs would generate the following intermediate representation:

$$\{word(batted),$$

$$suffix(\mathbf{z}),$$

$$role(Subject)\wedge word(boy)\wedge target,$$

$$role(Object)\wedge word(ball)\wedge target\}$$

which are likely to be useful for learning verb representations. Note that these RGFs produce both quantified and ground formulae. The specific formulae produced depend on the data observed. As pointed out in Section 2, the consistency of the generation guarantees that similar relations that hold in different instances would be given the same identifiers and will thus facilitate generalization. In addition, the learning procedure would “decide” which of the formulae produced actually contribute to the sought after definition, and would produce a program that combines these appropriately.

7 Conclusion

This work describes a knowledge representation paradigm that is developed as part of a research program that develops a unified approach for knowledge representation, learning and inference. In particular, the representations studied here support efficient, data-driven generation of expressive representations that can be used by propositional learning algorithms. In addition to allowing the learnability of relational programs using propositional means, this approach provides a uniform way to abstract domain information that is available at a given stage of the learning process, using the notion of *sensors*. Domain information may be available from many sources and modalities and in many forms. These can all be viewed as sensors and integrated in a unified way into the developed paradigm.

Sensors also provide a clean solution from a software engineering point of view. A given learning architecture can be used to learn at several levels in the hierarchy, and for many domains. To do that, one has to supply it as input observations along with sensors that can interpret it. This approach has already been used successfully in several applications and is further developed and abstracted in [20]. This work opens up several directions for further work from the learning as well as the knowledge representation points of view; we hope that work along these lines would contribute to the development of a unified paradigm.

Acknowledgments

We are grateful to Jerry DeJong and Roni Khardon for useful comments on an earlier draft.

This research is supported by NSF grants IIS-9801638 and SBR-987345.

References

- [1] A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, October 1992.
- [2] A. Carleson, C. Cumby, J. Rosen, and D. Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May 1999.
- [3] W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *Journal of Artificial Intelligence Research*, 2:501–539, 1995.
- [4] W. Cohen. PAC-learning recursive logic programs: Negative result. *Journal of Artificial Intelligence Research*, 2:541–573, 1995.
- [5] S. Dzeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proceedings of the Conference on Computational Learning Theory*, pages 128–135, Pittsburgh, PA, 1992. ACM Press.
- [6] Y. Even-Zohar and D. Roth. Learning in NLP: Incorporating knowledge into the process. 2000. Submitted.
- [7] A. R. Golding and D. Roth. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999. Special Issue on Machine Learning and Natural Language.
- [8] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In *Proc. of the 8th conference on Automated Ddeduction*, volume 230, pages 489–495. Springer Verlag, 1986.
- [9] R. Khardon. Learning to take actions. *Machine Learning*, 35(1):57–90, 1999.
- [10] R. Khardon and D. Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, Sept. 1997. Earlier version appeared in AAAI-94.
- [11] R. Khardon and D. Roth. Learning to reason with a restricted view. *Machine Learning*, 35(2):95–117, 1999.
- [12] R. Khardon, D. Roth, and L. G. Valiant. Relational learning for NLP using linear threshold elements. In *Proc. Int'l Joint Conference on Artificial Intelligence*, pages 911–917, 1999.
- [13] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
- [14] J. W. Lloyd. *Foundations of Logic Programming*. Springer-verlag, 1987.
- [15] J. McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic information processing*, pages 403–418. MIT Press, 1968.
- [16] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 20:629–679, 1994.
- [17] M. Munoz, V. Punyakanok, D. Roth, and D. Zimak. A learning approach to shallow parsing. In *EMNLP-VLC'99, the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, June 1999.
- [18] J. Rosen. Scaling up context sensitive text correction. Master's thesis, UIUC, Department of Computer Science, May 1999.
- [19] D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proc. National Conference on Artificial Intelligence*, pages 806–813, 1998.
- [20] D. Roth. Learning based programming. Technical Report UIUCDCS-R-99-2127, UIUC Computer Science Department, October 1999.
- [21] D. Roth, M-H. Yang, and N. Ahuja. A SNoW-based face detector. In *NIPS-12*, 2000.
- [22] D. Roth and D. Zelenko. Part of speech tagging using a network of linear separators. In *COLING-ACL 98, The 17th International Conference on Computational Linguistics*, pages 1136–1142, 1998.
- [23] M. Sebag and C. Rouveirol. Any-time relational reasoning: Resource-bounded induction and deduction through stochastic matching. *Machine Learning*. To Appear.
- [24] L. G. Valiant. Robust logic. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, 1999.