Functional Subsumption in Feature Description Logic

Rodrigo de Salvo Braz[1] and Dan Roth[2]

[1] braz@uiuc.edu
[2] danr@cs.uiuc.edu
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N. Goodwin
Urbana, IL 61801-2302

Summary. Most machine learning algorithms rely on examples represented propositionally as feature vectors. However, most data in real applications is structured and better described by sets of objects with attributes and relations between them. Typically, ad-hoc methods have been used to convert such data to feature vectors, taking, in many cases, a significant amount of the computation. Propositionalization becomes more principled if generating features from structured data is done using a formal, domain independent language that describes feature types to be extracted. This language must have limited expressivity in order to be useful while some inference procedures on it are still tractable.

In this chapter we present Feature Description Logic (FDL), proposed by Cumby & Roth, where feature extraction is viewed as an inference process (subsumption). We also present an extension to FDL we call Functional FDL. FDL is ultimately based on the unification of object attributes and relations between objects in order to detect feature types in examples. Functional subsumption provides further abstraction by using unification modulo a Boolean function representing similarity between attributes and relations. This greatly improves flexibility in practical situations by accomodating variations in attributes values and relation names, incorporating background knowledge (e.g., typos, number and gender, synomyms etc). We define the semantics of Functional subsumption and how to adapt the regular subsumption algorithm for implementing it.

1 Introduction

Research in Machine Learning has shifted in the last few years to concentrate largely on the study of theoretical and algorithmic issues that pertain to "how to learn the best classifier or function approximator for a given data set". Work in this direction has been quite successful in developing appropriate theoretical understanding as well as practical algorithmic approaches that have been shown successful in a large number of applications. One of the key

abstractions made in this view is in the assumption that the input to the learning algorithm is given in a form of feature vectors. With the maturity of these models, though, it has become (again) clear that a key issue that needs to be addressed is that of "what are the features", namely, how should domain elements be represented in a way that learning algorithms can effectively use them. This work focuses on this stage – the problem of mapping domain elements into representations that are useful as inputs to a learning algorithm.

The process of re-representing domain elements, in a way that takes into account what is known (or previously learned) about the domain and the problem, is necessary in order to facilitate learning in complex situations, when learning requires exploiting the structural and relational information within the domain elements. Indeed, researchers and practitioners spend a significant portion of their time on the problem of feature engineering and a significant amount of computing cycles goes into this stage in the process. This computation is typically left unreported in research papers, where the starting point is usually *after* features have already been extracted but, clearly, are significant in applications. Mapping domain elements into representations that can be processed by learning algorithms (e.g., feature vectors), that is, extracting the relevant substructures and properties of the domain elements – typically done today in ad-hoc ways – might be computationally expensive and should be studied as an integral part of the learning process. The benefits of this integration are numerous, one of them being the possibility for kernels directly over structured examples, as in [5].

We argue, following [4], that the process of converting a domain element into a feature based representation is best viewed as an inference process, and we formalize it this way. At the heart of our approach is a knowledge representation language – Feature Description Logic (FDL) – that we use in order to represent both domain elements and potentially expressive "types" of features – along with Feature Generation Functions (FGFs), a mechanism to convert domain elements, given the features, and re-represent them in a way that can be used by general purpose learning algorithms.

FDL is a specific Description Logic (DL), a knowledge representation that has been found useful in describing sets of individuals, or *concepts*, by specifying constraints which all members of the set must obey, and in establishing taxonomies involving those sets. DLs strike a good balance between expressivity and tractability, achieved by a careful choice of the types of constraints allowed. This choice makes description logics less expressive than full First Order Logic but expressive enough to allow efficient inference in structured relational domains. Of particular use for us will be *subsumption*, a standard inference task in DLs. A description C is said to subsume a description D if and only if all the individuals described by D must also be described by C.

Within the FDL framework, examples can be richly structured objects composed by many individuals with relations among them. As in [4], we view examples as *concept graphs*, where nodes represent individuals with primitive

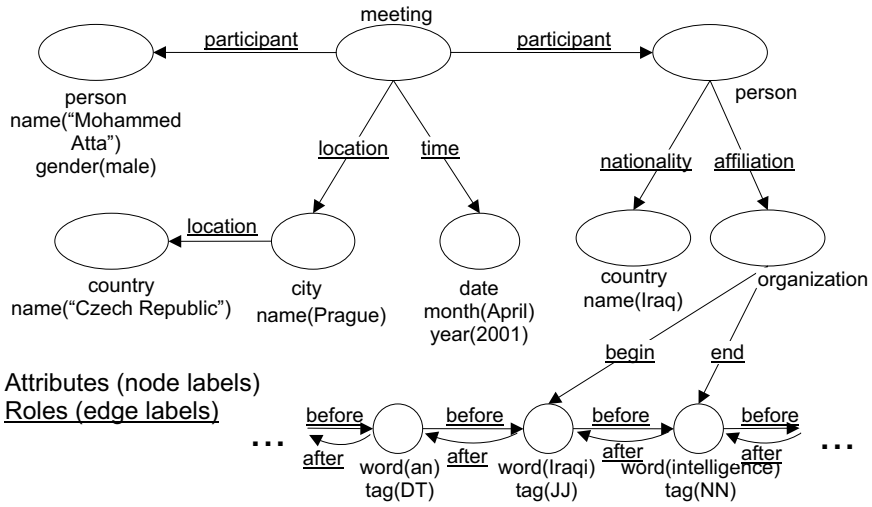

Fig. 1. A Concept Graph describing an example from the Information Extraction domain. The bottom right shows a simple structural representation, consisting of the words, their parts-of-speech and their order. The rest of the graph contains richer structural and semantic information.

properties (node labels), while relations (or *roles*) among the individuals are represented by edge labels, as depicted in Fig. 1.

Feature types are descriptions in the FDL language and can be interpreted as functionals that, when evaluated on examples, indicate whether there is some individual in it contained in the concept specified by the description.

Determining whether an individual in a concept graph is part of a feature type concept can be done by deciding whether the feature type description subsumes that individual's most specific container, a description easily obtained from the concept graph. The process of Feature Extraction thus relies on subsumption and becomes an inference process.

In Fig. 2 we show the process of converting a domain element into a feature based representation. In this figure we describe not only data, but also feature types. Feature types are usually described in the FDL language, but before we introduce it we represent them as graphs too, relying on the reader's intuition as to what they mean, and noting that they can be seen as abstractions of individuals, activating the corresponding feature in the final feature vector if an individual of that type occurs in the example.

Further abstraction can be obtained with *Functional Subsumption*. Subsumption has, as its most primitive operation, a unification step between

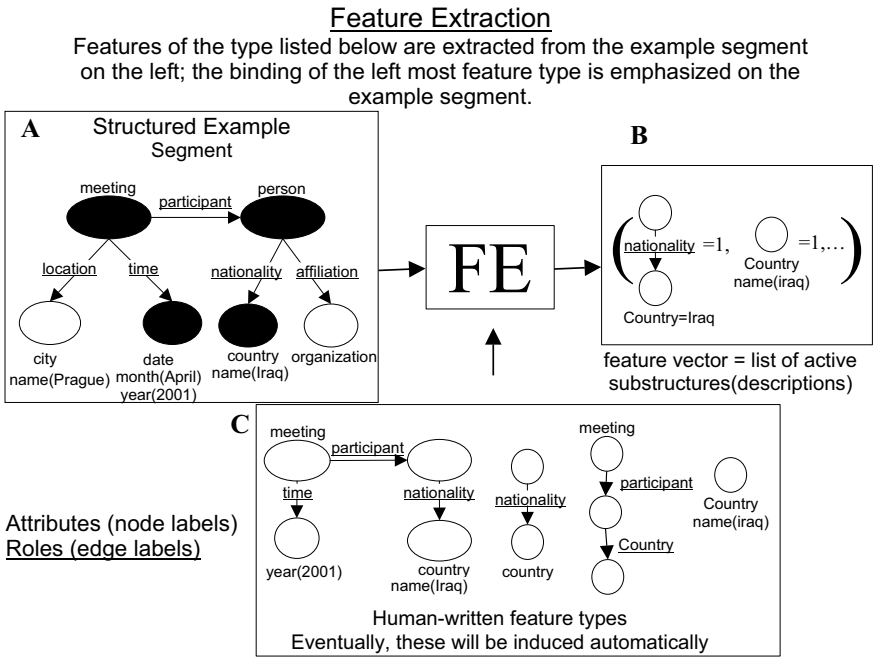

Fig. 2. A conceptual view of the FDL process; a concept graph represents a domain element (A); several (typically simpler) other graphs (corresponding to FDL descriptions) represent feature types (B). If a feature type description describes a class of individuals of which some individual in the domain element is a member, a corresponding feature is *active* and is listed in the resulting feature vector, used as the input for learning (C).

attributes in both descriptions, as well as roles. This unification relies on the identity of attribute and roles names, as well as parameter values. We generalize subsumption by lifting the restriction of identity and replacing it by some pre-specified function indicating what is to match and what is not. For example, in natural language processing applications, when unifying an FDL description to an example graph, we may want the verb "buy" to match other verbs including, say, "purchase", or the typo "buuy", or the variation "bought."

In the rest of this paper we first present $\mathcal{FDL}$, the Description Logic used in the FDL framework, and then the framework itself. At last we present Functional Subsumption and a modified subsumption algorithm for it.

2 The $\mathcal{FDL}$ language

This section presents syntax and semantics of the language $\mathcal{FDL}$, used in the FDL framework. This language resembles $\mathcal{EL}$ from [2], based on existential

quantification, the main difference being that our attributes are richer, being possibly parameterized.

2.1 $\mathcal{FDL}$ syntax

Definition 1. *A* $\mathcal{FDL}$ alphabet *consists of a role symbol set Role, a unary attribute symbol set UnAttr, a binary attribute symbol set BinAttr and a value symbol set Val.*

The set of descriptions in $\mathcal{FDL}$ over a given alphabet is defined in a recursive fashion:

Definition 2. *The set of descriptions in* $\mathcal{FDL}$ *over the alphabet* $(Role, UnAttr, BinAttr, Val)$ *is defined as follows:*

- ** is a description.*
- *If* $o \in UnAttr$, $t \in BinAttr$ *and* $v \in Val$, *then o, t and* $t(v)$ *are* atoms. *The set of all atoms is denoted Atom.*
- *If* $A \in Atom$, *then A and* (NOT *A*) *are* (positive and negative, respectively) literals. *The set of all literals is denoted Literal.*
- *if* $A \in Literal$, *then A is a description (also called an* attribute description*).*
- *if* $r \in Role$ *and C is a description then* (SOME *r C*) *is a description.*
- *if* $C_1, C_2, \ldots, C_n$ *are descriptions, then* (AND C_1 C_2 $\ldots C_n$) *is a description, called a* conjunctive description, *or simply* conjunction. *Each* C_i *is a* conjunct *of C.*

The symbols NOT, SOME, and AND are known as connectives *or* constructors.

A description D is an immediate subdescription *of a description C (denoted* $D \gg C$*) if* $C =$ (SOME *r D*) *for some r, or C is a conjunction and D one of its conjuncts. D is a* subdescription *of C if it is C or it is an immediate subdescription of C or it is a subdescription of an immediate subdescription of C.*

The depth *of a description D is* 0*, if the description is an attribute description, and* $d+1$ *otherwise, where d is the maximum depth of any immediate subdescription of D.*

The length *of a description C is* $1 + \sum_{i:C_i \gg C} length(C_i)$ *(therefore attribute descriptions have length 1).*

2.2 $\mathcal{FDL}$ semantics

Definition 3. *A* domain *for a description logic language is* $\Delta = (X, V)$ *consisting of two disjoint sets of* individuals, *X, and* values, *V. Informally, X represents the individuals we are interested in, while V contains the parameters of attributes of these individuals.*

A concept *is a set of individuals contained in* X.

A pre-interpretation $\cdot^{\mathcal{K}}$ *of an alphabet* $(Role, UnAttr, BinAttr, Val)$ *is a function which maps role, attribute and value symbols into their* extensions *according a domain* $\Delta = (X, V)$. *Role symbol extensions are binary relations over* $X \times X$, *unary attribute symbol extensions are unary relations over* X *(that is, subsets of* X*) and binary attribute symbol extensions are binary relations over* $X \times V$. *If* $(i, j) \in r^{\mathcal{K}}$ *for some role symbol* r, j *is said to* fulfill *role* r *for* i *in* $\mathcal{K}$.

Given a pre-interpretation $\cdot^{\mathcal{K}}$ *over* $(Role, UnAttr, BinAttr, Val)$, *we can define an* interpretation *to be a function* $\cdot^{\mathcal{I}}$ *extending* $\cdot^{\mathcal{K}}$. *It maps role, attribute and value symbols to the same extensions as* $\cdot^{\mathcal{K}}$ *and atoms, literals and descriptions to their extensions as defined in the second column of the table below. Let* C *and* C_i *be arbitrary descriptions,* r *any role symbol,* o *any unary attribute symbol,* t *any binary attribute symbol,* A *any atom and* v *any value symbol. Note that the extensions of descriptions are always concepts.*

Description form	*Description's extension*
$*$	X
(SOME r C)	$\{x \mid \exists y (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
o	$\{x \mid x \in o^{\mathcal{K}}\}$
t	$\{x \mid \exists w \in V : (x, w) \in t^{\mathcal{K}}\}$
$t(v)$	$\{x \mid (x, v^{\mathcal{I}}) \in t^{\mathcal{K}}\}$
(NOT A)	$X - A^{\mathcal{I}}$
(AND C_1 C_2 $\ldots C_n$)	$\{x \mid x \in \bigcap_{i \in \{1,\ldots,n\}} C_i^{\mathcal{I}}\}$

A model $(\mathcal{I}, \Delta)$ *is a pair of a domain* Δ *and an interpretation* $\mathcal{I}$ *on that domain. We often refer to a model simply by its interpretation's name, omitting the domain.*

We say that a description C has a model $\mathcal{I}$, *or that* $\mathcal{I}$ is a model of C, *if* $|C^{\mathcal{I}}| > 0$.

Example 1. Let a domain $\Delta = (X, V)$ be formed by $X = \{john, mary, paul, susie\}$ and $V = \{1, 2, \ldots\}$, and an $\mathcal{FDL}$ alphabet $(Role, UnAttr, BinAttr, Val)$ be formed by $Role = \{married, child\}$, $UnAttr = \{male, female\}$, $BinAttr = \{age\}$ and $V = \{1, 2, \ldots\}$. A model $\mathcal{I}$ is defined on Δ such that

- $married^{\mathcal{I}} = \{(john, mary), (paul, susie)\}$
- $child^{\mathcal{I}} = \{(john, paul), (mary, paul)\}$
- $male^{\mathcal{I}} = \{john, paul\}$
- $female^{\mathcal{I}} = \{mary, susie\}$
- $age^{\mathcal{I}} = \{(mary, 54), (paul, 30), (susie, 24)\}$

Then some description extensions are:

- (SOME $married$ $female$)$^{\mathcal{I}} = \{john, paul\}$
- (SOME $married$ (AND $female$ $age(54)$))$^{\mathcal{I}} = \{john\}$
- (AND $female$ age)$^{\mathcal{I}} = \{mary, susie\}$

- (SOME *child* (SOME *married* *))$^{\mathcal{I}} = \{john, mary\}$

Definition 4. *Two descriptions C and D are* equivalent *if, for any interpretation $\mathcal{I}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$.*

2.3 Subsumption

Given the definition for the extension of descriptions, we can define our basic inference task, which is deciding whether a description subsumes another.

Definition 5. *A description C* subsumes *a description D (denoted as $C \sqsupseteq D$) if for every model $\mathcal{I}$, $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$.*

Subsumption for atom descriptions is easily determined on a syntactic basis, which is the basis for deciding subsumption in general later on:

Theorem 1. *Let A and B be atom descriptions. Then $A \sqsupseteq B$ if and only if $A = B$ or $A = t$ and $B = t(v)$ for some $t \in BinAttr$ and $v \in Val$.*

Proof. ($\Rightarrow$) We assume $A \sqsupseteq B$. Because A and B are atoms, one of four possible cases must be true:

- A has attribute symbol c_1 and B has attribute symbol c_2 for some distinct $c_1, c_2 \in UnAttr \cup BinAttr$.
- A and B are identical.
- $A = t$ and $B = t(v)$ for some $t \in BinAttr$ and $v \in Val$.
- $A = t(v)$ and $B = t$ for some $t \in BinAttr$ and $v \in Val$.

The first case cannot occur because we can easily build an interpretation in which an individual i is in the extension of B but not A (since they have different attribute symbols), which would contradict $A \sqsupseteq B$.

The fourth case cannot occur because we can easily build an interpretation in which an individual i is such that $i \in t(v')^{\mathcal{I}}$ for some $v' \in Val$, for $v'^{\mathcal{I}} \neq v^{\mathcal{I}}$, again causing it to belong to the extension of B but not A, which would contradict $A \sqsupseteq B$.

Therefore at least one of the second or third cases will be true, satisfying the theorem's conclusion.

($\Leftarrow$) We assume that one of the two conditions is true:

- $A = B$ or
- $A = t$ and $B = t(v)$ for some $t \in BinAttr$ and $v \in Val$.

If the first case is true, then for every interpretation $\mathcal{I}$, $A^{\mathcal{I}} = B^{\mathcal{I}} \Rightarrow A^{\mathcal{I}} \supseteq B^{\mathcal{I}} \Rightarrow A \sqsupseteq B$.

If the second case is true, for any interpretation $\mathcal{I}$, each individual i in $B^{\mathcal{I}}$ is such that $(i, v^{\mathcal{I}}) \in t^{\mathcal{I}}$, so there exists $v \in Val$ such that $(i, v^{\mathcal{I}}) \in t^{\mathcal{I}}$, which determines that $i \in A^{\mathcal{I}}$. Thus $A \sqsupseteq B$. □

Theorem 2. *Let A and B be literal descriptions. If A and B are both positive literals, $A \supseteq B$ can be determined according to Theorem 1, and if A and B are both negative literals* (NOT A') *and* (NOT B') *respectively, $A \supseteq B$ if and only if $B' \supseteq A'$.*

Proof. ($\Rightarrow$) Let A and B be literals such that $A \supseteq B$. Then A and B cannot have different polarities as literals. If they had, either B is negative and A is positive, or the other way around. In the first case, the model with a single individual with all attribute extensions being empty will be a model for B but not A. In the second case, a model with a single individual belonging to all attribute extensions will be a model for B but not for A. So in either case $A \not\supseteq B$.

Therefore A and B have the same polarity. If they are both positive, the hypothesis ensures that the first condition is satisfied. If they are both negative of forms (NOT A') and (NOT B') and $A \supseteq B$, then for every interpretation $\mathcal{I}$, $B^{\mathcal{I}} \subseteq A^{\mathcal{I}}$. Since all individuals in $A'^{\mathcal{I}}$ are *not* in $A^{\mathcal{I}}$, they cannot be in $B^{\mathcal{I}}$ either, and therefore must be in $B'^{\mathcal{I}}$, so $B' \supseteq A'$.

($\Leftarrow$) If A and B are positive literals and $A \supseteq B$, then obviously $A \supseteq B$. If A and B are negative literals (NOT A') and (NOT B') and $B' \supseteq A'$, the again, by the reasoning shown above, $A \supseteq B$. □

2.4 Canonical form

In order to simplify much of our presentation, it is useful to consider a canonical representation for descriptions,

Definition 6. *A description is in* canonical form *if it is a conjunction of descriptions which are not themselves conjunctions and whose immediate subdescriptions are in canonical form. In other words, the levels in a canonical description alternate between all-conjunction levels and levels with no conjunctions.*

The width *of a description in canonical form is the maximum number of conjuncts in any of its subdescriptions (including itself).*

For example, *male* has canonical form (AND *male*) and (AND *male* (AND *dentist* (SOME $friend$ (AND *dentist* $female$))) has canonical form (AND *male dentist* (SOME $friend$ (AND *dentist* $female$))).

Theorem 3. *Every finite-depth description C has an equivalent description C' in canonical form, and calculating it takes time $O(length(C))$.*

Proof. Let C be a description of finite depth. We prove by induction on C's depth that there is a description in canonical form which is equivalent to C, and that calculating it takes time $O(length(C))$.

For the base case of C with depth 0, C is an attribute description, with length 1. Then (AND C) is in canonical form, is equivalent to C and takes constant time to be computed.

For depth greater than 0, C is either (SOME r E) or (AND C_1 C_2 ... C_n). If C = (SOME r E), then $length(C) = length(E) + 1$ and by induction E has a canonical equivalent E' calculated in time $O(length(E))$. Then (AND (SOME r E')) is in canonical form, is equivalent to C and is computed in time $O(length(E) + 1) = O(length(C))$. If C = (AND C_1 C_2 ... C_n), by induction each C_i has an equivalent canonical description C_i' calculated in time $O(length(C_i))$, so C is equivalent to (AND C_1' C_2' ... C_n'). Because each C_i' is in canonical form, it can be written as (AND $C_{i,1}$ $C_{i,1}$ ... C_{i,n_i}) where $C_{i,j}$ is not a conjunction. Since AND is an associative operator, C is equivalent to (AND $C_{1,1}$ $C_{1,2}$ ... C_{1,n_1} $C_{2,1}$ $C_{2,2}$ ... C_{2,n_2} ... $C_{m,1}$ $C_{m,2}$ ... C_{m,n_m}), which is in canonical form and can be computed in $O(n) + \sum_{i=1}^{n} O(length(C_i)) = O(length(C))$ □

2.5 Satisfiability

The notion of *satisfiability* will be an important component of the subsumption algorithm presented later on.

Definition 7. *A description D is* satisfiable *if there is a model for D, and* unsatisfiable *otherwise.*

We are now interested in showing how the semantic notion of unsatisfiability reduces to the syntactic notion of *self-contradiction*.

Self-contradictory descriptions

Definition 8. *A description is* shallowly self-contradictory *if it is a conjunction and has conjuncts A and* (NOT B) *for any atoms A and B such that $B \supseteq A$. A description is* self-contradictory *if it contains a subdescription (possibly itself) which is shallowly self-contradictory.*

Theorem 4. *Deciding whether a description C is self-contradictory takes time $O(length(C)^2)$.*

Proof. We prove it by induction on the length of C.

If C has length 1, then it is an attribute description, so it has no conjunction subdescriptions and cannot be self-contradictory. This decision is made in constant time, thus $O(1)$ and $O(1^2)$.

If C has length greater than 1, it is either a SOME description or a conjunction. If it is a SOME description, by induction we can decide whether its only immediate subdescription is self-contradictory in time $O((length(C) - 1)^2)$, which is $O(length(C)^2)$. If it is a conjunction (AND C_1 C_2 ... C_n), each C_i with length l_i, by induction we can determine whether any of its conjuncts is self-contradictory in time $O(\sum_{i=1}^{n} length(C_i)^2)$, which is $O(length(C)^2)$. If none is, all that remains to be done is to verify that C itself is not shallowly self-contradictory by checking whethere there are two conjuncts A and (NOT

B) for any atoms A and B such that $B \supseteq A$ (SOME descriptions are not relevant for this purpose). Given that subsumption between atoms is a constant time operation, this takes time $O(n^2)$ and thus $O(length(C)^2)$. So the whole decision procedure for a conjunction description takes time $O(length(C)^2)$.

□

Self-contradictory descriptions are unsatisfiable

We want to eventually prove that canonical descriptions are self-contradictory if and only if they are unsatisfiable. The first part of that statement is provided by the theorem below.

Theorem 5. *If a description C is self-contradictory then it is unsatisfiable.*

Proof. Because a canonical form of a description always exists and has the same models as the original, we can assume without loss of generality that C is in canonical form.

We prove the theorem by induction on the number of conjuncts of C.

For the base case of 0 conjuncts, C cannot be self-contradictory, so the theorem is vacuously correct.

For C with more than 0 conjuncts, it is equivalent to $C' =$ (AND D D'), where D is the first conjunct and D' is the conjunction of the remaining conjuncts (possibly zero). Note that C' is not in canonical form but D' is.

C is self-contradictory, so either one of D and D' is self-contradictory or C is shallowly self-contradictory. If D (D') is self-contradictory, then by induction D (D') is unsatisfiable. Because any model satisfying C would also satisfy D (D'), C must also be unsatisfiable.

If C is shallowly self-contradictory, then it contains conjuncts A and (NOT B) for any atoms A and B such that $B \supseteq A$. Any model satisfying C would therefore have to satisfy A and (NOT B). However this is impossible because, since $B \supseteq A$, for any model $\mathcal{I}$, $i \in A^{\mathcal{I}} \Rightarrow i \in B^{\mathcal{I}} \Rightarrow i \notin (\text{NOT } B)^{\mathcal{I}}$. Therefore no model satifies C, thus it is unsatisfiable. □

Concept graphs

Concept graphs are useful constructs that can represent structured examples. They are defined over some set of nodes which are simply placeholders. Nodes are labeled by sets of literals, and edges by sets of role symbols.

Definition 9. *Given a description logic alphabet* $(Role, UnAttr, BinAttr, Val)$ *and a set of nodes* $Node$*, a* concept graph *is* (N, E, nl, el)*, where* $N \subseteq Node$*,* $E \subseteq Role$*,* nl *is a function mapping each node to a set of node labels (literals) and* el *is a function mapping each edge to a set of edge labels (role symbols).*

Induced models

We now introduce the notion of a model induced by a concept graph. Intuitively, this is a model made up of the concept graph's own components, reflecting the knowledge encoded in it. It is made a complete model by possibly creating new symbols necessary to instantiate attributes referred to by labels with uninstantiated attributes.

Definition 10. *A concept graph $G = (N, E, nl, el)$ defined on alphabet $(Role, UnAttr, BinAttr, Val)$ induces a model $(\Delta, \mathcal{I}(G))$ with domain $\Delta = (N, V)$, where $V = Val \cup W$, where W is a set of new symbols $W = \{v_{n,t} | n \in N, t \in nl(n)\}$ and the extensions to role and attribute symbols are determined according to their labeling edges and nodes as follows:*

- $n \in o^{\mathcal{I}(G)} \Leftrightarrow o \in nl(n) \wedge o \in UnAttr$
- $(n, v_{n,t}) \in t^{\mathcal{I}(G)} \Leftrightarrow t \in nl(n) \wedge t \in BinAttr$
- $(n, v) \in t^{\mathcal{I}(G)} \Leftrightarrow t(v) \in nl(n) \wedge t \in BinAttr$
- $(n_1, n_2) \in r^{\mathcal{I}(G)} \Leftrightarrow r \in el(n_1, n_2)$

Note that this model induction makes a *closed-world assumption*, since everything not explicitly defined as true is assumed to be false. This is why negative literal labels are ignored; they are always automatically satisfied.

The model induced by a concept graph is in a sense analogous to the least Herbrand model of a theory (as defined in, for example, [7]).

Definition 11. *A* concept tree *is a concept graph that is a tree.*

Unsatisfiable descriptions have self-contradictory canonical forms

We now present the counterpart of Theorem 5.

Theorem 6. *Every description in canonical form that is not self-contradictory has a model induced by a concept tree.*

Proof. Let C be a non-self-contradictory description in canonical form (AND $A_1 \ldots A_k$(NOT B_1)$\ldots$(NOT B_l)(SOME E_1)(SOME E_2)$\ldots$(SOME E_m)) where A_i, B_j are atoms for $i = 1, \ldots, k, j = 1, \ldots, l$.

Let us prove the existence of a concept tree model for C by induction on the depth of C. If C has depth 1, $m = 0$ and C has only attribute description conjuncts. In this case, let G be the concept tree with a single node with labels $A_1 \ldots A_k$ inducing a model $\mathcal{I}(G)$. We can see that $\mathcal{I}(G)$ is a model of C, because it satisfies all of its conjuncts: the A_i's, straightforwardly from the definition of $\mathcal{I}(G)$, and the (NOT B_j)'s from the fact that C is non-self-contradictory, so every B_j does not subsume any A_i (which either means that B is defined with an attribute symbol not used by any A_i, and therefore with an empty extension, or that is it an instantiated attribute of form $t(v)$ for

some v not present in any A_i defined on t, and therefore not present in the extension of t).

If C has depth greater than 1, C has, besides its attribute description conjuncts, conjuncts of form (SOME E_i). We can then form a concept tree G, just like in the base case, from the attribute description conjuncts alone, that we have already been shown to satisfy all of them. By induction, each (SOME E_i) has a model $\mathcal{I}_i$ induced by concept tree e_i. We add these trees to G by adding an edge from the root of G to the root of e_i labeled by E_i, for each i, forming a new tree G'. The model $\mathcal{I}(G')$ induced by G' does not change the extensions of attributes for the extension of the root of G, so the attribute description conjuncts of C continue to be satified. Moreover, the (SOME E_i) conjuncts are also now satisfied, so $\mathcal{I}(G')$ is a model of C induced by a concept tree. □

Corollary 1. *All canonical forms of an unsatisfiable description are self-contradictory.*

Proof. Let C be an unsatisfiable description and C' be any of its canonical forms. If C' were not self-contradictory, by the previous theorem it would have a model, but this model would also be a model of C, which is unsatisfiable. Therefore C' must be self-contradictory. □

2.6 Subsumption algorithm

We now proceed to present an algorithm for deciding subsumption between two descriptions in $\mathcal{FDL}$. This is an algorithm in the class of *structural subsumption algorithms* commonly refered to in the DL literature [1]. It will be used as the basis for feature extraction in the next section.

Algorithm 1 (Regular Subsumption Algorithm). Given two descriptions C and D, where C has finite depth and D is in canonical form (AND D_1 D_2 $\ldots$ D_n), decide whether $C \supseteq D$.

If D is self-contradictory, return *true*.

Otherwise, we examine the possible cases for C:

1. C = (AND C_1 C_2 $\ldots$ C_n):
   Then return *true* if and only if $\forall i$ $C_i \supseteq D$ (by recursion).
2. C is a literal A:
   Then return *true* if and only if there is a D_i which is an literal description and $A \supseteq D_i$ (according to Theorem 2).
3. C = (SOME r E):
   Then return *true* if and only if $\exists i$ *s.t.* D_i = (SOME r F) and $E \supseteq F$ (by recursion).

Theorem 7. *Algorithm 1 for descriptions C and D always stops, is sound, complete and has time complexity $O(length(C)width(D) + length(D)^2)$.*

Proof. Let us first show that the algorithm stops. If D is unsatisfiable, the algorithm returns *true* and stops. Otherwise, we show that the algorithm stops by induction on the depth of C. If the depth of C is 0, it must be an attribute description. Therefore one of the second or third cases of the algorithm is used; the algorithm decides subsumption between two atoms according to Theorem 1 in constant time and stops. If the depth of C is greater than 0, one of the first or third cases will be used. In both cases, the algorithm will be recursively called for a description of smaller depth, which by induction will stop. After that, some other constant time operation (depending on the case) will be performed, and the algorithm will stop.

The algorithm's time, *excluding* the time for checking D's satisfiability, is $O(lw)$ for l the length of C and w the width of D. We prove this by induction on the depth of C. If C has depth 0, C is an attribute description and its attribute symbol will be compared to each conjunct of D in constant time. Thus its time will be $O(n)$. Since $n \leq w$, the time is also $O(w)$ and thus $O(lw)$. If C has depth greater than 0, it is either (AND C_1 C_2 $\ldots C_m$) or (SOME r E). If C is (AND C_1 C_2 $\ldots C_m$), by induction its time will be $\sum_{i=1}^{m} O(l_i w) = O((\sum_{i=1}^{m} l_i)w) = O(lw)$, where $l_1, l_2, \ldots, l_m$ are the lengths of $C_1, C_2, \ldots, C_m$ respectively. If C is (SOME r E), by induction its time will be $O((l-1)w) \leq O(lw)$.

Since checking D's satisfiability takes time $O(length(D)^2)$, the total time is $O(length(C)width(D) + length(D)^2)$.

We prove that the algorithm is sound and complete by analysing cases.

If D is unsatisfiable, its extension will be empty for every model and all its individuals will vacuously be in any other description's extension, including C's. So the algorithm is correct in returning *true*.

For the case in which C is a conjunction (AND C_1 C_2 $\ldots C_n$), if the algorithm returns *true*, every C_i subsumes D, which means that all individuals in $D^{\mathcal{I}}$ are in $C_i^{\mathcal{I}}$. By the definition of AND, this means that all individuals in $D^{\mathcal{I}}$ are in $C^{\mathcal{I}}$, so $C \supseteq D$ and the algorithm is correct. If the algorithm returns $false$, some C_i does not subsume D. Thus there is an interpretation $\mathcal{I}$ with an individual i in $D^{\mathcal{I}}$ but not in $C_i^{\mathcal{I}}$ and consequently not in $C^{\mathcal{I}}$ as well. It follows that $C \not\supseteq D$, so the algorithm is correct.

For the case in which C is (SOME r E), if the algorithm returns *true*, then some D_i conjunct of D is equal to (SOME r F) with $E \supseteq F$. For each individual i in $D^{\mathcal{I}}$, i is also in $D_i^{\mathcal{I}}$, so $(i, j) \in r^{\mathcal{I}}$ for some j in the extension of F. Therefore j is also in the extension of E, which implies that i is in the extension of (SOME r E), that is, C, so the algorithm is correct in indicating that $C \supseteq D$. If the algorithm returns $false$, there is no conjunct D_i equal to (SOME r F) with $E \supseteq F$. Then it is the case that either there is no conjunct D_i equal to (SOME r F) for any F, or there is such a conjunct with $E \not\supseteq F$. If there is no conjunct D_i = (SOME r F) for any F, there is a model for D induced by a concept tree whose root individual is not in the extension of r, so this model cannot satisfy C, and the algorithm is correct in returning $false$. If there is a conjunct D_i = (SOME r F) but $E \not\supseteq F$, let $\mathcal{I}$ be a concept

tree-induced model for D and $\mathcal{I}'$ be any model for which $E^{\mathcal{I}'} \not\supseteq F^{\mathcal{I}'}$. Now we can form a new model $\mathcal{I}''$ by merging the domains of $\mathcal{I}$ and $\mathcal{I}'$ (renaming objects if necessary to avoid collisions) and replacing the individual in $F^{\mathcal{I}}$ for some individual k in $F^{\mathcal{I}'}$ that does not belong to $E\mathcal{I}'$. Because $\mathcal{I}$ is a concept tree-induced model, $\mathcal{I}''$ still satisfies D as much as $\mathcal{I}$ and does not satisfy C, so the algorithm is correct in returning $false$.

For the case in which C is a literal A based on an atom A' (that is, C is either A' or (NOT A')) based on an attribute symbol c, the algorithm returns *true* only if there is D_i a literal description subsumed by C. If there is such D_i, then, for any model $\mathcal{I}$ of D, all individuals in $D\mathcal{I}$ must be in $D_i\mathcal{I}$ and therefore in $C\mathcal{I}$, so the algorithm is correct. If there is no such D_i, this is either because no D_i is based on attribute symbol c or every D_i based on c if such that $C \not\supseteq D_i$. For the first case, we can take a concept tree-induced model $\mathcal{I}$ of D and make a new model $\mathcal{I}'$ by fixing $c^{\mathcal{I}'}$ so that $C^{\mathcal{I}'}$ is empty. Since no D_i involves c, $D^{\mathcal{I}'} = D^{\mathcal{I}}$ and $\mathcal{I}'$ is a model of D but not of C, so the algorithm is correct in returning $false$. For the second case, we again take a concept tree-induced model $\mathcal{I}$ of D and fix the extension of symbol c into a new model $\mathcal{I}'$ so that $C^{\mathcal{I}'}$ is empty but not $D^{\mathcal{I}'}$. This is possible because no D_i based on c is subsumed by C, so there is an extension for c that satisfies all such D_i without satisfying C. Then $\mathcal{I}'$ is a model of D but not of C, which means that $C \not\supseteq D$ and the algorithm is correct in returning $false$. □

Note that the algorithm halting does not depend on the subsumee's depth, which is therefore possibly infinite. This will be useful to us in the next section.

3 Feature Description Logic

We call *Feature Description Logic* the framework where one uses description logics for the purpose of describing features of interest to be extracted from structured examples (that is, examples represented by sets of objects with attributes and relations between themselves).

In this framework, an example can be structured and arbitrarily complex, being represented by a concept graph whose node labels are fully instantiated literals (see Fig. 1).

Definition 12. *An* example graph *is a concept graph whose node labels are fully instantiated literals.*

Concept graphs have extensions under an interpretation, which we define now.

Definition 13. *Given a model $\mathcal{I}$ with domain (X, V) and a concept graph $G = (N, E, nl, el)$, the extension $G^{\mathcal{I}}$ of G is defined as*

$$\{I \subseteq X | \ \exists\ node \in M(I,N),\ \forall i \in I,$$
$$\forall A \in Literal,\ i \in A^{\mathcal{I}} \Leftrightarrow A \in nl(node(i)) \wedge$$
$$\forall r \in Role,\ \forall j \in I,\ (i,j) \in r^{\mathcal{I}} \Leftrightarrow r \in el(node(i), node(j))\}$$

where $M(I,N)$ is the set of one-to-one mappings from individuals set I and node set N.

A concept graph has a model $\mathcal{I}$ *if* $|G^{\mathcal{I}}| > 0$*, and is* satisfiable *if it has some model.*

The intuitive interpretation of $G^{\mathcal{I}}$ is that it is the set of all sets of individuals that can be mapped into nodes such that each individual satisfies the constraints imposed by the concept graph on its corresponding node.

Feature extraction can be formalized as a function mapping an example graph to a feature set, parameterized by a given set of *feature type descriptions*. An example graph is mapped into the set of feature type descriptions that, for every model of the graph, have a non-empty extension in that model.

Definition 14. *Let $\mathcal{D}$ be a set of descriptions in $\mathcal{FDL}$. A* feature vector indexed by $\mathcal{D}$ *is the characteristic function of a set of descriptions (a function from feature descriptions in $\mathcal{D}$ to $\{0,1\}$). Given a set of feature type descriptions $\mathcal{D}$, we define the* feature generating function $F_{\mathcal{D}}$ *to be a function which maps examples to feature vectors indexed on $\mathcal{D}$ as follows: for any example graph $G = (N, E, nl, el)$ and feature type description $C \in \mathcal{D}$,*

$$(F_{\mathcal{D}}(G))_C = 1 \Leftrightarrow (\forall \mathcal{I}\ \ |G^{\mathcal{I}}| > 0 \Rightarrow |C^{\mathcal{I}}| > 0)$$

(in which case C is said to be active *in G), and 0 otherwise*[3]*.*

We show that determining whether C is active in G is equivalent to determining $C \supseteq descr_G(n)$ for some node n, where $descr_G$ is a function on nodes, described below, generating the most specific description such that $i \in descr_G(n)^{\mathcal{I}(G)}$, for any individual i associated to n in one of the sets in the graph's extension. This is similar to the approaches taken by [3] and [6], in the context of learning descriptions from most specific descriptions rather than from concrete examples.

For the remaining of this section, let $G = (N, E, nl, el)$ be an example graph and n a node in N.

Definition 15. *Let $\{(n, m_1), (n, m_2), \ldots, (n, m_k)\}$ be the outgoing edges of n and $l_1, l_2, \ldots, l_j$ its labels. We define $descr_G(n)$ recursively to be the description*

(AND
(SOME $r_{m_1,1}$ $descr_G(m_1)$)

[3] We do not differentiate between distinct instantiations of the same feature type (when the feature type used not fully instantiated attributes), as done in the variation presented in [4].

(SOME $r_{m_1,2}$ $descr_G(m_1)$)
...
(SOME $r_{m_1,|el(n,m_1)|}$ $descr_G(m_2)$)
(SOME $r_{m_2,1}$ $descr_G(m_2)$)
...
(SOME $r_{m_2,|el(n,m_2)|}$ $descr_G(m_2)$)
...
(SOME $r_{m_k,1}$ $descr_G(m_k)$)
...
(SOME $r_{m_k,|el(n,m_k)|}$ $descr_G(m_k)$)
l_1 l_2 ... l_j)

where the labels for edge (n, m_j) *are indexed* $r_{m_j,1}, r_{m_j,2}, \ldots, r_{m_j,|el(n,m_j)|}$, *for* $j = 1, 2, \ldots k$.

Just like descriptions, concept graphs will be satisfiable only if they do not have self-contradictory labels in some node. This can be formalized through $descr_G$:

Theorem 8. *G has a model if and only if for every $n \in N$, $descr_G(n)$ is not self-contradictory.*

Proof. ($\Rightarrow$) If G has a model $\mathcal{I}$, there is a set of individuals I and a mapping *node* such that $i \in descr_G(n)^{\mathcal{I}}$ for every node $n \in N$ with i such that $n = node(i)$, according to the definition of a concept graph's extension, which poses the same constraints on i that $descr_g(n)$ poses on its individuals. Therefore $descr_G(n)$ for any node n in G is satisfiable and not self-contradictory.

($\Leftarrow$) Let $\mathcal{I}$ be the model induced by G. We need to show that this is a model *for* G as well, that is, that $|G^{\mathcal{I}}| > 0$. In a model induced by a concept graph, all positive literal labels and edge labels are immediately satisfied. It only remains to be shown that negative literal labels are also satisfied. Because no description $descr_G(n)$ is self-contradictory, no node in G has labels (NOT B) and A such that $B \supseteq A$. By the same argument presented in Theorem 6's proof, the negative literal labels will also be satisfied. So $\mathcal{I}$ is also a model for G. □

Theorem 9. *Let C be a description and $G = (N, E, nl, el)$ be a concept graph which is not self-contradictory. Then C is active in $G \Leftrightarrow \exists n \in N \quad C \supseteq descr_G(n)$.*

Proof. ($\Rightarrow$) Let us prove this by negation. Suppose that there is no $n \in N$ such that $C \supseteq descr_G(n)$. So, for each node $n \in N$, there is a constraint c_n in C that is not in $descr_G(n)$. Let $\mathcal{I}$ be a model such that $|G^{\mathcal{I}}| > 0$. Then, according to the definition of $G^{\mathcal{I}}$, there is a set of individuals I mapped to nodes through a mapping *node*. Let $\mathcal{I}'$ be a new model such that c_n is false for every individual j such that $node(j) = n$. Now I is still in $G^{\mathcal{I}'}$ because c_n is not in $descr_G(n)$ and therefore is not required in order to I be in $G^{\mathcal{I}'}$

(see definition of the extension of a concept graph to see that). But now $C^{\mathcal{I}'}$ must be empty, for no individual in $\mathcal{I}'$ contains all contraints expressed by C. Therefore C cannot be active in G due to model $\mathcal{I}'$, violating the hypothesis. So it must be that if C is active in G, $C \supseteq descr_G(n)$ for some node $n \in N$.

($\Leftarrow$) We assume $C \supseteq descr_G(n)$ for some node $n \in N$. Let $\mathcal{I}$ be any model such that $|G^{\mathcal{I}}| > 0$. Such a model exists because G is not self-contradictory. Then there is a set I of individuals with a mapping *node* to nodes satisfying the conditions in the definition of extensions of concept graphs, which are the same conditions posed by the $descr_G$ descriptions of those respective nodes. Therefore, if j is the individual such that $n = node(j)$, it must be that $j \in descr_G(n)^{\mathcal{I}}$, and because $C \supseteq descr_G(n)$, j must also be in $C^{\mathcal{I}}$. So for every model $\mathcal{I}$ such that $|G^{\mathcal{I}}| > 0$, $|C^{\mathcal{I}}| > 0$, thus C is active in G. □

The previous theorem ensures that, in order to decide whether a description is active in a concept graph G, we can reduce the problem to subsumption by deciding $C \supseteq descr_G(n)$ for every $n \in N$. Note that $descr_G(n)$ might have possibly infinite depth, in case G has cycles. However, for our purposes this is not a problem, since this description will be used as the subsumee candidate in the subsumption algorithm, and since the number of description levels examined by the algorithm does not exceed the subsumer's depth, we only need to generate $descr_G(n)$ up to that depth.

4 Functional subsumption

In a real application, certain properties can be expressed in more than one way. For example, a certain type of action we are interested in may have several verbs which describe it, or we may want to recognize obvious typos or variations in writing as instances of the intended word. In a more concrete situation, suppose we are examining descriptions involving actions and the arguments of these actions, which are related through role *arg*. If we have the feature description

(AND *buy* (SOME *arg car*))

and we know, in some yet unspecified way, that *purchase*, *acquire* and *buuy* are possible replacements for *buy*, and *argument* is a possible replacements for *arg*, we would like it to subsume all of the descriptions below:

(AND *buy* (SOME *arg* (AND *car toyota*))
(AND *buy* (SOME *argument* (AND *car toyota*))
(AND *purchase* (SOME *arg* (AND *car honda*))
(AND *acquire* (SOME *arg* (AND *car buick*))
(AND *acquire* (SOME *argument* (AND *car buick*))
(AND *buuy* (SOME *arg* (AND *car ford*))

Such a mechanism would solve the problem of accounting for varying forms of attributes and roles.

If we want to do something like this with plain FDLs as explained in the previous sections, we would need to create many distinct feature type descriptions (one for each replacement combination for *buy* and arg), causing much redundancy in their specification at best and incompleteness at worst (for example, we may not be able to generate all possible typos of a word, assuming we want them to be considered able to replace the original word).

The problem could be alleviated by making use of a TBox [1], the specification of a terminology declaring equivalence between certain concepts, but this would be too strong since it forces us to establish equivalence classes, while we may be interested to say that a attribute a can pass as b and b as c, without a passing as c. Even a terminology with inclusion axioms, that is, axioms declaring that one attribute is subsumed by another, would also impose some kind of ordering on attributes, while we are interested in specifying subsumption between attributes on independently pair-by-pair basis.

What we want instead is to be able to utilize a binary replaceability function on attribute and roles that defines which attributes are replaceable by which other attributes, and which roles by which other roles. A user could then specify this function, reflecting some piece of background knowledge specific to the task. We call this *functional subsumption*.

A key advantage in using a function is that it does not need to be explicitly defined, allowing for its specification to be done by any means available and convenient, and that it can depend on the context in ways not easily representable otherwise in our framework. This also allows for using functions that are outcomes of learning themselves.

4.1 Functional subsumption semantics

We now define more clearly what functional subsumption means.

Definition 16. *A* functional subsumption function *is a function* $f : (Atom \cup Role) \times (Atom \cup Role) \to \{0, 1\}$.

The semantics behind functional subsumption is that, if a description C contains an attribute A, it can be regarded as containing the conjunctions of all other attributes that can replace A as well. The rationale behind this is that, if B can replace A, then everytime A is present, B is also present for subsumption purposes. The analogous case holds for roles.

Definition 17. *Let* f *be a functional subsumption function, and* C *and* D *be two descriptions in* $\mathcal{FDL}$. C functionally subsumes D with respect to f, *denoted by* $C \sqsupseteq_f D$, *if* $C \sqsupseteq D_f$, *where* D_f *is* D *after two operations in sequence: first, each atom* A *is replaced by* (AND A_1 A_2 ...), *for* A_1, A_2, ... *being all atoms such that* $f(A_i, A) = 1$, *and each literal* (NOT B) *is replaced by* (AND (NOT B_1) (NOT B_2) ...) *for* B_1, B_2, ... *being all atoms such that* $f(B, B_i) = 1$ *and, second, each description* (SOME r E) *is replaced by* (AND (SOME r_1 E) (SOME r_2 E) ...), *where* r_1, r_2, ..., *are all roles such that* $f(r_i, r) = 1$.

4.2 Functional Subsumption Algorithm

The semantics of functional subsumption presented above has the advantage of being clear but involves some issues. In particular, obtaining D_f for a description D may take a long time depending on the number of replaceable elements there may be according to the function, or be even impossible when this number is infinite.

There is, however, the alternative of directly plugging the replaceability function into the Regular Subsumption Algorithm, generalizing the atom comparison on which it is based. Instead of accepting only cases in which the attribute from the left is more general than the attribute from the right, or the role in the left is equal to the role in the right, we now accept the cases in which the attribute or role in the left can replace the attribute or role in the right, according to the function given.

Algorithm 2 (Functional Subsumption Algorithm).
Given descriptions C and D and a functional subsumption function f, the *functional subsumption algorithm* is as follows:

- C is (AND C_1 C_2 ... C_n):
  Then return *true* if and only if, for each C_j, $C_j \sqsupseteq_f D$ (decided with a recursive call).
- C is (SOME r E):
  Then return *true* if and only if there is a D_i = (SOME s F) such that $f(r, s) = 1$ and $E \sqsupseteq_f F$ (decided with a recursive call).
- C is an atom A:
  Then return *true* if and only if there is a $D_i \in Atom$ such that $f(A, D_i) = 1$.
- C = (NOT A):
  Then return *true* if and only if $\exists i$ D_i =(NOT B) such that $f(B, A) = 1$.

Theorem 10. *Given two descriptions C and D and a functional subsumption function f, with D in canonical form, the Functional Subsumption Algorithm applied to C and D returns true if and only if C functionally subsumes D wrt f, and has time complexity $O(length(C)width(D) + length(D)^2)$.*

Proof. We prove this by recursion on the depth of C. For the base case, C is a literal. If it is a positive literal, the algorithm returns *true* if and only if $f(C, D_i) = 1$ for some D_i in D, which is equivalent to say that C is in the corresponding description in D_f, which happens if and only if $C \sqsupseteq D_f$, the same as $C \sqsupseteq_f D$. The analogous reasoning works for negative literals. If the depth of C is greater than 0, it is either a conjunction or a SOME description. If it is a conjunction, the algorithm returns *true* if and only if $C_j \sqsupseteq_f D$ for each C_j in C. By induction, this is equivalent to $C_j \sqsupseteq D_f$, which is equivalent to $C \sqsupseteq D_f$, the same as $C \sqsupseteq_f D$. If C is (SOME r E), the algorithm returns *true* if and only if there is D_i = (SOME s F) such that $f(r, s) = 1$ and

$E \supseteq_f D$. By induction, this is equivalent to $E \supseteq F_f$. Also, $f(r,s) = 1$, if and only if D_f contains a conjunct (SOME r F_f). This fact and $E \supseteq F_f$ are equivalent to $C \supseteq D_f$, which is equivalent to $C \supseteq_f D$.

As for complexity, the Functional Subsumption Algorithm replaces the constant checking of subsumption between literals and the checking of roles in the Regular Subsumption Algorithm by evaluating f on atoms and role symbols, which does not depend on the size of C and D. Therefore its complexity is the same, $O(length(C)width(D) + length(D)^2)$. □

What the above theorem assures us of is that the Functional Subsumption Algorithm correctly implements the Functional Subsumption Semantics described above. This is a very practical algorithm since the alternative (obtaining D_f) may be impossible (if there are infinite atoms able to replace a certain atom) or simply take a long time to run.

5 Conclusion

Machine Learning is about abstracting domain elements into a representation that captures the important/relevant properties of it, and abstracts away those that are not. In many AI problems, such as natural language processing and visual inference, there is a need to learn predicates over complex, structurally and relationally, domain elements. Our work focuses on the problem of mapping these domain elements into representations that are useful as inputs to a learning algorithm – best viewed as an inference process.

We suggest that this process is an integral part of the learning process and that, sometimes, much of the complexity of learning is in fact in this stage.

This paper integrates ideas and tools from learning, knowledge representation and reasoning, to develop a new FDL language, along with an efficient *functional subsumption* algorithm, which supports abstraction over attributes and roles between substructures of domain elements – thus addressing a situation of great practical relevance, in which the same features are extracted despite some variability in the data.

In addition to providing a way to incorporate background knowledge into the process of transforming domain elements into an input to a learning algorithm, the formal approach allows one to analyze this stage of the learning process. It opens up several future directions, including extending functional subsumption to *structural subsumption* – allowing also restricted structural transformations between FDL descriptions, extending it to probabilistic functional subsumption, and developing graph kernels [5] that correspond to functional subsumption.

6 Acknowledgments

We would like to thank Vasin Punyakanok for carefully reading our drafts and providing insightful comments. This research is supported by NSF grants ITR-IIS-0085836, ITR-IIS-0085980 and IIS-9984168 and an ONR MURI Award.

References

[1] F. Baader. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, pages 47–100. Cambridge University Press, Cambridge, 2002.

[2] F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In O. Herzog and A. Günter, editors, *Proceedings of the 22nd Annual German Conference on Artificial Intelligence, KI-98*, volume 1504, pages 129–140, Bremen, Germany, 1998. Springer–Verlag.

[3] W. W. Cohen and H. Hirsh. The learnability of description logics with equality constraints. *Machine Learning*, 17:169–199, 1994.

[4] C. Cumby and D. Roth. Learning with feature description logics. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, pages 32–47. Springer-Verlag, 2002.

[5] C. Cumby and D. Roth. On kernel methods for relational learning. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning*, pages 107–114. AAAI, 2003.

[6] M. Frazier and L. Pitt. CLASSIC learning. In *Computational Learning Theory*, pages 23–34, 1994.

[7] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.