A Description Classifier for the Predicate Calculus

Robert M. MacGregor

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
macgregor@isi.edu

Abstract

A description classifier organizes concepts and relations into a taxonomy based on the results of subsumption computations applied to pairs of relation definitions. Until now, description classifiers have only been designed to operate over definitions phrased in highly restricted subsets of the predicate calculus. This paper describes a classifier able to reason with definitions phrased in the full first order predicate calculus, extended with sets, cardinality, equality, scalar inequalities, and predicate variables. The performance of the new classifier is comparable to that of existing description classifiers. Our classifier introduces two new techniques, dual representations and auto-Socratic elaboration, that may be expected to improve the performance of existing description classifiers.

Introduction

A *description* is an expression in a formal language that defines a set of instances or tuples. A *description logic* (also called a terminological logic) consists of a syntax for constructing descriptions and a semantics that defines the meaning of each description. Description logics [MacGregor 90].provide the foundation for a number of modern knowledge representation systems, including Loom[MacGregor 91], BACK[Peltason 91], CLASSIC[Borgida et al 89], KREP[Mays et al 91], and KRIS[Baader&Hollunder 91]. Each of these systems includes a specialized reasoner called a *description classifier* that computes subsumption relationships between descriptions, and organizes them into one or several taxonomies. Subsumption computations play a role in these systems analogous to the match or unification operations performed in other classes of deductive systems.

Description logic systems as a class implement a style of deductive inference that is deeper than standard backchaining, and that is much more efficient than theorem prover-based deduction. A hallmark of description logics is that they severely limit the expressive power of their description languages. We believe that the absence of full expressivity is one of the factors that is preventing description classifiers from becoming a standard component in knowledge base management systems [Doyle&Patil 91]. Accordingly, we have developed a new classifier that accepts description expressions phrased using the full predicate calculus, extended with sets, cardinality, equality, scalar inequalities, and predicate variables. The description syntax is uniform for predicates of arbitrary arity, and recursive definitions are supported.

We have found that architectural principles developed for description logic classifiers can be transferred into a classifier that reasons with predicate calculus expressions. This paper begins by describing the internal format and subsumption algorithm used in the predicate calculus (PC) classifier. We next discuss the normal form transformations used in this classifier. The strategy for normalization incorporates two innovations, dual representations and auto-Socratic elaboration, that increase both the performance and the flexibility of the classifier. Finally, we present results indicating that the new classifier has performance comparable to that of existing classifiers.

Descriptions

A *relation description* specifies an intensional definition for a relation. It has the following components:

- a name (optional);
- a list of domain variables $< dv_1, \ldots ,dv_k >$, where k is the arity of the relation;
- a definition—an open sentence in the prefix predicate calculus whose free variables are a subset of the domain variables.
- a partial indicator (true or false)— if true, it indicates that the predicate represented by the relation definition is a necessary but not sufficient test for membership in the relation.

If a relation description is partial, then the relation is said to be primitive. If R is a non-primitive relation with arity one, then its extension is defined as the set $\{dv_1 \mid defn_R\}$, where dv_1 is the domain variable and $defn_R$ is the definition in the description of R. If R is a non-primitive relation with arity k greater than one, then its extension is defined as the set of tuples $\{< dv_1, \ldots ,dv_k > \mid defn_R\}$, where $dv_1, \ldots ,dv_k$ are the domain variables and $defn_R$ is the definition in the description of R. If R is primitive, then its extension is a subset of the set associated with its definition.

Relation descriptions are introduced by the **defrelation** operator, with the syntax

```
(defrelation <name> (<domain variables>)
             [:def | :iff-def] <definition>)
```

The keyword **:def** indicates that the definition is partial, while the keyword **:iff-def** indicates that it is not. For example

```
(defrelation Person (?p) :def (Mammal ?p))
```

defines a relation Person to be a primitive subrelation of the relation Mammal.[1] The description

```
(defrelation daughter (?p ?d)
   :iff-def (and (child ?p ?d)
                 (Female ?d))))
```

defines the relation **daughter** to be a non-primitive binary subrelation of the relation **child**.

A simple sentence is predication of the form **(P** t_1 ... t_k**)** where t_1 ... t_k are terms, and **P** is either the name of a k-ary relation or a term that evaluates to a k-ary relation. Complex sentences are constructed from simple sentences using the operators **and**, **or**, **not**, and **implies**, and the quantifiers **forsome** and **forall**. A term is either a constant, a variable, a set expression, or a form **(F** t_1 ... t_j**)** where **F** is a single-valued relation of arity (**j+1**) (i.e., **F** is a function). A variable is a string of characters prefixed by "**?**". A set expression is a term of the form **(setof (<variables>) <definition>)** that defines an unnamed relation with domain variables **<variables>** and definition **<definition>**. The function **the-relation** takes as arguments a name (a string of characters not prefixed by "**?**") and a positive integer indicating the arity, and returns the relation with that name and arity (relations of different arity may share the same name).

Subsumption

The primary deductive task of a description classifier is the computation of "subsumption" relationships. We say that a relation A *subsumes* a relation B if, based upon their respective definitions, the extension of A contains the extension of B. A description classifier organizes descriptions into a hierarchy, with description A placed above description B if A's relation subsumes B's relation. The inverse relation to subsumes is called *specializes*. B specializes A if A subsumes B.

Consider the following pair of descriptions. **At-Least-One-Son** defines the set of all persons that have at least one son:

```
(defrelation At-Least-One-Son (?p)
  :iff-def (and
    (Person ?p)
    (>= (cardinality
           (setof (?c) (son ?p ?c)))
        1))
```

More-Sons-Than-Daughters defines the set of persons that have more sons than daughters:

```
(defrelation More-Sons-Than-Daughters (?pp)
  :iff-def (and
    (Person ?pp)
    (> (cardinality
          (setof (?b) (son ?pp ?b)))
       (cardinality
          (setof (?g) (daughter ?pp ?g))))))
```

The PC classifier can prove that **At-Least-One-Son** subsumes **More-Sons-Than-Daughters**, i.e. it will classify **More-Sons-Than-Daughters** below **At-Least-One-Son**. The remainder of this paper describes the proof strategy that it uses to find such subsumption relationships.

The majority of description classifiers currently implemented, including the PC classifier, employ a *structural* subsumption test.[2] Roughly speaking, to prove that a description A subsumes a description B, a structural subsumption prover attempts to demonstrate that for every structural "component" in (the internal representation for) the definition of A there exists a "corresponding component" in (the internal representation for) the definition of B. An appealing feature of a classification strategy that uses a structural test is that much of the inferencing occurs in a "normalization" phase that precedes the actual test. If a description is repeatedly tested for subsumption against other descriptions (the usual situation in a classifier) the cost of normalization is amortized across all tests, thereby lowering the average cost of each subsumption test.

Most classifiers adopt frame-like representations for their internal representation of descriptions, and their subsumption tests operate by comparing the structure between a pair of such frames. The PC classifier parses definitions phrased in the prefix predicate calculus into graph-based structures, and all subsequent reasoning involves operations on these graphs. Each of our graphs represents a set expression, consisting of a list of domain variables and a set membership test (the definition). The root node of such a graph is equated with a **setof** expression, additional nodes represent each of the domain variables, and the remaining edges and nodes represent the membership test. A node can represent a variable, a constant, or another set expression. A predicate applied to a list of terms is represented by a (hyper) edge connecting the nodes corresponding to the terms in the list, together with a pointer to the node that corresponds to the predicate. Nodes representing skolem variables are introduced to represent function terms. Variables other than domain

[1] A sortal relation such as Person would ordinarily be introduced as a concept, using the **defconcept** operator, rather than being defined as a unary relation. The algorithm is the same for classifying concepts and for classifying unary relations. For simplicity, we avoid distinguishing between concepts and unary relations in this paper.

[2] The KRIS classifier [Baader&Hollunder 91] is the notable exception.

variables are assumed to be existentially quantified. As we shall see below, our graph representation eliminates the use of (or need for) universal quantifiers. The notation includes explicit representations for disjunction, negation, and enumerated sets—these constructs lie outside of the scope of the present discussion.

Because edges in a graph can point to (root nodes of) other graphs, our graph structures form a network. Each edge in the network "belongs to" exactly one graph—the graph corresponding to the innermost **`setof`** expression that contains the predication that defines that edge. Each graph is defined to consist of a set of edges plus the set of all nodes referenced by those edges. A node can therefore "belong to" many different graphs. The job of a subsumption test that is comparing graphs A and B is to find a substitution that maps nodes and edges belonging to graph A to corresponding nodes and edges belonging to graph B.

The parser that converts predicate calculus descriptions into graphs applies a few simple transformations in the process, including skolemization of set and function expressions, and conversion of material implications into subset relations (this is illustrated later in this section). In the remainder of this paper, we shall use the term "graph" to refer to a **`setof`** expression that has undergone these transformations, and we will use graph terminology (e.g., nodes, edges, paths) when referring to structural components and features within our set expressions.

The **`At-Least-One-Son`** relation defined above is associated with a graph representing the following set

```
 (setof (?p)
    (and (Person ?p)
         (>= (cardinality
                (setof (?c) (son ?p ?c)))
         1)
```

The addition of variables to represent the nested **`setof`** expression and the Skolemized cardinality function produces the following equivalent set expression

```
 (setof (?p) (exists (?s1 ?card1)        [1]
    (and (Person ?p)
         (= ?s1 (setof (?c) (son ?p ?c)))
         (cardinality ?s1 ?card1)
         (>= ?card1 1))))
```

We can now illustrate how a structural subsumption algorithm finds a subsumption relationship between **`At-Least-One-Son`** and **`More-Sons-Than-Daughters`**. Here is the "graph" for **`More-Sons-Than-Daughters`**:

```
 (setof (?pp)                             [2]
    (exists (?s2 ?s3 ?card2 ?card3) (and
       (Person ?pp)
       (= ?s2 (setof (?b) (son ?pp ?b)))
       (= ?s3 (setof (?g) (daughter ?pp ?g)))
       (cardinality ?s2 ?card2)
       (cardinality ?s3 ?card3)
       (> ?card2 ?card3))))
```

To prove that **`At-Least-One-Son`** subsumes **`More-Sons-Than-Daughters`**, we look for a substitution mapping nodes in [1] to nodes in [2] such that for each edge in the graph [1] there is a corresponding edge in the graph [2]. The correct substitution σ is

?p $\Rightarrow_\sigma$?pp; ?s1 $\Rightarrow_\sigma$?s2; ?card1 $\Rightarrow_\sigma$?card2

except that there is a problem—no edge in graph [2] corresponds to the edge **`(>= ?card1 1)`** in graph [1]. However, a constraint representing the missing edge **`(>= ?card2 1)`** is logically derivable from the existing constraints/edges present in graph [2]—the addition of this missing edge would result in a set expression logically equivalent to the expression [2]. Using a process we call "elaboration" (explained in Section 5), the PC classifier applies forward chaining rules to augment each graph with edges that logically follow from the existence of other edges already present in the graph. In the case of graph [2], the following edges would be added during elaboration

```
     (Integer ?card2), (>= ?card2 0),
     (Integer ?card3), (>= ?card3 0),
     (>= ?card2 1)
```

This last edge, representing our "missing edge", derives from the fact that **`?card3`** is non-negative, **`?card2`** is strictly greater than **`?card3`**, hence greater than zero, and that **`?card2`** is an integer. After applying elaboration to graph [2], the substitution σ successfully demonstrates that the relation **`At-Least-One-Son`** subsumes the relation **`More-Sons-Than-Daughters`**.

To our knowledge, no existing description classifier other than the PC classifier can compute this subsumption relation. We know this because none of them have the expressive power needed to *represent* the relation **`More-Sons-Than-Daughters`**. Here are two more relations that cannot be expressed in any existing description logic:

```
 (defrelation One-of-Five-Fastest-Ships (?s)
   :iff-def (and
      (Ship ?s)
      (<= (cardinality
             (setof (?fs)
                (faster-than ?fs ?s)))
          4)))
```

and

```
 (defrelation Third-Fastest-Ship (?s)
   :iff-def (and
       (Ship ?s)
       (= (cardinality
             (setof (?fs)
                (faster-than ?fs ?s)))
          2)))
```

Knowledge about upper and lower bounds (in this case, that "= 2" is a stricter constraint than "<= 4") is hardwired into the PC classifier. Since the remaining structure is identical between the two graphs, it is straightforward for the PC classifier to determine that the relation **`One-of-Five-Fastest-Ships`** subsumes the relation **`Third-Fastest-Ship`**.

Before we conclude our discussion of graph notation, recall that it does not provide a means for explicitly representing universally quantified variables. Instead, when the parser encounters an expression of the form

```
 (forall (?v1 ... ?vk)
    (implies <antecedent> <consequent>))
```
it transforms this expression into the equivalent expression
```
 (contained-in
    (setof (?v1 ... ?vk) <antecedent>)
    (setof (?v1 ... ?vk) <consequent>))
```
where **contained-in** is the subset/superset relation. In effect, reasoning about universally quantified variables is transformed into reasoning about set relationships. For example, the graph for
```
 (defrelation Relaxed-Parent (?p)
   :iff-def (and
      (Parent ?p)
      (forall (?c) (implies (child ?p ?c)
                            (Asleep ?c)))))
```
is
```
 (setof (?p) (and
    (Parent ?p)
    (contained-in (setof (?c) (child ?p ?c))
                  (setof (?c) (Asleep ?c)))))
```
Substituting a reference to the unary relation **Asleep** for the set of things satisfying the **Asleep** predicate yields
```
 (setof (?p) (and
    (Person ?p)
    (contained-in (setof (?c) (child ?p ?c))
                  (the-relation Asleep 1))))
```

The Subsumption Test

Let A and B be relations defined by expressions/graphs G_A and G_B. We apply the following test to prove that A subsumes B: If A is primitive (if its description is partial) then succeed if G_B explicitly inherits a relation known to specialize A. Formally, B specializes a primitive relation A if G_B contains an edge $R(dv_1, ... ,dv_k)$ where $dv_1, ... ,dv_k$ are the domain variables in the root node of G_B and R is a relation that specializes A. Otherwise (A is not primitive) succeed if there exists a substitution σ from nodes in G_A to nodes in G_B such that all of the following conditions hold:

(1a) If x is a constant node in G_A, then $\sigma(x)$ denotes the same constant, i.e., $\sigma(x) = x$;

(1b) If x is a set node in G_A, then $\sigma(x)$ is also a set node and definition(x) $\equiv_\sigma$ definition(σ(x)), where for all set nodes y, "definition(y)" refers to the subgraph that defines y, and "$\equiv_\sigma$" denotes structural equivalence under the substitution σ;

(2) If $P_A(x_1, ... ,x_k)$ is an edge in G_A then there exists an edge $P_B(\sigma(x_1), ... , \sigma(x_k))$ in G_B such that either

(i) $P_B = \sigma(P_A)$ or

(ii) P_A and P_B are relations and P_B specializes P_A, or

(iii) the edge $P_A(x_1, ... ,x_k)$ matches one the special cases 3a, 3b, 4a, 4b, 4c, or 4d;

(3a) If contained-in(x,R_A) is an edge in G_A and R_A is a relation, then there exists an edge contained-in(σ(x),R_B) in G_B such that R_B specializes R_A;

(3b) If contains(x,R_A) is an edge in G_A and R_A is a relation, then there exists an edge contains(σ(x),R_B) in G_B such that R_B subsumes R_A;

(4a) If >=(x,k_A) is an edge in G_A and k_A represents a numeric constant (k_A denotes a number) then there exists an edge >=(σ(x), k_B) in G_B such that value(k_B) >= value(k_A), where for all constant nodes k, "value(k)" is the denotation of k;

(4b, 4c, 4d) Analogous to (4a) for the relations <=, <, and >.

Remark: The alternatives (ii) and (iii) in condition 2 above serve to relax what would otherwise be a strictly *structural* subsumption test. Their inclusion in our test enables us to reduce the size of our graphs. For example, if one of our graphs contains both of the edges C(x) and C'(x) and C' specializes C, then we can eliminate the edge C(x) without sacrificing inferential completeness.

Canonical Graphs

To the best of our knowledge, all classifiers that utilize a structural subsumption test preface that test with a series of transformations designed to produce a "canonical" or "normalized" internal representation for each of the relations being tested. The strategy underlying these canonicalization transformations is to make otherwise dissimilar representations become as alike as possible, so that ideally, a structural test would suffice for determining subsumption relationships. For all but very restricted languages this strategy cannot result in a test for subsumption that is both sound and complete. For languages as expressive as NIKL , Loom, or BACK, theory tells us that a sound and complete decision procedure for testing subsumption relationships does not exist (i.e., subsumption testing is "undecidable") [Patel-Schneider 89]. The designers of structural subsumption-based classification systems have concluded that a strategy that relies on (imperfect) canonicalization transformations and a structural subsumption test represents a reasonable approach to "solving" this class of undecidable problems.

The PC classifier splits the normalization process into two phases. In the *canonicalization* phase, equivalence-preserving transformations (rewrite rules) are applied that substitute one kind of graph structure for another. In the subsequent *elaboration* phase, structure is added to a graph (again preserving semantic equivalence), but no structure is subtracted. The PC classifier implements several canonicalization strategies. The most important is the procedure that "expands" each of the edges in a graph that is labeled by a non-primitive relation. An edge with label R is expanded by substituting for the edge a copy of the graph for R. A second important canonicalization is one that substitutes an individual node for a nested set in cases

Representative Selection of Elaboration Rules

<u>Inequality rules</u>:	
I1 >= MIN and Number(MIN) and I2 >= I1 ⇒ I2 >= MIN	; propagate lower bound
I1 > MIN and Number(MIN) and I2 >= I1 ⇒ I2 > MIN	; propagate strict lower bound
I1 >= MIN and Number(MIN) and I2 > I1 ⇒ I2 > MIN	; propagate strict lower bound
I1 <= MAX and Number(MAX) and I2 <= I1 ⇒ I2 <= MAX	; propagate upper bound
I1 < MAX and Number(MAX) and I2 <= I1 ⇒ I2 < MAX	; propagate strict upper bound
I1 <= MAX and Number(MAX) and I2 < I1 ⇒ I2 < MAX	; propagate strict upper bound
I > MIN and Integer(I) ⇒ I >= floor(MIN) + 1	; round lower bound up
I >= MIN and Integer(I) and Number(MIN) and not(Integer(MIN)) ⇒ I >= floor(MIN) + 1	
I < MAX and Integer(I) ⇒ I <= ceiling(MAX) - 1	; round upper bound down
I <= MAX and Integer(I) and Number(MAX) and not(Integer(MAX)) ⇒ I <= ceiling(MAX) - 1	
I1 >= I2 and I2 >= I1 ⇒ I1 = I2	; equate two-way greater or equal
I1 >= I2 ⇒ I2 <= I1	; inverse greater or equal
I1 <= I2 ⇒ I2 >= I1	; inverse lesser or equal
I1 > I2 ⇒ I2 < I1	; inverse greater
I1 < I2 ⇒ I2 > I1	; inverse lesser
<u>Cardinality rules</u>:	
set(S) ⇒ exists(I) cardinality(S,I)	; sets have cardinalities
cardinality(S,I) ⇒ Integer(I)	; integer cardinality
cardinality(S,I) ⇒ I >= 0	; non-negative cardinality
contained-in(S1,S2) ⇒ cardinality(S1) <= cardinality(S2))	; greater cardinality superset
I >= MIN and I <= MAX and Integer(I) and Integer(MIN) and Integer(MAX) and domain-variable(S,I) and arity(S) = 1 ⇒ cardinality(S) <= MAX - MIN	
in(I,S) ⇒ cardinality(S) >= 1	; non-empty set
cardinality(S) = 1 and in(I,S) and in(J,S) ⇒ I = J	; equate members of singleton set
<u>Contained-in rules</u>:	
contained-in(S1,S2) and in(I,S1) ⇒ in(I,S2)	; propagate members up
contained-in(S1,S2) and cardinality(S1) = cardinality(S2) ⇒ S1 = S2	;equate equal cardinality superset
contained-in(S1,S2) and contained-in(S2,S3) ⇒ contained-in(S1,S3)	; transitivity of contained-in
contained-in(S1,S2) and contained-in(S2,S1) ⇒ S1 = S2	; equate two-way containment
S1 = S2 ⇒ contained-in(S1,S2)	; reflexivity of contained-in
contained-in(S1,S2) and contained-in(S1,S3) and intersection(S2,S3,S4) ⇒ contained-in(S1,S4)	; contained-in intersection set
contained-in(S1,S2) ⇒ contains(S2,S1)	; inverse contained-in
contains(S1,S2) ⇒ contained-in(S2,S1)	; inverse contains
<u>Other rules</u>:	
in(I,S1) and in(I,S2) and intersection(S1,S2,S3) ⇒ in(I,S3)	; member of intersection set
domain-variable(S1,I1) and arity(S1) = 1 and in(I1,S2) ⇒ contained-in(S1,S2)	

Table 1

when this transformation is guaranteed to preserve semantic equivalence.

Elaboration

This section describes two of the elaboration procedures implemented in the PC classifier.[1] Each of them implements a form of constraint propagation. Collectively, the constraint propagation procedures incorporated into the PC classifier implement four of the five classes of forward constraint propagation (all but Boolean constraint propagation) embodied in McAllester's SCREAMER system [McAllester&Siskind 93].

Elaboration Rules and Dual Representations

An "elaboration rule" is an if-then rule that adds edges (or occasionally, nodes) to a graph. Table 1 illustrates many of the elaboration rules used in the PC classifier. A comparison of our rules with those published by Borgida to describe the Classic classifier [Borgida 92] reveals that our rules tend to be finer grained than those in Classic,

[1]Other elaboration procedures include primitive edge expansion, recognition ,and realization.

enabling it, for example, to have a superior ability to reason about cardinality relationships (as evidenced by the sons-and-daughters and fastest-ships examples in Section 2).

A graph is elaborated by applying the rules in Table 1 repeatedly until no additional structure can be produced (the use of these rules is similar to the use of a "local" rule set [Givan&McAllester 92]). In addition, the elaboration procedure applies a structural subsumption test between each pair of nested sets, and adds a "contained-in" edge if it finds a subsumption relationship. Consider the following definition:

```
 (defrelation Brothers-Are-Friends (?p)
    :iff-def (contained-in
                 (setof (?b) (brother ?p ?b))
                 (setof (?f) (friend ?p ?f))))
```

The graph for this relation is

```
 (setof (?p) (exists (?s1 ?s2)
    (and
       (= ?s1 (setof (?b) (brother ?p ?b)))
       (= ?s2 (setof (?f) (friend ?p ?f))))
       (contained-in ?s1 ?s2))))
```

Applying the applicable elaboration rules results in the following graph

```
 (setof (?p) (exists (?s1 ?s2 ?card1 ?card2)
    (and
       (= ?s1 (setof (?b) (brother ?p ?b)))
       (= ?s2 (setof (?f) (friend ?p ?f))))
       (cardinality ?s1 ?card1)
       (cardinality ?s2 ?card2)
       (Integer ?card1) (>= ?card1 0)
       (Integer ?card2) (>= ?card2 0)
       (>= ?card2 ?card1) (<= ?card1 ?card2)
       (contained-in ?s1 ?s2)
       (contains ?s2 ?s1))))
```

Elaboration is applied to a graph for the purpose of making implicit structure explicit, and therefore accessible to our structural subsumption algorithm. We observe that our graph for **Brothers-Are-Friends** now has quite a bit of additional structure. The up side to elaboration is that when seeking to prove that Brothers-Are-Friends is subsumed by some other relation R, the additional structure increases the possibility that our subsumption test will discover that R subsumes **Brothers-Are-Friends** (because in this case the graph for **Brothers-Are-Friends** contains additional structure to map *to*). The down side is that the additional structure could make it less likely that the subsumption test finds the inverse subsumption relationship, i.e., that **Brothers-Are-Friends** subsumes R (because in this case the graph for **Brothers-Are-Friends** contains additional structure that must be mapped *from*). Intuitively, if G is a graph, applying elaboration rules to G makes it "easier" to classify G below another graph, but it makes it "harder" to classify another graph below G.

The standard answer to this apparent conundrum is to elaborate all graphs before computing subsumption relationships between them. We find two problems with the standard approach: (1) For this strategy to succeed, it is necessary to apply "the same amount" of elaboration to all graphs . As we shall soon see, in the scheme we have implemented for the PC classifier, the amount of elaboration applied to a graph is variable, depending on more than just the initial graph structure. (2) Because it causes the size of a graph to increase, elaboration degrades the performance of a subsumption algorithm at the same time that it increases the algorithm's completeness.

Dual Representations

Our solution is to maintain two separate graphs for each relation, one elaborated and one not. Given a relation R, let g(R) refer to the canonicalized but unelaborated graph for R, and let e-g(R) refer to the canonicalized *and* elaborated graph for R. To test if relation R subsumes relation S, our subsumption algorithm compares g(R) with e-g(S), i.e., it looks for a substitution that maps from the unelaborated graph for R to the elaborated graph for S.

This "dual representation" architecture completely solves the first of the two problems we cited above, and reduces the negative effect on performance of the second: (1) For relations R and S, increasing the amount of elaboration applied to e-g(R) increases the completeness of a test to determine if S subsumes R, without affecting the completeness of a test to determine if R subsumes S. (2) Assume that the cost of a subsumption test between two graphs is proportional to the product of the "sizes" of those graphs. If elaboration causes the size of each graph to grow by a factor of K, then the cost of comparing e-g(R) and e-g(S) is (K * K) times the cost of comparing g(R) and g(S). However the cost of comparing e-g(R) with g(S) is only K times the cost of comparing unelaborated graphs. Hence, according to this rough calculation, the standard elaboration strategy has a cost K times that of the dual representation strategy, where K is the ratio between the relative sizes of elaborated and unelaborated graphs.

Auto-Socratic Elaboration

A potentially serious drawback of conventional (overly aggressive) elaboration strategies is that they may generate graph structures that are never referenced by any subsequent subsumption tests (these represent a waste of both time and space). Alternatively, an overly timid strategy may suffer incompleteness by failing to generate structures that it should. This section introduces a new technique, called "auto-Socratic elaboration", that assists the classifier in controlling the generation of new graph structure.

Given a graph G, if we add a new set node N to G containing any definition whatsoever, but we do not add any new edges that relate N to previously existing nodes in G, then the denotation of G remains unchanged. Hence, this represents a legal elaboration of the graph G. Consider the following pairs of graphs:

"The set of things with at most two female children"

```
(setof (?p)                                   [3]
    (exists (?s0 ?card0) (and
      (= ?s0 (setof (?c) (and
                (child ?p ?c) (Female ?c)))
      (cardinality ?s0 ?card0)
      (>= 2 ?card0))))
```

"The set of things with at most two children"

```
(setof (?p) (exists (?s1 ?card1)        [4]
   (and (= ?s1 (setof (?c) (child ?p ?c)))
        (cardinality ?s1 ?card1)
        (>= 2 ?card1))))
```

In this section, we discuss the problem of determining that the graph [3] subsumes the graph [4]. Our structural subsumption test fails initially because no set node in [4] corresponds to the set node ?s0 in [3]. We can elaborate graph [4] by adding to it a new set node ?s2 having the same definition as that of ?s0, resulting in:

```
(setof (?p)                                   [5]
   (exists (?s1 ?card1 ?s2) (and
      (= ?s1 (setof (?c) (child ?p ?c)))
      (= ?s2 (setof (?c) (and
                (child ?p ?c) (Female ?c)))
      (cardinality ?s1 ?card1)
      (>= 2 ?card1))))
```

The elaboration procedure described in the previous section will apply a subsumption test to the pair <?s1,?s2>, resulting in the addition of the edge "contains(?s1,?s2)" (thereby making an implicit subsumption relationship explicit). Application of Table 1 elaboration rules yields

```
(setof (?p)                                   [6]
   (exists (?s1 ?card1 ?s2 ?card2) (and
      (= ?s1 (setof (?c) (child ?p ?c)))
      (= ?s2 (setof (?c) (and
                (child ?p ?c) (Female ?c)))
      (cardinality ?s1 ?card1)
      (cardinality ?s2 ?card2)
      (contains ?s1 ?s2)
      (>= 2 ?card1)
      (>= ?card1 ?card2)
      (>= 2 ?card2))))
```

Structural subsumption can determine that graph [3] subsumes graph [6], implying that graph [3] also subsumes graph [4]. It remains for us to specify how and when the PC classifier decides to add a new set node to a graph, as exemplified by the transformation from graph [4] to graph[5].

Our PC classifier implements a "demand-driven" strategy for adding new set nodes to a graph. If a test to determine if a graph G_A subsumes a graph G_B returns a negative result, and if the result is due to the identification of a set node N_A in G_A for which there is no set node in G_B having an equivalent definition, the following steps occur:

(1) A new set node N_B with definition equivalent to that for N_A (after substitution) is added to G_B;
(2) Tests are made to see if N_B subsumes or is subsumed by any other sets in G_B;
(3) If so, new contained-in edges are added, triggering additional elaboration of G_B;
(4) The subsumption test is repeated.

We call this procedure "auto-Socratic elaboration". "Socratic" inference [Crawford&Kuipers 89] refers to an inference scheme in which the posing of questions by an external agent triggers the addition (in forward chaining fashion) of new axioms to a prover's internal knowledge base. We refer to our procedure as "auto-Socratic" because in the PC classifier, the system is asking *itself* (subsumption) questions in the course of classifying a description, and its attempts to answer such questions may trigger forward-driven inferences (elaborations).

Performance

The PC classifier was compared with that of the Loom classifier on three different knowledge bases. The largest (containing approximately 1300 definitions) is a translated version of the Shared Domain Ontology (SDO) knowledge base used by researchers in the ARPA/Rome Labs Planning Initiative. The other two knowledge bases were synthetically generated using knowledge base generator procedures previously used in a benchmark of six classifiers performed at DKFI[Profitlich et al 92].[1] The results of Table 2 indicate that the Loom classifier is roughly twice as fast the PC classifier.[2]

[1]Loom was one of the faster classifiers in the DFKI benchmark.

[2]Testing was performed on a Hewlitt-Packard 730 running Lucid Common Lisp.

Knowledge Base	PC Classifier	Loom Classifier
SDO	80 seconds	45 seconds
Synthetic #1	58 seconds	22 seconds
Synthetic #2	45 seconds	34 seconds

Table 2

Completeness

DL languages have been developed that have complete classifiers. However, completeness comes at a steep price: the DL languages that support complete classification have very restricted expressiveness. While such languages are of theoretical interest, and may be useful for certain niche applications, their severe constraints limit their utility and preclude them from broad application.

In contrast, our approach provides a rich and highly expressive representation language. If this expressiveness is used, then classification must be incomplete. But where should it be incomplete? Different applications and domains will stress different sorts of reasoning. Inferences that are important in one will be inconsequential in another. A virtue of the PC architecture is that it is flexible and extensible. By changing elaboration rules, we can fine-tune the performance of the classifier, allowing us to change the kinds of inferences that are supported and tradeoff the breadth and depth of inference against efficiency. Thus, the PC classifier and language frees an application developer from a representational straitjacket by enhancing both the expressiveness of the language and the range of inference that can be supported.

Conclusions

DL languages have been developed that have complete classifiers. However, completeness comes at a steep price: the DL languages that support complete classification have very restricted expressiveness. While such languages are of theoretical interest, and may be useful for certain niche applications, their severe constraints limit their utility and preclude them from broad application.

In contrast, our approach provides a rich and highly expressive representation language. If this expressiveness is used, then classification must be incomplete. But where should it be incomplete? Different applications and domains will stress different sorts of reasoning. Inferences that are important in one will be inconsequential in another. A virtue of the PC architecture is that it is flexible and extensible. By changing elaboration rules, we can fine-tune the performance of the classifier, allowing us to change the kinds of inferences that are supported and tradeoff the breadth and depth of inference against efficiency. Thus, the PC classifier and language frees an application developer from a representational straitjacket by enhancing both the expressiveness of the language and the range of inference that can be supported.

Acknowledgments. I would like to thank Bill Swartout and Tom Russ for their edits and criticisms of earlier drafts of this paper. Eric Melz provided the timings listed in Table 2.

References [Baader&Hollunder 91] Franz Baader and Bernhard Hollunder, “KRIS: Knowledge Representation and Inference System”, *SIGART Bulletin*, 2(3), 1991, pp.8-14.

[Borgida et al 89] Alex Borgida, Ron Brachman, Deborah McGuinness, and Lori Halpern-Resnick, “CLASSIC: A Structural Data Model for Objects”, *Proc. of the 1989 ACM SIGMOD Int’l Conf. on Data*, 1989, pp.59-67.

[Borgida 92] Alex Borgida, “From Types to Knowledge Representation: Natural Semantics Specifications for Description Logics”, *International Journal on Cooperative and Intelligent Information Systems* [1,1] 1992.

[Crawford&Kuipers 89] J.M. Crawford and Benjamin Kuipers, “Towards a Theory of Access-Limited Logic for Knowledge Representation, *Proc. First Int’l Conf. on Principles of Knowledge Representation and Reasoning* , Toronto, Canada, May, 1989, pp.67-78.

[Doyle&Patil 91] Jon Doyle and Ramesh Patil, “Two Theses of Knowledge Representation: Language Restrictions, Taxonomic Classification, and the Utility of Representation Services”, *Artificial Intelligence*, 48, 1991, pp.261-297.

[Givan&McAllester 92] Robert Givan and David McAllester, “New Results on Local Inference Relations”, *Proc. Third Int’l Conf. on Principles of Knowledge Representation and Reasoning* , Cambridge, Massachusetts, October 1992. pp.403-412.

[MacGregor 90] Robert MacGregor, “The Evolving Technology of Classification-based Knowledge Representation Systems”, *Principles of Semantic Networks: Explorations in the Representation of Knowledge,* Chapter 13, John Sowa, Ed., Morgan-Kaufman, 1990.

[MacGregor 91] Robert MacGregor, “Using a Description Classifier to Enhance Deductive Inference”, *Proc. Seventh IEEE Conference on AI Applications*, Miami, Florida, February, 1991, pp 141-147.

[Mays et al 91] Eric Mays, Robert Dionne, and Robert Weida, K-REP System Overview, *SIGART Bulletin*, 2(3), 1991, pp.93-97.

[McAllester&Siskind 93] Jeffrey M. Siskind and David A. McAllester, “Nondeterministic Lisp as a Substrate for Constraint Logic Programming”, *AAAI-93 Proc. of the Eleventh Nat’l Conf. on Artificial Intelligence*, Washington, DC, pp.133-138.

[Patel-Schneider 89] Peter Patel-Schneider, “Undecidability of Subsumption in NIKL”, *Artificial Intelligence*, 39(2), 1989, pp.263-272.

[Peltason 91] “The BACK System - An Overview”,*SIGART Bulletin*, 2(3), 1991, pp.114-119.

[Profitlich et al 92] Jochen Heinsohn, Danial Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich, “An Empirical Analysis of Terminological Representation Systems”, AAAI-92 Proc. of the Tenth Nat’l Conf., San Jose, Calif., 1992, pp. 767-773.