Adornments in Database Programs

Inderpal Singh Mumick*
AT&T Bell Laboratories
Murray Hill, NJ 07974, USA.
mumick@research.att.com

Hamid Pirahesh
IBM Almaden Research Center
650 Harry Road, K55/801
San Jose, CA 95120, USA.
pirahesh@ibm.com

Raghu Ramakrishnan†
Computer Science Department
University of Wisconsin at Madison
Madison, WI 53706, USA.
raghu@cs.wisc.edu

Abstract

Several adornment schemes have been proposed to distinguish between calling patterns of goals using an abstract interpretation of a database program. The bound-free adornment scheme, often used with different meanings, is the most common. Condition, constraints, singleton, and EDB adornments have also been introduced. This paper formalizes adornments as languages, and answers the following questions: What do adornments mean? How are adornments used? What are the desirable properties of adornments? How do we know if a given set of adornments is effective? Why do we need different types of adornments? How do we compare different types of adornments? The work is important in understanding, improving, and developing query optimization techniques.

Keywords: *Deductive Databases, Logic Programming, Query Optimization, Abstract Interpretation.*

1 Introduction

Adornments were introduced in the context of optimizing database queries by Ullman [18] as annotations on predicates reflecting their calling patterns. Ullman used bound (b) and free (f) adornments to distinguish between bound and free arguments of a goal. The *bf* adornment scheme influenced many query optimization ideas: amongst them are the magic-sets transformation [2, 3],

*Parts of this work were done at Stanford University and IBM Almaden Research Center. Work at Stanford was supported by an NSF grant IRI-87-22886, an Air Force grant AFOSR-88-0266, and a grant of IBM Corporation.

†Part of this work was done while the author was visiting IBM Almaden Research Center. Work at Wisconsin was supported by an IBM Faculty Development Award and an NSF grant IRI-88-04319.

right-linear, left-linear, mixed-linear [16], context-right linear [6, 13], overbound right-linear [14], factoring [15], subgoal ordering [9], and C-transformation [1].

The basic ideas is to adorn an argument of a goal in a rule b if the argument is bound to a constant in *all* runtime goals generated from for this goal. However, the exact meaning of the b and f adornments was not defined formally in [18]. Consequently authors attached different semantics to the b and f adornments, perhaps without realizing that they were doing so. In magic templates [17], an argument of a subgoal was considered bound if the argument was potentially restricted (that is, it may be bound, or it may not be bound) by some predicate in the runtime goals generated from this subgoal.

The adornment idea goes beyond the b and f adornments. Mumick et al. [12] introduced a condition (c) adornment and used it to develop a ground magic-sets transformation for propagating constraints and bindings. Morris [10] introduced a singleton s adornment to represent a binding to a single constant, and used the s adornment to recognize when the right-linear optimization is applicable. She also defined an adornment to partly represent restrictions due to function symbols. Levy and Sagiv [8] introduced conjunctions of constraints as adornments, and used them to detect redundant rules. Levy et al. [7] introduced conjunctions of edb atoms as adornments, and used them to determine satisfiability of predicates.

This paper answers the following questions:

1. How are adornments used? What are the desirable properties of adornments. How do we compare adornment schemes?

   For instance, adornments or often used to represent restrictions. Optimization techniques try to propagate the restrictions represented by adornments, or use the restrictions to deduce properties of the programs. What are the properties an adornment class should have to represent restrictions? How do we compare adornment classes in the restrictions they represent? Can *better* adornments improve optimization techniques?

2. What is the meaning of an adornment? How do we define new adornment schemes?

   Adornments may be seen as an abstraction of the calling sets of goals. One may define an abstract domain of adornments, with each adornment representing a set of goals. The process of computing adornments is then a specialized abstract interpretation process. An adornment scheme is then a well defined abstract domain.

3. The c adornment of [12] represents *independent* conditions on one column, such as $X > 5$, or $X \leq 8$. *Dependent* conditions on two columns, such as $X > Y$, or $X \leq Y$ are not represented. Why?

Answers to the above questions can help us choose the right adornment scheme for the purpose at hand. They can help us in improving existing optimization and static analysis techniques. For

example, we can infer that the magic templates [17] and the C-transformation [1] can both be improved by using a different adornment scheme [11].

The concept of adornments is closely related to *modes* in the logic programming literature. Modes, like adornments, classify the argument positions of a predicate; modes classify argument positions into "input" and "output" positions. Algorithms for inferring modes of predicates (given modes for some query predicates) are closely related to algorithms for generating adorned programs. The questions that we consider can be recast in terms of modes and mode analysis.

Perhaps an even closer connection exists with respect to techniques for static analysis of logic programs, such as abstract interpretation. One common goal of many static analysis algorithms is to infer a collection of *calling patterns* (and corresponding *success patterns*) for the predicates in a program. The collection of adorned forms of a predicate can be viewed as being similar to a collection of calling patterns. Techniques for inferring calling patterns are closely related to techniques for adorning programs; and any criteria that we discuss for determining the quality of an adornment scheme should also shed light on the quality of an abstract domain (in terms of which calling patterns are determined). A careful comparison of the connections between adornments, modes and static analysis techniques in general is beyond the scope of this paper, but we ask the reader to keep these parallels in mind.

This paper is a definitional and classification paper. We define adornments and their desirable properties, both informally and formally. We also show how to compare adornment schemes. The paper is organized as follows: After the preliminaries (Section 2), we describe how adornments may be used in a system, and the properties adornments should have to be useful (Section 3). Section 4 formalizes the meaning of an adornment string as a set of runtime goals. Adornment classes are defined in Section 5. In Section 6 we show that an adornment class can be viewed as a language to represent restrictions. An adornment class that is more expressive is said to be more *descriptive*. Section 7 defines composibility: a property that allows an adorned predicate to be used as a common subexpression. Section 8 considers the ability of an adornment class to predict accurately the restrictions as they would appear during query evaluation, while Section 9 provides means to compare adornment classes. Related work and some new adornment classes are discussed in Section 10, and we conclude in Section 11.

2 Background

We use Datalog, as described in [19]. Datalog consists of horn clauses written as range-restricted rules. We allow six built-in predicates: $<, \leq, >, \geq, =$, and $\neq$. A letter, such as T, is used to represent a program comprised of rules $T1, T2, \ldots$

EXAMPLE 1 Rules $T1$ and $T2$ below comprise the transitive closure program T:

$(T1)$: $p(X,Y) :\!-\ e(X,Y).$
$(T2)$: $p(X,Y) :\!-\ e(X,Z)\ \&\ p(Z,Y).$

Each rule has two parts: an "if" part called the *body*, and a "then" part called the *head*. In rule $T1$, $e(X,Y)$ is the body, and $p(X,Y)$ is the head. In rule $T2$, $e(X,Z)\ \&\ p(Z,Y)$ is the body, and $p(X,Y)$ is the head.

Rules $T1$ and $T2$ compute the transitive closure, represented by p for `path`, of the edge relation, represented by e. □

Relations such as e and p are called *predicates*, or more specifically, *ordinary predicates*. An ordinary predicate whose extension is stored in the database is called an *extensional database* (edb) or *base* predicate, else it is called an *intensional database* (idb) or *derived* predicate. An edb predicate cannot appear in the head of any rule. An idb predicate must appear in the head of at least one rule. In program T, e is an edb predicate and p is an idb predicate.

2.1 Adornments

An adornment is an annotation on a predicate that gives information about how the predicate is used. The idea of bound (b) and free (f) adornments was introduced by Ullman [18]. We illustrate the idea of adornments through an example.

EXAMPLE 2 Consider the query Q:

(Q): ?- $p(1,Y)$.

on the transitive closure program T of Example 1. Query Q *binds* the first argument of p to the constant 1. The binding is used in a top-down evaluation of p. For example, in rule $T2$, the variable X gets bound to 1. We look up e using this binding, and get a set of bindings for variable Z. These bindings are then passed to the p subgoal, generating p goals that have the same *binding pattern* as the query goal, and will be evaluated in a similar manner.

Thus, whenever we generate a p goal during the top-down evaluation of query Q, the first argument of p will be bound to a constant. □

The presence or absence of a binding (equivalently, an equality predicate) on an argument of a predicate can be indicated by adorning the predicate. For example, the query goal Q can be adorned $p^{bf}(1,Y)$. The superscript bf is an adornment for predicate p, indicating that the first argument is bound (b) and the second argument is free (f). We do not adorn edb predicates.

Under the bound/free adornment scheme, the adornment of an n-ary predicate is defined to be a string of b's and f's of length n. A b in the i^{th} position means that the i^{th} argument is *bound*, while an f means that the i^{th} argument is *free*.

EXAMPLE 3 The flow of information during the top-down evaluation of query Q as discussed in Example 2 is represented by the adorned program AT:

$(AT0)$: ?- $p^{bf}(1, Y)$.

$(AT1)$: $p^{bf}(X, Y)$:- $e(X, Y)$.
$(AT2)$: $p^{bf}(X, Y)$:- $e(X, Z)$ & $p^{bf}(Z, Y)$.

Adorned predicates p^{bf} and p^{fb} are distinct from each other and from the unadorned predicate p, though the rules for p^{bf} and p^{fb} were derived from the rules for p. Program AT is called an adorned program. □

Definition 1 (*Informal*) **The** *bf* **Adornment Class:** The *bf* adornment class is a system of adornments that distinguishes between bound (b adorned) and free (f adorned) argument positions.

An argument of a subgoal in a rule is *bound* if, at runtime, every goal generated from this subgoal has a set of constants in this argument position. Otherwise, the argument is *free*. □

By Definition 1, an argument of a subgoal gets a b adornment if the argument is restricted by an *equality predicate*, either an equality selection predicate or an equijoin predicate. Otherwise the argument gets an f adornment.

2.2 *bcf* Adornments

We further build on the idea of adornments through the condition (c) adornment introduced in [12, 11] to represent independent conditions. A condition is any built-in predicate other than equality. For example, given relations $p(X, Y)$ and $q(Z)$, the local predicate $X > 1$ and the join predicate $Y \leq Z$ are conditions. Equality predicates $X = 1$ and $Y = Z$ are not conditions. A condition is independent if it refers to only one non-bound variable. The independent condition is said to be *on* the non-bound variable. A *dependent* condition is a condition that is not independent. For example, the condition $X > 5$ is independent. Condition $X > Y$ is independent if either X or Y is bound (but not both), otherwise condition $X > Y$ is dependent.

EXAMPLE 4 Consider the query U on the transitive closure program T of Example 1.

(U): ?- $X > 5$ & $p(X, Y)$.

The query U *conditions* the first argument of p to be greater than 5. The condition may be used in a top-down evaluation of p. For example, in rule $T2$, the variable X must also be greater than 5. We should look up e using this *condition*, and get a set of *bindings* for variable Z. These bindings are then passed to the p subgoal, generating p goals that have first argument *bound*.

The presence or absence of a condition or a binding on an argument of a predicate can be indicated by adorning the predicate. For example, the query goal in Q can be adorned $p^{cf}(X, Y)$. The

superscript cf is an adornment for predicate p, indicating that the first argument is conditioned (c) and the second argument is free (f). As before, we do not adorn edb predicates.

The condition $X > 5$ can be represented by the adorned query

$(AT0)$: ?- $X > 5$ & $p^{cf}(X, Y)$.

The propagation of the condition is represented by the adorned program AT:

$(AT1)$: $p^{cf}(X, Y) :\!- e(X, Y)$.
$(AT2)$: $p^{cf}(X, Y) :\!- e(X, Z)$ & $p^{bf}(Z, Y)$.

$(AT3)$: $p^{bf}(X, Y) :\!- e(X, Y)$.
$(AT4)$: $p^{bf}(X, Y) :\!- e(X, Z)$ & $p^{bf}(Z, Y)$.

Adorned predicates p^{cf} and p^{bf} are distinct from each other and from the unadorned predicate p. □

The adornment of an n-ary predicate is thus defined to be a string of b's, c's, and f's of length n. A b in the i^{th} position means that the i^{th} argument is *bound*, a c in the i^{th} position means that the i^{th} argument is *conditioned*, while an f means that the i^{th} argument is *free*.

Definition 2 *bcf* **Adornment Class:** The *bcf* adornment class is a system of adornments that distinguishes between bound (b adorned), independently conditioned (c adorned), and free (f adorned) argument positions.

An argument of a subgoal in a rule is *bound* if, at runtime, the goal generated from this subgoal has a set of constant in this argument position. The argument is *independently conditioned* if it is not bound and, at runtime, the goal generated from this subgoal has a set of constants or an independently conditioned variable in this argument position. Otherwise, the argument is *free*. □

2.3 *sbf*, *1sbf*, and more general adornments

An s adornment [10] indicates that an argument is bound to exactly one constant, as in

?- $:\!- p^{sf}(5, Y)$.

The 1 adornment is used for arguments that are bound to a functional term where the outermost functor is from a given set of functors. The adornment symbol is a functional term, as in

?- $p^{g(fs)f}(g(Z, 5), Y)$.

Actual constraints and edb atoms in adornments: Levy et al. [8, 7] define adornments that are a conjunction of constraints and edb literals. Any goal invoked with a particular adornment

can be satisfied by tuples that satisfy the constraints and use the edb literals in the adornment. However, the adornments don't represent constraints on goals during a top-down evaluation of the program: they predict what constraints would appear in the answers.

2.4 SIPS

A sideways information passing strategy (sips[1]) is a decision on how to pass information sideways in the body of a rule while evaluating the rule. A sips specifies how bindings available in the head will be used, the order in which subgoals in the body will be evaluated (join order), and how bindings will be passed between predicates in the body.

Formal definitions of sips for the *bf* and *bcf* adornment classes were given by Beeri and Ramakrishnan [3] and Mumick [11].

Definition 3 sips: The sideways information passing strategy, $\mathsf{sips}(r, \alpha)$, for a rule r and an head adornment α is a directed labeled graph. The edges and labels of the graph specify how information is to be passed between subgoals. The edges of the graph induce a partial order (sips order) in which the subgoals are to be evaluated, while the labels indicate the information to be passed from one subgoal to another. □

3 How are Adornments Used?

An adornment is an abstraction of a restriction on a goal. For instance, consider the query $Q1$:

$(Q1)$: ?- $p^{bf}(1, Y)$.

$(r1)$: $p(X, Y) :- q_1(X, Z)\ \&\ q_2(Z, Y)$.

The goal $p^{bf}(1, Y)$ asks for predicate $p(X, Y)$ to be computed with the restriction $X = 1$. The adornment *bf* abstracts the restriction as: X is bound to a finite set of constants.

An adornment is thus an element of an abstract domain. The concretization function maps an adornment to a set of runtime goals with the particular pattern of restrictions represented by the adornment.

Query Rewrite: The abstraction provided by adornments is used to guide query rewrite optimization. The magic-sets transformation [2, 3] uses the *bf* adornment class. The right-linear and left-linear optimizations [16, 6, 13, 14] use the *bf* adornment class, and can benefit from the *sbf* and *bcf* adornment classes. The ground magic-sets transformation [11] depends on the *bcf* adornment class. The C-transformation [1] depends on the *bf* adornment class, and can benefit from the *bcf* adornment class.

[1] Note that sips is an acronym for both, *a* sideways information passing strategy, and, *many* sideways information passing strategies.

Plan Optimization: A decision can be made to always use an index on a relation if the adornments indicate that an attribute will be bound to constants, or will have a condition. However, we must then guarantee that the binding/condition will be present at run-time.

Subgoal order: The subgoal order chosen to evaluate rule $r1$ for predicate p depends on the statistical information about the relations for subgoals q_1 and q_2, and on the restrictions on the $p(X, Y)$ goals to be answered. For example, knowing that X is bound is likely to lead to the subgoal order (q_1, q_2). If Y was bound instead, the subgoal order is likely to be (q_2, q_1). Thus, the efficient plan to evaluate a rule frequently varies with the nature of restrictions on the head. The head adornment abstracts the restriction on all the goals to be answered by evaluating the rule, and this abstraction can be used in the subgoal ordering algorithm. Adornments are used to guide subgoal ordering in *NAIL!* [10].

The abstraction provided by the *bf* adornments may not provide the information needed by more sophisticated subgoal ordering algorithms. For example, the adornment p^{bf} tells the subgoal ordering algorithm that X is bound, but does not provide any information about the finite set of constants to which X is bound. If we believe that the subgoal order does not usually depend on the actual bindings, then the abstraction provided by the b adornment is a good one. However the subgoal order can depend on the actual bindings. Example 5 demonstrates one such instance.

EXAMPLE 5 Let $q_1(X, Z)$ be a large relation indexed on X and Z, and let $X = 5$ be the least frequent X value, with a frequency of 1. Let $q_2(Z, Y)$ be a small relation indexed by Z. Then, given a head restriction $X = 5$, (q_1, q_2) is the best subgoal order. We first look up the single $q_1(5, Z)$ tuple, and then find the matching $q_2(Z, Y)$ tuples. However, if the head is given to be bound, but we don't know the binding, the subgoal order (q_2, q_1) is preferable. Looking up the q_1 relation on a given set of X values may retrieve many more tuples than first retrieving the full q_2 relation, and then selecting only the matching $q_1(X, Z)$ tuples. □

In relational databases the actual predicates, and not just the bound abstraction, are often used to influence the subgoal (join) ordering. Further, in systems such as IBM DB2 [5] the actual binding value may be used to select indices if there are spikes in the probability density function of attribute values.

3.1 Desirable Properties of Adornments

Granularity of Abstraction: When restrictions are being abstracted by adornments, it is the adornment, and not the actual restriction, that will be used in deciding how to evaluate a rule or how to rewrite a rule. If two restrictions are abstracted in an identical manner, then the rule is evaluated in an identical manner for the two restrictions. Consequently, if we expect to evaluate a rule differently for two restrictions, we should not abstract them with the same adornment.

For instance, if we expect to evaluate rules differently when a head variable X is bound to the set $\{5\}$ than when X is bound to any other finite set of constants, we should not use the same adornment for X in both cases. If we do choose to use the same adornment, the *granularity of abstraction* will be too coarse for us to differentiate between the two cases.

Similarly, if we expect that a rule will be evaluated differently when a head variable X is bound to a finite set of constants than when X is conditioned, we must not abstract a binding and a condition by the same adornment. We should use a *finer granularity of abstraction* that distinguishes between the two types of restrictions. This is the intuition for why we don't use the b adornment to represent conditions.

To summarize, each adornment class provides a certain *granularity of abstraction*, and we want this granularity to match or be finer than the granularity at which the optimization and rule evaluation decisions are made, or the granularity at which abstract operations are defined.

Accuracy of Abstraction: Adornments are computed at compile-time, and used to guide further compile-time optimizations, such as subgoal order determination and query rewrite. For these compile-time optimizations to lead to an efficient evaluation, and often for the optimizations to be correct, it is important that runtime goals actually have the restrictions predicted by the adornments. Also, if runtime goals do have certain restrictions, it is important that the adornment on the goal predict that the restrictions would be present. Otherwise, we may choose a suboptimal subgoal order, and we may loose out on possibilities for optimization.

Thus, a desirable property of an adornment class is that it should be possible to adorn programs such that each adorned predicate *abstracts accurately* the restrictions that would be present during evaluation of the rules.

Composibility: If restrictions on two goals are abstracted by the same adornment, then not only do we want to evaluate both goals using the same rules, but we may also want to evaluate the two goals together, in a set-oriented fashion. Consequently, the restriction at run-time would be the disjunction of the two individual restrictions.

EXAMPLE 6 Consider the adorned query V on the transitive closure program T:

(V): ?- $p^{bf}(1, Y)$ & $p^{bf}(2, Z)$.

One would like to treat p^{bf} as a common subexpression and evaluate the rules for p^{bf} only once during bottom-up evaluation, with the set $\{(1), (2)\}$ as the binding set for the first argument of p^{bf}. □

Since each adornment string can be viewed as an element of an abstract domain, composibility requires that the abstract merge operation be able to combine two identical strings into one instance of the string.

4 Adornments as Sets of Goals

The meaning of an adorned literal, say $p^{\alpha}(\bar{t})$ with adornment α in some adornment class $\mathcal{A}$, can be formalized through two concretization functions mapping adornments to sets of runtime goals: the first is the set of all *legal runtime goals* that can be generated for a $p^{\alpha}(\bar{t})$ subgoal during the execution of a program; the second is the set of *representative runtime goals* that have the restrictions represented by adornment α. A representative goal must be a legal goal, so the second set is a subset of the first set.

4.1 Legal Runtime Goals for an Adornment

In the *bf* adornment class, let $p^{bf}(X, Y)$ indicate that the first argument is bound to some finite set of constants and that nothing is known about the second argument. The set of legal runtime goals for $p^{bf}(X, Y)$ is then the set of goals in which the first argument is bound to a finite set of constants, while the second argument could be anything.

We denote the set of legal runtime goals described by an adorned literal l under some adornment class $\mathcal{A}$ by $\mathsf{legal}(l, \mathcal{A})$, to be read as the *legal set of goals for an adorned literal l under an adornment class $\mathcal{A}$*. When comparing two different adorned literals, we will require that their predicate names be identical; beyond this, the predicate names are not relevant. We will therefore talk of the "*set of goals described by an adornment string*" — it should be understood that we mean the "*set of goals described by an adorned literal with the given adornment string.* As an example, $\mathsf{legal}(bfcf, bcf)$ is an abbreviation for $\mathsf{legal}(p^{bfcf}, bcf)$.

When we define an adornment class, such as the *bcf* adornment class, we must specify the set of runtime goals that are legal for an adorned literal for every adornment string in the class being defined.

EXAMPLE 7 Our usage of the *bf*, *bcf*, and *sbf* adornment classes is consistent with the following: $\mathsf{legal}(p^{bf}, bf)$ is the set of goals $p(X, Y)$ where X is bound to a finite set of constants, and Y could be anything (free, have a condition $Y > 5$, or be bound to a set of constants). $\mathsf{legal}(p^{c}, bcf)$ is the set of goals $p(X)$ where X is bound to a finite set of constants, or X has some condition on it. $\mathsf{legal}(p^{cc}, bcf)$ includes goals $p(X, Y)$ with bindings or independent conditions on X and Y. The goal $p(X, Y)$ & $X > Y$ is not included. $\mathsf{legal}(p^{s}, sbf)$ includes goals $p(X)$ where X is bound to some one constant. □

4.2 Representative Runtime Goals for an Adornment

The set $\mathsf{legal}(p^{bf}, bf)$ includes the goal $p(5, 8)$, and the goal $p(5, Y)$ & $Y > 5$. However, we do not *intend to represent* a binding or a condition through the *f* adornment. We therefore do not consider these goals to be representative of p^{bf}. Similarly, we introduced the *c* adornment with an

intention to represent conditions, not bindings. The goal $p(5)$, though in $\mathsf{legal}(p^c, bcf)$, is therefore not representative of the adorned literal $p^c(X)$.

The representative set of goals for an adorned literal l under some adornment class $\mathcal{A}$, written $\mathsf{rep}(l, \mathcal{A})$, is the set of goals with restrictions that we *intend to represent* and then propagate by the given adornment. For the literal p^{bf}, the representative set consists of goals with first argument bound to a set of constants, and the second argument completely free. The representative set $\mathsf{rep}(l, \mathcal{A})$ must be a subset, not necessarily proper, of the legal set $\mathsf{legal}(l, \mathcal{A})$.

When we define an adornment class, such as the *bcf* adornment class, we must also specify the set of runtime goals that are representative of an adorned literal for every adornment string in the adornment class.

EXAMPLE 8 Our usage of the *bf* and *bcf* adornment classes is consistent with the following: $\mathsf{rep}(p^{bf}, bf)$ is the set of goals $p(X, Y)$ where X is bound to some finite set of constants, and Y is a free variable. The goal $p(5, Y)$ & $Y > 5$ is in $\mathsf{legal}(p^{bf}, bf)$, but is not included in $\mathsf{rep}(p^{bf}, bf)$. $\mathsf{rep}(p^c, bcf)$ is the set of goals where X has a condition on it. Goals with X bound are not included. □

The motivation behind having a representative set in addition to a legal set is that, when we want to utilize the information in the head adornment of a rule (say for p^{bf}), we must make some sort of a *worst-case* assumption about a goal, say g, that could invoke the rule. We know that $g \in \mathsf{legal}(p^{bf}, bf)$. Thus we know that X is bound to some finite set of constants. However, with the *bf* adornment class, this is all that we can safely assume — it is possible that the binding $Y = 5$, or the condition $Y > 5$ also holds (these goals are also in $\mathsf{legal}(p^{bff}, bf)$), but we cannot make use of these bindings or conditions in evaluating the rule. That is fine, since we are not trying to represent and propagate bindings or conditions through the f adornment of the *bf* adornment class. But, how do we know that we are not trying to represent a condition through the f adornment? One way is for the *bf* adornment class to define a representative set for the f adornment, indicating that the f adornment does not represent a condition.

We require that legal and rep sets both be specified for all adornment strings in every adornment class $\mathcal{A}$, as a part of the definition of class $\mathcal{A}$.

5 Adornment Classes

An adornment class is a collection of adornment strings. Since each adornment string denotes two sets of runtime goals, an adornment class can be viewed as a language to represent sets of goals. Each adornment string is a word in the language. The adornment strings are built from adornment symbols that form the alphabet of the language.

Definition 4 Adornment Class: An adornment class $\mathcal{A}$ consists of the following:

- An alphabet $\Sigma(\mathcal{A})$: The set of adornment symbols.
- A language $L(\mathcal{A}) \subseteq \Sigma^*$. Each word s in $L(\mathcal{A})$ is called an adornment string, or simply an adornment.
- For each adornment string $s \in L(\mathcal{A})$, the legal set of runtime goals: $\mathsf{legal}(s, \mathcal{A})$.
- For each adornment string $s \in L(\mathcal{A})$ the representative set of runtime goals: $\mathsf{rep}(s, \mathcal{A})$.

□

An adornment class is thus an abstract domain of adornments, with two concretization functions, legal and rep mapping adornments to sets of runtime goals. There is an abstraction mapping corresponding to each concretization function, with the provision that a goal can be abstracted by multiple adornments.

When we define an adornment class, we have in mind a certain set of restrictions we are trying to represent through the adornment class. We want to use the adornment class to express the propagation of this set of restrictions. The rep sets provide a formal mechanism to specify the set of restrictions we want to represent with each adornment string in the adornment class.

The arity of an adornment string s is the number of arguments of a literal adorned by string s. Thus, $\mathsf{arity}(bff) = 3$, $\mathsf{arity}(c) = 1$, and $\mathsf{arity}(cc) = 2$.

Often, we need only specify the legal and rep sets for adornment strings of arity 1. Given a longer adornment string $t = t_1, \ldots, t_n$ of arity n, where t_i is the i^{th} adornment symbol in string t, the legal and rep sets can be derived *symbol-by-symbol* as follows: The legal set for t is the set of goals whose i^{th} argument has the restriction specified by $\mathsf{legal}(t_i, \mathcal{A})$, and the rep set is the set of goals whose i^{th} argument has the restriction specified by $\mathsf{rep}(t_i, \mathcal{A})$, where $\mathcal{A}$ is the adornment class under consideration. Many of the adornment classes we consider have the *symbol-by-symbol derivation property*, however all adornment classes do not have such a property.

Definition 5 The bf **and** bcf **Adornment Classes:** The bf adornment class: $\Sigma = \{b, f\}$, $L = \{b, f\}^*$, $\mathsf{legal}(b, bf)$ = Goals with arguments bound to a finite set of constants, $\mathsf{legal}(f, bf)$ = All goals, $\mathsf{legal}(t, bf)$ = Goals obtained using symbol-by-symbol derivation, $\mathsf{rep}(b, bf) = \mathsf{legal}(b, bf)$, $\mathsf{rep}(f, bf)$ = Goals with argument free, and $\mathsf{rep}(t, bf)$ = Goals obtained using symbol-by-symbol derivation.

The bcf adornment class: $\Sigma = \{b, c, f\}$, $L = \{b, c, f\}^*$, $\mathsf{legal}(b, bcf) = \mathsf{legal}(b, bf)$, $\mathsf{legal}(c, bcf)$ = Goals with argument bound or independently conditioned, $\mathsf{legal}(f, bcf) = \mathsf{legal}(f, bf)$, $\mathsf{legal}(t, bcf)$ = Goals obtained using symbol-by-symbol derivation, $\mathsf{rep}(b, bcf) = \mathsf{rep}(b, bf)$, $\mathsf{rep}(c, bcf)$ = Goals with argument independently conditioned, $\mathsf{rep}(f, bcf) = \mathsf{rep}(f, bf)$, and $\mathsf{rep}(t, bcf)$ = Goals obtained using symbol-by-symbol derivation. □

We define the *Bf* adornment class used by Ramakrishnan [17], and the *sbf* adornment class used by Morris [10]:

Definition 6 The *Bf* and *sbf* Adornment Classes: The *Bf* adornment class: $\Sigma = \{B, f\}$, $L = \{B, f\}^*$, $\mathsf{legal}(B, Bf)$ = all goals, since a $p^B(X)$ goal is not *required* to have any restriction, $\mathsf{legal}(f, Bf)$ = all goals, $\mathsf{legal}(t, Bf)$ = all goals, $\mathsf{rep}(B, Bf) = \mathsf{legal}(B, Bf)$, $\mathsf{rep}(f, Bf) = \mathsf{rep}(f, bf)$, and $\mathsf{rep}(t, Bf)$ = Goals obtained using symbol-by-symbol derivation.

The *sbf* adornment class: $\Sigma = \{s, b, f\}$, $L = \{s, b, f\}^*$, $\mathsf{legal}(s, sbf)$ = Goals with argument bound to a single constant, $\mathsf{legal}(b, sbf)$ = Goals with argument bound to a finite set of constants, $\mathsf{legal}(f, sbf)$ = All goals, $\mathsf{legal}(t, bf)$ = Goals obtained using symbol-by-symbol derivation, $\mathsf{rep}(s, sbf) = \mathsf{legal}(s, sbf)$, $\mathsf{rep}(b, sbf) = \mathsf{legal}(b, sbf) - \mathsf{rep}(s, sbf)$, $\mathsf{rep}(f, sbf)$ = Goals with argument free.
$\mathsf{rep}(t, sbf)$ = Goals obtained using symbol-by-symbol derivation. □

We now represent dependent and independent conditions through the C adornment. The b and f adornments are interpreted as in the *bcf* class. The C adornment on an argument means that there is a condition on the argument, and the condition may depend on another argument having the C adornment, or the condition may be independent.

Definition 7 The *bCf* Adornment Class: For the *bCf* class, $\Sigma = \{b, C, f\}$ and $L = \{b, C, f\}^*$. The legal and rep sets for the b, C, and f adornment strings of arity 1 are the same as for the b, c, and f adornment strings in the *bcf* adornment class. However, the legal and rep sets for adornment strings containing more than one C adornment symbol cannot be obtained by symbol-by-symbol derivation.

The rep set for a string with i occurrences of adornment symbol C is the set of goals with conditions on all the C adorned arguments. The condition on each argument may be independent, or may be dependent on any subset of the other $i - 1$ arguments. For example, $\mathsf{rep}(p^{CC}, bCf)$ includes the goal $g1 = p(X, Y)$ & $X > 5$ & $Y > 5$ as well as the goal $g2 = p(X, Y)$ & $X > Y$. Note that the *bcf* class would have disallowed the latter goal. The restrictions on the b and f adorned arguments are still derived symbol-by-symbol.

The legal set includes the rep set and all goals with bindings on one or more C adorned arguments. The restrictions on the b and f adorned arguments are derived symbol-by-symbol. □

6 Descriptiveness

The previous section presented adornment classes as languages to express sets of goals. If an adornment class $\mathcal{A}_1$ can express more sets than another adornment class $\mathcal{A}_2$, then adornment class $\mathcal{A}_1$ is more *expressive* than the adornment class $\mathcal{A}_2$, and we will say that class $\mathcal{A}_1$ is more *descriptive* than class $\mathcal{A}_2$. We formalize the concept of descriptiveness in this section.

Figure 1 shows the relationships between the goals in the legal and rep sets of s, b, c, C, B,

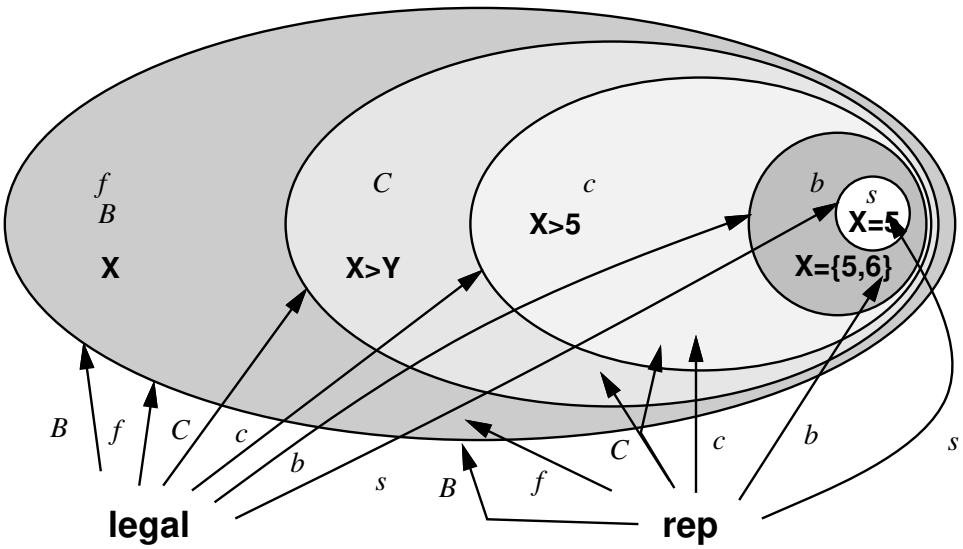

Figure 1: Relationships between legal and rep sets of s, b, c, C, B, and f adornments.

and f adornments, in the appropriate class(es) where the symbol is defined. Different adornments represent different sets of goals. For the c adornment, the legal set is the entire oval labeled c, while the rep set consists of the shaded area pointed to by the rep arrow labeled c. We note that $\mathsf{rep}(B, Bf)$ contains every other rep set, and $\mathsf{rep}(cc, bcf) \subset \mathsf{rep}(CC, bCf)$, the rep sets of b, c, and f adornments are disjoint. To compare the expressiveness of adornments, we need a subsumption relation between sets of goals.

6.1 Goal Subsumption

Definition 8 Goal Subsumption: A goal g_1 is subsumed by goal g_2, written as $g_1 \sqsubseteq g_2$, if goal g_2 can be further restricted to be made identical to goal g_1. A set of goals G_1 is subsumed by the set of goals G_2, written as $G_1 \sqsubseteq G_2$, if for every goal $g_1 \in G_1$, there is a goal $g_2 \in G_2$ such that $g_1 \sqsubseteq g_2$. If $g_1 \sqsubseteq g_2$ and $g_2 \not\sqsubseteq g_1$, we say $g_1 \sqsubset g_2$. □

EXAMPLE 9 $p(\{(5),(6)\}) \sqsubseteq p(\{(5),(6),(7)\})$, since we can add the restriction $(X = 5 \vee X = 6)$ to the goal $p(\{(5),(6),(7)\})$ to derive the goal $p(\{(5),(6)\})$. $p(5) \sqsubset p(X)$, $p(5) \sqsubset (p(X)\ \&\ X > 4)$, and $(p(X)\ \&\ X > 4) \sqsubset p(X)$. Note that $p(X) \not\sqsubseteq p(5)$, since we cannot add any restriction to $p(5)$ and derive the goal $p(X)$. In fact, $\mathsf{rep}(b, \mathcal{A}_1) \sqsubset \mathsf{rep}(c, bcf) \sqsubset \mathsf{rep}(f, \mathcal{A}_2)$ where $\mathcal{A}_1$ and $\mathcal{A}_2$ are any adornment classes where the b and f adornments are defined. □

Given two adornment strings α_1 and α_2, we would like to know which one denotes stronger restrictions. The adornment string with stronger restrictions is said to be more descriptive.

Definition 9 Descriptiveness of Adornment Strings: Given two adornment strings, α_1 in adornment class $\mathcal{A}_1$, and α_2 in adornment class $\mathcal{A}_2$, let l_1 and l_2 be adorned literals obtained by adorning the same predicate with adornment strings α_1 and α_2.

If $\mathsf{legal}(l_1, \mathcal{A}_1) \subseteq \mathsf{legal}(l_2, \mathcal{A}_2)$, and $\mathsf{rep}(l_1, \mathcal{A}_1) \sqsubseteq \mathsf{rep}(l_2, \mathcal{A}_2)$, then $(\alpha_1, \mathcal{A}_1)$ is **equally or more descriptive** ($\succeq$) than $(\alpha_2, \mathcal{A}_2)$, and $(\alpha_2, \mathcal{A}_2)$ is **equally or less descriptive** ($\preceq$) than $(\alpha_1, \mathcal{A}_1)$. If $(\alpha_1, \mathcal{A}_1) \succeq (\alpha_2, \mathcal{A}_2)$ and $(\alpha_2, \mathcal{A}_2) \succeq (\alpha_1, \mathcal{A}_1)$, then $(\alpha_1, \mathcal{A}_1)$ is **equally descriptive** ($=$) to $(\alpha_2, \mathcal{A}_2)$. If $(\alpha_1, \mathcal{A}_1) \succeq (\alpha_2, \mathcal{A}_2)$ and $(\alpha_1, \mathcal{A}_1) \neq (\alpha_2, \mathcal{A}_2)$, then $(\alpha_1, \mathcal{A}_1)$ is **more descriptive** ($\succ$) than $(\alpha_2, \mathcal{A}_2)$, and $(\alpha_2, \mathcal{A}_2)$ is **less descriptive** ($\prec$) than $(\alpha_1, \mathcal{A}_1)$. In absence of any of the above relations, $(\alpha_1, \mathcal{A}_1)$ and $(\alpha_2, \mathcal{A}_2)$ are incomparable. In particular, this is so if $\mathsf{arity}(\alpha_1) \neq \mathsf{arity}(\alpha_2)$. □

Lemma 1 *$(s, sbf) \succ (b, bcf) \succ (c, bcf) \succ (f, bcf)$, and $(cc, bcf) \succ (CC, bCf) \succ (ff, bcf)$, while $(b, bcf) = (b, bf) = (b, sbf)$ and $(f, bcf) = (f, bf)$, and $(B, Bf) = (f, Bf) = (f, bf) = (f, bcf)$. (f, bf) is incomparable to (ff, bf)* □

Proof: The results follow directly from the definitions of the legal and rep sets, and the definitions of comparative descriptiveness. □

Lemma 1 shows that the *cc* adornment represents a smaller set of goals than the CC adornment.

6.2 Descriptiveness of Adornment Classes

As languages for representing sets of goals, different adornment classes have different expressivity. Literals adorned according to an adornment class $\mathcal{A}_1$ can only represent certain sets of goals (the legal and rep sets of the literals in the class). An adornment class $\mathcal{A}_1$ that can represent more goal sets than an adornment class $\mathcal{A}_2$, defines goal sets at a finer *resolution*, and is therefore said to be *more descriptive*.

EXAMPLE 10 Using the *bcf* adornment class, $p^{bc}(X, Y)$ describes the set of $p(X, Y)$ goals where the first argument is bound to a finite set of constants, and the second argument is restricted by an independent condition. This set of goals cannot be expressed using the *bf* adornment class; we can only approximate it by $p^{bf}(X, Y)$, thereby losing the condition on the second argument.

The *bcf* adornment class is able to do a finer resolution of goal sets than is possible through the *bf* class, and is thus *more descriptive*. □

Definition 10 Descriptiveness of Adornment Classes: An adornment class $\mathcal{A}_1$ is said to be *more descriptive* than adornment class $\mathcal{A}_2$ if (1) For every adornment string α_2 in adornment class $\mathcal{A}_2$, there is an equally descriptive adornment string α_1 in adornment class $\mathcal{A}_1$, and (2) there is some adornment string α_1 in adornment class $\mathcal{A}_1$ such that no adornment string α_2 in adornment class $\mathcal{A}_2$ is equally descriptive to $(\alpha_1, \mathcal{A}_1)$. □

Condition 1 above says that every set of goals that can be represented in adornment class $\mathcal{A}_2$ can also be represented in adornment class $\mathcal{A}_1$. Condition 2 says that there is at least one set of goals that can be represented in adornment class $\mathcal{A}_1$, but cannot be represented in adornment class $\mathcal{A}_2$. Together, Conditions 1 and 2 imply that adornment class $\mathcal{A}_1$ represents more sets of goals, and is therefore more descriptive.

Two adornment classes are equally descriptive if for every adornment string in one class, there is an equally descriptive adornment string in the other class.

Lemma 2 *The bf adornment class is more descriptive than the Bf adornment class, the bcf adornment class is more descriptive than the bf adornment class, and the bcf adornment class is incomparable to the bCf adornment class. The 1sbf adornment class is more descriptive than the sbf adornment class, which is more descriptive than the bf adornment class, but these two are incomparable to the bcf adornment class.* □

Proof: From Lemma 1, $(f, bf) = (f, Bf) = (B, Bf)$. However, no adornment string in the Bf adornment class is equally descriptive to (b, bf). Hence, the bf adornment class is more descriptive than the Bf adornment class.

From Lemma 1, $(f, bcf) = (f, bf)$, $(b, bcf) = (b, bf)$. However, no adornment string in the bf adornment class is equally descriptive to (c, bcf). Hence, the bcf adornment class is more descriptive than the bf adornment class.

From Lemma 1, $(cc, bcf) \succ (CC, bCf)$. Hence, no adornment string in the bcf adornment class is equally descriptive to (CC, bCf), and no adornment string in the bCf adornment class is equally descriptive to (cc, bcf). Hence, the bcf adornment class is incomparable to the bCf adornment class. □

By making a finer resolution between sets of goals, a more descriptive adornment class describes restrictions more precisely. An optimization or subgoal ordering algorithm therefore gets more precise information from a more descriptive adornment class. Assuming that the extra information can be fruitfully utilized, a more descriptive adornment class is desirable. Thus, by Lemma 2, the bcf class is preferable to the bf class, which is preferable to the Bf adornment class. However, descriptiveness does not help chose between the bcf and bCf classes. That question will be answered by the loyalty property (Section 8).

6.3 Lattice Classes

The adornment strings of many adornment classes form a lattice with respect to the descriptiveness ($\succ$) relation.

Definition 11 Lattice Class: An adornment class $\mathcal{A}$ forms a lattice and is said to be a lattice class if it has (1) for every arity n, a top adornment string $\top(n)$ that is more descriptive than any other adornment string of arity n in $\mathcal{A}$, (2) for every arity n, a bottom adornment string $\bot(n)$ that is less descriptive than any other adornment string of arity n in $\mathcal{A}$, (3) a $glb(s_1, s_2)$ operator

that gives the most descriptive adornment string $s_3 \in \mathcal{A}$ such that $s_3 \preceq s_1$ and $s_3 \preceq s_2$, provided $\mathsf{arity}(s_1) = \mathsf{arity}(s_2)$. (4) a $lub(s_1, s_2)$ operator that gives the least descriptive adornment string $s_3 \in \mathcal{A}$ such that $s_3 \succeq s_1$ and $s_3 \succeq s_2$, provided $\mathsf{arity}(s_1) = \mathsf{arity}(s_2)$. □

EXAMPLE 11 The *bf*, *bcf*, and *sbf* adornment classes are lattice classes. For *bf* and *bcf* classes, b^n is the most descriptive adornment of arity n, while for the *sbf* adornment class, s^n is the most descriptive adornment of arity n. For each of the three classes, f^n is the least descriptive adornment of arity n. For every arity n, the total order $n^n \succ b^n \succ c^n \succ f^n$ defines the *glb* and *lub* operators. □

7 Composibility

Each element of the legal and rep sets gives us one possible goal for an adorned literal. Taking a disjunction of two goals from the above sets corresponds to constructing a goal with a union of binding sets and a disjunction of the conditions. For an adornment scheme to be composible, we must guarantee that this constructed goal is also in the legal set.

Definition 12 Goal Disjunction: Given two goals $g1$ and $g2$, the goal g is defined to be the disjunction of $g1$ and $g2$ if (1) goal g subsumes goals g_1 and g_2 ($g_1 \sqsubseteq g$ and $g_2 \sqsubseteq g$), and (2) if g' is any other goal that subsumes goals g_1 and g_2, then either (a) $g \sqsubseteq g'$, or (b) g and g' are incomparable. □

Definition 13 Composibility: Let α be any adornment string in adornment class $\mathcal{A}$, and let $g1$ and $g2$ be any two goals in the set $\mathsf{legal}(\alpha, \mathcal{A})$. Adornment class $\mathcal{A}$ is said to be composible if the disjunction of goals $g1$ and $g2$ is in the set $\mathsf{legal}(\alpha, \mathcal{A})$. □

Proposition 3 *The bf and 1bf adornment classes are composible. The sbf adornment class is not composible.* □

Proof: *(The bf adornment class):* Consider $\alpha = f$. Since every goal is in the set $\mathsf{legal}(f, bf)$, the composibility condition is satisfied. Consider $\alpha = b$. The disjunction of two goals restricted to sets of constants S_1 and S_2 is a goal g restricted to the set of constants $S = S_1 \cup S_2$. Goal $g \in \mathsf{legal}(b, bf)$, so the composibility condition is satisfied. For longer adornment strings, the composibility condition can be verified by symbol-by-symbol derivation.
(The 1bf adornment class): Consider $\alpha = 1$. The disjunction of two goals restricted to a functional term is a goal restricted to one or the other functional term. The resulting goal $g \in \mathsf{legal}(1, 1bf)$, so the composibility condition is satisfied. The remainder of the proof is the same as for the *bf* adornment class.
(The sbf adornment class): Consider $\alpha = s$, $g1 = p(5)$ and $g2 = p(6)$. Then the disjunction of goals $g1$ and $g2$ is $g = p(\{(5), (6)\})$. Goal g is not in $\mathsf{legal}(s, sbf)$, violating the composibility condition. □

A consequence of Proposition 3 is that an optimization algorithm using the *sbf* adornment class cannot create common subexpressions.

8 Loyal Adornment Property

An important property for an adornment class is its ability to *predict accurately* restrictions on runtime goals (Section 3.1). Indeed, the accurate prediction also depends on the adornment algorithm. In this section, we first formalize what we mean by an *accurate prediction* of restrictions, and then define a *loyal adornment property* that requires an adornment class to predict restrictions accurately.

Definition 14 Rule Evaluation: Consider the rule r:

(r): $\quad p^{\alpha}(\bar{t}) :- C \ \& \ q_1^{\alpha_1}(\bar{t}_1) \ \& \ \ldots \ \& \ q_n^{\alpha_n}(\bar{t}_n).$

where C represents the built-in predicates in the body of rule r, and $q_1^{\alpha_1} \ldots q_n^{\alpha_n}$ are the ordinary subgoals of rule r. Given a goal G for a p^{α} subgoal, an evaluation of rule r proceeds as follows:

for $i = 1$ **to** n
{Compute restrictions $f_i(G, C, \{q_1^{\alpha_1}(\bar{t}_1), \ldots, q_{i-1}^{\alpha_{i-1}}(\bar{t}_{i-1})\})$ on variables of $\bar{t}_i$;
Generate the goal $G_i = q_i^{\alpha_i}(\bar{t}_i) \ \& \ f_i$;
Evaluate the rules for $q_i^{\alpha_i}$ for the goal G_i;
}

f_i is a function that computes all restrictions over variables in $\bar{t}_i$ due to the goal G, built-in predicates C, and the evaluated subgoals $\{q_1^{\alpha_1}(\bar{t}_1) \ldots q_{i-1}^{\alpha_{i-1}}(\bar{t}_{i-1})\}$. The restrictions computed by f_i will include bindings and conditions for variables of $\bar{t}_i$. □

Definition 14 abstracts a top-down rule evaluation using a full sips based on the given subgoal order. The definition highlights the computation of all restrictions for each subgoal of the rule.

Definition 15 Legal Rule Adornment: Given an adornment class $\mathcal{A}$, let the rule r

(r): $\quad p^{\alpha}(\bar{t}) :- C \ \& \ q_1^{\alpha_1}(\bar{t}_1) \ \& \ \ldots \ \& \ q_n^{\alpha_n}(\bar{t}_n).$

be evaluated for any goal g in $\mathsf{legal}(p^{\alpha}, \mathcal{A})$. If for each i, the goal g_i generated for subgoal $q_i^{\alpha_i}$ is in $\mathsf{legal}(q_i^{\alpha_i}, \mathcal{A})$, then the rule is said to be *legally adorned* in adornment class $\mathcal{A}$.

A *program is legally adorned* if every rule in it is legally adorned. An adornment algorithm that produces legally adorned programs in class $\mathcal{A}$ is a *legal adornment algorithm* for adornment class $\mathcal{A}$. □

Definition 16 Tight Rule Adornment: Given an adornment class $\mathcal{A}$, let the rule r

(r): $\quad p^{\alpha}(\bar{t}) :- C \ \& \ q_1^{\alpha_1}(\bar{t}_1) \ \& \ \ldots \ \& \ q_n^{\alpha_n}(\bar{t}_n).$

be evaluated for a goal g in $\mathsf{rep}(p^{\alpha}, \mathcal{A})$. For each i, let g_i be the goal generated for subgoal $q_i^{\alpha_i}$, and let β_i be the most descriptive adornment in class $\mathcal{A}$ such that $g_i \in \mathsf{legal}(q_i^{\beta_i}, \mathcal{A})$.

If for each i, $(\alpha_i, \mathcal{A}) \succeq (\beta_i, \mathcal{A})$, then the rule is said to be *tightly adorned* in adornment class $\mathcal{A}$.

A *program is tightly adorned* if every rule in it is tightly adorned. An adornment algorithm that produces tightly adorned programs in class $\mathcal{A}$ is a *tight adornment algorithm* for adornment class $\mathcal{A}$. □

Tightness requires that the subgoal adornments in a rule predict all restrictions in runtime goals generated when the rule is evaluated for a representative goal for the rule head.

EXAMPLE 12 Consider the three rules:

$(r1)$: $p^f(X) :\!- q^b(X).$
$(r2)$: $p^b(X) :\!- q^f(X).$
$(r3)$: $p^b(X) :\!- q^b(X).$

Using the bf adornment class, (1) rule $r1$ is adorned tightly, but not legally. To see why the adornment is tight, consider a goal $p(X) \in \mathsf{rep}(p^f, bf)$. The rule generates the goal $q(X)$, best described by the adornment string f. The adornment string b used in rule $r1$ is more descriptive than adornment f, so the rule is tightly adorned. (2) rule $r2$ is adorned legally, but not tightly. To see why the adornment is not tight, consider a goal $p(5) \in \mathsf{rep}(p^b, bf)$. The rule generates the goal $q(5) \in \mathsf{legal}(q^b, bf)$. Since adornment string b is more descriptive than the adornment string f used in rule $r2$, the rule is not tightly adorned. (3) rule $r3$ is adorned tightly and legally. □

8.1 Combining Legal and Tight Adornments

Accurate prediction of restrictions on runtime goals requires (1) Never predict a restriction that will not be generated: A legally adorned rule ensures that every runtime goal is within the predicted set, and (2) Predict a restriction that will be generated: A tightly adorned rule ensures that the predicted sets are the most restrictive sets within which a goal generated from any representative head goal will lie.

Definition 17 Loyal Rule Adornment: A rule is said to be loyally adorned in adornment class $\mathcal{A}$ if it is legally and tightly adorned in adornment class $\mathcal{A}$. □

The rule $r3$ in Example 12 is adorned tightly and legally. Hence, rule $r3$ is adorned loyally. It is not always possible to adorn a rule in a legal and tight manner. For instance, using the bCf adornment class and the query

(Q): ?- $X > Y$ & $p^{CC}(X, Y)$.

the rule

$(r4)$: $p(X, Y) :\!- q_1(X)$ & $q_2(Y)$.

cannot be adorned loyally.

EXAMPLE 13 For the query Q and rule $r4$ above, the rule $r5$

$(r5)$: $\quad p^{CC}(X,Y) :\!-\, q_1^f(X)\ \&\ q_2^C(Y).$

is not adorned tightly. If the rule is evaluated for the head goal

$$(p(X,Y)\ \&\ X>5\ \&\ Y>5) \in \mathsf{rep}(p^{CC}, bCf),$$

we would generate the goal $q_1(X)\ \&\ X>5$. Now, the most descriptive adornment for this goal is q_1^C, not q_1^f.

However, it is not legal to use the adornment q_1^C. The head goal

$$p(X,Y)\ \&\ X>Y \in \mathsf{legal}(p^{CC}, bCf)$$

would generate the goal $q_1(X)$, with X free, and $q_1(X) \notin \mathsf{legal}(q_1^C, bCf)$.

We infer that rule $r5$ cannot be adorned both tightly and legally. The reason for this behavior is that abstracting both dependent and independent conditions with the same adornment was not a good idea. We want to propagate these conditions differently, and using the same adornment does not let us do so. The *bCf* adornment class is thus unable to predict accurately the restrictions generated from some of the head restrictions it tries to represent. □

Definition 18 Loyal Adornment Property: An adornment class $\mathcal{A}$ has the loyal adornment property if there exists an adornment algorithm that is legal and tight in adornment class $\mathcal{A}$. □

The loyal adornment property is a desirable property of adornment classes. If an adornment class has the loyal adornment property, we can adorn rules so as to predict accurately the restrictions on each runtime goal.

Proposition 4 *The bCf, sbf, and 1bf adornment classes do not have the loyal adornment property. The Bf adornment class has the loyal adornment property.* □

Proof:
(The bCf adornment class): Follows from Example 13.
(The sbf adornment class): The rule $r6$:

$(r6)$: $\quad p^s(X) :\!-\, q_1^{sf}(X,Y)\ \&\ q_2^s(Y).$

is adorned tightly and legally in the *sbf* adornment class if the functional dependency $X \rightarrow Y$ is known to hold in the relation $q_1(X,Y)$.

In the absence of the dependency $X \rightarrow Y$, rule $r6$ is not legally adorned. The alternative adornment in rule $r7$:

$(r7)$: $\quad p^s(X) :\!-\, q_1^{sf}(X,Y)\ \&\ q_2^b(Y).$

is adorned legally but not tightly. If the current extension of q_1 happens to have only one $q_1(X,Y)$ tuple for the single X value of interest, the subgoal $q_2(Y)$ is better described by the adornment $q_2^s(Y)$. Hence, the lack of loyalty.

(The 1bf adornment class: The presence of functional bindings in a goal can result from the binding of a variable to a functional term in the database. Since the presence of a functional term in the database cannot be determined at compile time, tightness is not possible.

(The Bf adornment class: Every adornment string of arity n is equally descriptive, and the legal sets for both B and f adornment strings include all goals. Thus, every adornment algorithm is legal and tight. □

Before deriving results about loyalty of *bf* and *bcf* adornment classes, we need an assumption about the manner in which we are permitted to deduce restrictions on variables during a top-down evaluation of rules.

Lemma 5 *Let $V_b = X_1, \ldots, X_n$ be a set of bound variables, $V_c = Y_1, \ldots, Y_m$ be a set of independently conditioned variables, and let C be a set of conditions (built-in predicates of the form "X op Y" or "X op a" where op is one of $\{=, \neq, >, \geq, <, \leq\}$, and a is a constant).*

Let $\mathsf{bound}(V_b, V_c, C)$ be the set of all variables for whom a binding is implied by V_b, V_c, and C, let $\mathsf{cond}(V_b, V_c, C)$ be the set of all variables for whom an independent condition is implied by V_b, V_c, and C, let $a_1, \ldots, a_n$ be a vector of symbolic constants, and let $C^+ = C \cup \{X_1 = a_1, \ldots, X_n = a_n\}$. Then, $\mathsf{bound}(V_b, V_c, C) = \mathsf{bound}(V_b, V_c, C^+)$ and $\mathsf{cond}(V_b, V_c, C) = \mathsf{cond}(V_b, V_c, C^+)$. □

Proof: Let Z be a variable in $\mathsf{bound}(V_b, V_c, C^+)$. Since $X_1, \ldots, X_n$ are bound, it is known for each X_i that $X_i = a_{i1} \vee X_i = a_{i2} \vee \ldots \vee X_i = a_{if(i)}$, for some constants $a_{i1}, a_{i2}, \ldots, a_{if(i)}$. Take a conjunction of the sets bound to each variable and convert into disjunctive normal form. Each disjunctive factor in the resulting expression must be of the form $F = \{X_1 = a_{1g(1)} \wedge X_2 = a_{2g(2)} \wedge \ldots \wedge X_n = a_{ng(n)}\}$. Since $a_{1g(1)}, \ldots, a_{ng(n)}$ are symbolic constants, $Z \in \mathsf{bound}(V_b, V_c, C^+) \Rightarrow Z \in \mathsf{bound}(F, V_c, C)$. If each disjunctive factor of $(V_b \wedge V_c \wedge C)$ implies a binding on Z, then the full disjunction also implies a binding on Z.

The proof for equality of the cond functions is similar. □

Property 1 *Using the notation of Lemma 5, let $C^{++} = C \cup \{X_1 = k_1, \ldots, X_n = k_n\}$, where each k_i is a given constant whose value is known. Then, $\mathsf{bound}(V_b, V_c, C^+) = \mathsf{bound}(V_b, V_c, C^{++})$ and $\mathsf{cond}(V_b, V_c, C^+) = \mathsf{cond}(V_b, V_c, C^{++})$* ⊙

Proposition 6 *If Property 1 holds, then the bf adornment class has the loyal adornment property.* □

Proof: Given the rule r:

(r): $p^\alpha(\bar{t}) :- C \;\&\; q_1^{\alpha_1}(\bar{t}_1) \;\&\; \ldots \;\&\; q_n^{\alpha_n}(\bar{t}_n)$.

consider the adornment algorithm that, for every subgoal $q_i(\bar{t}_i)$, first computes a bound set

$$\mathcal{B} = \mathsf{bvar}(p_h^\alpha, C, q_1(\bar{t}_1), \ldots, q_{i-1}(\bar{t}_{i-1})),$$

and then derives a b adornment for every argument of q_i that is equal to a constant, or equal to a variable in set $\mathcal{B}$. $\mathsf{bvar}(l)$ is a function that returns the set of variables bound by literals l. p_h^α is the head literal p^α restricted to its bound arguments. Such an adornment algorithm would adorn rule r legally and tightly.
Legality: Each variable in $\mathcal{B}$ is deduced to be bound from the bound head variables, built-in predicates, and bound subgoal variables for subgoals evaluated before q_i. The same bindings must be deducible when the

rule is evaluated and actual bindings are available for the subgoal variables and for head variables indicated bound by adornment α.

Tightness: Let if possible the adornment not be tight. Then there exists a goal G for $p^{\alpha}(\bar{t})$ that has a free variable in each f adorned argument, such that for some subgoal q_i, rule evaluation derives a binding for an argument X that was adorned f by the adornment algorithm. Hence, we have $X \notin \mathsf{bvar}(p_h^{\alpha}, C, q_1(\bar{t}_1), \ldots, q_{i-1}(\bar{t}_{i-1}))$, but $X \in \mathsf{bvar}(G, C, q_1(\bar{T}_1), \ldots, q_{i-1}(\bar{T}_{i-1}))$, where $\bar{T}_j$ is the set of bindings for arguments of q_j. The only difference between goal G and condensed head literal p_h^{α} is that goal G also provides the binding set for each bound variable. But then, by Lemma 5 and Hypothesis 1, $\mathsf{bvar}(p_h^{\alpha}, C, q_1(\bar{t}_1), \ldots, q_{i-1}(\bar{t}_{i-1})) = \mathsf{bvar}(G, C, q_1(\bar{T}_1), \ldots, q_{i-1}(\bar{T}_{i-1}))$, a contradiction to our assumption that the adornment on rule r is not tight. □

Proposition 7 *The bcf adornment class does not have the loyal adornment property.* □

Proof: The c adornment represents the presence of a condition, but forgets the actual condition. It is possible that a particular condition on a head variable X, in conjunction with the built-in subgoals in a rule, derives an independent condition on another variable Y that cannot be deduced from every condition on variable X. For example, given the goal $p(X)$ & $X < 5$, consider an evaluation of the rule $r8$:

$(r8)$: $\quad p^{c}(X) \text{ :- } Z > X \;\&\; q^{cf}(X, Z).$

Conditions $X < 5$ and $Z > X$ do not imply an independent condition on Z, so the adornment q^{cf} is tight. However, if rule $r8$ is evaluated for the goal $p(X)$ & $X > 5$, the conditions $X > 5$ and $Z > X$ derive the independent condition $Z > 5$, so that the q subgoal is best described by the adorned literal q^{cc}. Consequently, rule $r8$ is not adorned tightly. There is no way to adorn rule $r8$ tightly and legally, since the subgoal adornment $q^{cc}(X, Z)$ is illegal. □

This should not dishearten us. We usually do not want to deduce conditions at runtime, and deriving conditions as we did in the proof to Proposition 7 requires us to run a deduction system over the incoming conditions and the conditions in the rule. If we restrict function f_i in Definition 14 so that it does not make any deductions possible only from the conjunction of the goal conditions and the built-in predicates, then the *bcf* adornment class does have the loyal adornment property.

This should not dishearten us. Loss of loyalty is due to deduction of conditions at runtime, and we usually don't want to run a deduction system over the incoming conditions and the conditions in the rule. If we restrict function f_i in Definition 14 so that it does not make any deductions possible only from the conjunction of the goal conditions and the built-in predicates, then the *bcf* adornment class does have the loyal adornment property.

Proposition 8 *If Property 1 holds, and if, during rule evaluation, we do not make deductions possible only from the conjunction of the goal conditions and the built-in predicates in the body of the rule, then the bcf adornment class has the loyal adornment property.* □

Proof: Algorithm 2.3 in [11] is a tight and legal adornment algorithm for the *bcf* adornment class. We reproduce the algorithm here as Algorithm 1, and then prove that it is a tight and legal adornment algorithm.

Algorithm 1 (Find a full sips and adorn a rule.)

Input:

1. A Datalog rule r of the form:
   (r): $p :- q_1 \ \& \ q_2 \ \& \ \ldots \ \& \ q_m \ \& \ c_1 \ \& \ c_2 \ \& \ \ldots \ \& \ c_n$.
   where each q_j is an ordinary subgoal, and each c_i is a condition.
2. A head adornment p^α.

Output:

A full sips, $\mathsf{sips}(r, \alpha) = (V, E, B)$, and an adorned rule ar.

Method:

% *Step 1: Initialize*
$C = \emptyset$. % *Set of conditions*
$\mathcal{B} = \mathsf{bvar}(p_h^\alpha)$; $\mathcal{C} = \mathsf{cvar}(p_h^\alpha)$; % *Set of bound and conditioned variables*
$V = \{q_1, \ldots, q_m\}$; $E = B = \emptyset$; % *Initialize* $\mathsf{sips}(r, \alpha)$
for $(j = 1$ **to** $m)$
{% *Step 2.A: Determine the* j^{th} *subgoal*
$u_j = \mathsf{next}(r, (u_1, \ldots, u_{j-1}), C, \mathcal{B}, \mathcal{C})$; % *Find next subgoal*

% *Step 2.B: Derive* $C(j-1)$*; propagate effect to* C*,* $\mathcal{B}$*, and* $\mathcal{C}$
$C(j-1) = \{c_i \mid \mathsf{eligibility}(c_i) = j-1\}$; % *Conditions eligible at position* $j-1$
$C = C \cup C(j-1)$; % *Conditions eligible at or before position* $j-1$
$\mathcal{B} = \mathsf{bvar}(\{p_h^\alpha, C, u_1, u_2, \ldots, u_{j-1}\})$; % *bound variables before subgoal* u_j
$\mathcal{C} = \mathsf{cvar}(\{p_h^\alpha, C, u_1, u_2, \ldots, u_{j-1}\})$; % *conditioned variables before subgoal* u_j

% *Step 2.C: Extend* $\mathsf{sips}(r, \alpha)$ *to record information passed into* u_j
Insert the edge $e = (\{p_h^\alpha, C, u_1, \ldots, u_{j-1}\} \rightarrow u_j)$ into the edge set E;
Insert the node $\{p_h^\alpha, C, u_1, \ldots, u_{j-1}\}$ into the vertex set V;
$\beta(e) = \mathcal{B} \cap \mathsf{var}(u_j)$; $\kappa(e) = \mathcal{C} \cap \mathsf{var}(u_j)$; % *labels for edge* e
% *Step 2.D: Adorn* u_j
Compute adornment α_j for u_j due to the bound and condition labels $\beta(e)$ and $\kappa(e)$;

% *Step 2.E: Determine* $\mathcal{B}$ *and* $\mathcal{C}$ *for the next subgoal*
$\mathcal{B} = \mathcal{B} \cup \mathsf{var}(u_j)$; % *As* u_j *is evaluated, its variables get bound*
$\mathcal{C} = \mathcal{C} - \mathsf{var}(u_j)$;
}

% *Step 3: Determine conditions evaluable only after the last subgoal*
$C(m) = \{c_i \mid \mathsf{eligibility}(c_i) = m\}$;
The adorned rule ar is:
(ar): $p^\alpha :- C(0) \ \& \ u_1^{\alpha_1} \ \& \ C(1) \ \& \ u_2^{\alpha_2} \ \& \ \ldots \ \& \ C(m-1) \ \& \ u_m^{\alpha_m} \ \& \ C(m)$.

◇

Figure 2: A tight and legal adornment algorithm for the *bcf* adornment class.

Legality: Each variable in $\mathcal{B}$ and $\mathcal{C}$ is deduced to be bound or conditioned from the bound head variables, conditioned head variables, built-in predicates, and the bound subgoal variables for subgoals evaluated before q_i. The same bindings and conditions must be deducible when the rule is evaluated for a goal and actual bindings or conditions are known for all head and subgoal variables assumed bound or conditioned when computing sets $\mathcal{B}$ and $\mathcal{C}$.

Tightness: Let if possible Algorithm 1 not be tight. Then there exists a goal G for $p^{\alpha}(\bar{t})$ that has a free variable in each f adorned argument and an independent condition in each c adorned argument, such that for some argument X of a subgoal q_i, rule evaluation derives a stronger restriction than indicated by the adornment derived by Algorithm 1. This would mean that knowing the actual subgoal bindings and the actual goal conditions and bindings, bvar or cvar are able to deduce a binding or a condition for variable X that is not deducible without knowing the actual bindings or conditions. But, from Lemma 5 and Hypothesis 1 knowing the actual bindings does not lead to any new deductions, the proposition specifically prohibits making deductions from the conjunction of actual goal conditions and built-in literals, and amongst themselves, the actual goal conditions cannot derive any new binding or condition since they are all given to be independent. □

9 Faithful Adornment Property

The loyalty property tells us if an adornment class, seen by itself, can accurately predict restrictions on runtime goals by doing an abstract interpretation at compile time. We now develop a property that lets us compare two adornment classes for their relative accuracy of prediction. To motivate the definitions that follow, consider the following example.

EXAMPLE 14 For the query $Q2$ on rule $r7$

$(Q2)$: ?- $X > 5$ & $p^f(X)$.

$(r9)$: $p^f(X)$:- $q^f(X)$.

adorned using the bf adornment class, let us see what the bcf adornment class can do. The goal $X > 5$ & $p(X)$ can be represented by the adorned literal p^c. Further, if rule $r9$ is evaluated for the goal $X > 5$ & $p(X)$, we would generate the goal $X > 5$ & $q(X)$ that is best described by q^c, which is more descriptive than the adorned literal q^f used in rule $r9$.

Thus, the bcf adornment class describes the restrictions on the head of rule $r9$, and their propagation into the body of the rule more accurately than the bf adornment class. □

Definition 19 Faithful Rule Adornment: The adorned rule r

(r): $p^{\alpha}(\bar{t})$:- $q_1^{\alpha_1}(\bar{t}_1)$ & ... & $q_n^{\alpha_n}(\bar{t}_n)$.

legally adorned in adornment class $\mathcal{A}_1$ is said to be faithfully adorned with respect to adornment class $\mathcal{A}_2$ if condition **D** below holds for any choice of an adornment string $\beta \in \mathcal{A}_2$ that satisfies the following two conditions:

1. $(\beta, \mathcal{A}_2) \succeq (\alpha, \mathcal{A}_1)$, and

2. If there exists an adornment string $\gamma \in \mathcal{A}_1$ that is more descriptive than $\alpha \in \mathcal{A}_1$, then it is not the case that $(\beta, \mathcal{A}_2) \succeq (\gamma, \mathcal{A}_1)$.

(D): Let rule r be evaluated for any goal G in $\mathsf{rep}(p^{\beta}, \mathcal{A}_2)$, and let G_i be the goal generated for the i^{th} subgoal q_i. Let $q_i^{\beta_i}$ be the most descriptive adornment in $\mathcal{A}_2$ such that $G_i \in \mathsf{legal}(q_i^{\beta_i}, \mathcal{A}_2)$. Then, for all i, $(q_i^{\alpha_i}, \mathcal{A}_1) \succeq (q_i^{\beta_i}, \mathcal{A}_2)$. □

The definition is based on the following intuition: suppose we find an adornment string $\beta \in \mathcal{A}_2$ that is *sandwiched* between α and any adornment string $\gamma \in \mathcal{A}_1$ more descriptive than α, and the use of the β adornment in the head lets us describe a subgoal by an adornment more descriptive than the one used in rule r. We then conclude that, for the goals in $\mathsf{rep}(\beta, \mathcal{A}_2)$, an adornment of rule r using adornment class $\mathcal{A}_2$ could better describe the presence and propagation of restrictions. Hence, we say that rule r is not faithfully adorned with respect to class $\mathcal{A}_2$.

EXAMPLE 15 Rule $r9$ in Example 14 adorned in the *bf* adornment class is not faithfully adorned with respect to the *bcf* adornment class. □

Note that if $\mathcal{A}_2 = \mathcal{A}_1$, the only choice for adornment string β is α, and faithful rule adornment (Definition 19) reduces to loyal rule adornment (Definition 17).

Definition 20 Faithful Adornment Property: Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be adornment classes and let L be a legal adornment algorithm in class $\mathcal{A}_1$. We say that $(\mathcal{A}_1, L)$ has the *faithful adornment property with respect to adornment class* $\mathcal{A}_2$ if, for every legally adorned program AP produced by algorithm L, every rule in program AP is faithfully adorned with respect to adornment class $\mathcal{A}_2$.

Adornment class $\mathcal{A}_1$ is said to have the faithful adornment property with respect to adornment class $\mathcal{A}_2$ if there exists a legal adornment algorithm L such that $(\mathcal{A}_1, L)$ has the faithful adornment property with respect to class $\mathcal{A}_2$. □

If an adornment class $\mathcal{A}_1$ does not have the faithful adornment property with respect to another class $\mathcal{A}_2$, then using class $\mathcal{A}_2$ we can represent propagation of restrictions that cannot be expressed in class $\mathcal{A}_1$.

The faithful adornment property is related to descriptiveness and loyalty of classes.

Lemma 9 *An adornment class $\mathcal{A}_1$ has the faithful adornment property with respect to itself iff the adornment class $\mathcal{A}_1$ has the loyal adornment property (Definition 18).* □

Proof: A rule adorned in class $\mathcal{A}_1$ can be faithfully adorned with respect to class $\mathcal{A}_1$ if, for no head goal, can a subgoal be adorned more tightly using another adornment in class $\mathcal{A}_1$. Since the adornments are required to be legal, the above condition implies that the original rule adornment be legal and tight. □

Theorem 1 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be lattice classes whose bottom adornments are equally descriptive for every arity n. If adornment class $\mathcal{A}_2$ is more descriptive than adornment class $\mathcal{A}_1$, then class $\mathcal{A}_1$ does not have the faithful adornment property with respect to class $\mathcal{A}_2$.* □

Proof: Let $\beta \in \mathcal{A}_2$ be an adornment string of arity 1 for which there is no equally descriptive adornment in adornment class $\mathcal{A}_1$. Let $\alpha_2 \in \mathcal{A}_2$ be an adornment string that is just below β in $\mathcal{A}_2$'s lattice. That is, $(\beta, \mathcal{A}_2) \succ (\alpha_2, \mathcal{A}_2)$, and there does not exist another adornment string $\lambda \in \mathcal{A}_2$ such that $(\beta, \mathcal{A}_2) \succ (\lambda, \mathcal{A}_2) \succ (\alpha_2, \mathcal{A}_2)$. Now, adornment α_2 may or may not have an equally descriptive adornment string in class $\mathcal{A}_1$. If it does not, our choice of β was incorrect. Assign $\beta := \alpha_2$, and redo the exercise above. Since the bottom adornments in classes $\mathcal{A}_1$ and $\mathcal{A}_2$ are equally descriptive, we must eventually find an adornment string α_2 for which there is an equally descriptive adornment string $\alpha_1 \in \mathcal{A}_1$.

Let γ_1 be any adornment string in class $\mathcal{A}_2$ that is more descriptive than α_1. Then, $(\beta, \mathcal{A}_2) \not\succeq (\gamma_1, \mathcal{A}_1)$, because if it were, then there must be an adornment $\gamma_2 \in \mathcal{A}_2$ that is equally descriptive to γ_1 (since class $\mathcal{A}_2$ is given to be more descriptive than class $\mathcal{A}_1$), and γ_2 would be between β and α_2 in the lattice for class $\mathcal{A}_2$. However we selected β and α_2 such that there is no adornment between them.

Now, consider rule $r8$ legally adorned in class $\mathcal{A}_1$:

$(r8)$: $\quad p^{\alpha_1}(X) \text{ :- } q^{\alpha_1}(X).$

Let rule $r8$ be evaluated for any goal $G_p \in \mathsf{rep}(p^\beta, \mathcal{A}_2)$. The goal G_q generated for q has the same restrictions as G_p, so $G_q \in \mathsf{legal}(q^\beta, \mathcal{A}_2)$. By Definition 19, rule $r8$ is not faithfully adorned with respect to adornment class $\mathcal{A}_2$.

Given the head adornment p^{α_1}, the only other legal adornment option for rule $r8$ is to adorn the subgoal q with an adornment μ equally or less descriptive than α_1. In either case, the adornment β constructed above would make the rule adornment unfaithful.

We conclude that there is no adornment algorithm L for which $(\mathcal{A}_1, L)$ has the faithful adornment property with respect to adornment class $\mathcal{A}_2$. □

An application of Theorem 1 to descriptiveness results obtained in Lemma 1 leads to Corollary 2.

Corollary 2 *The Bf adornment class does not have the faithful adornment property with respect to the bf adornment class. The bf adornment class does not have the faithful adornment property with respect to the bcf and sbf adornment classes.* □

Proposition 10 *The bcf and bCf adornment classes do not have the faithful adornment property with respect to each other.* □

Proof: The C adornment can be sandwiched between the f and c adornments to improve upon a legally adorned rule in the *bcf* class for the goal $p(X, Y)$ & $X > Y$.

The c adornment can be sandwiched between the C and b adornments to improve upon a legally adorned rule in the *bCf* class for the goal $p(X, Y)$ & $X > 5$ & $Y > 5$. □

We have defined the faithful adornment property using a model of rule evaluation based on full sips. We can define faithful rule adornments for partial sips too. The idea is that all restrictions that would be computed at runtime using the partial sips be predicted accurately by the adornments.

10 Extensions and Related Work

The formal framework in this paper will motivate definitions of new adornment classes and optimization techniques to represent different sets of restrictions. For instance, The *1sbf* and the *bcf* adornment classes can be combined to form a *1sbcf* adornment class, and we can extend the *bf* and *bcf* adornment classes to handle more general conditions. The most general adornment class possible is one where the actual restrictions are used as adornment symbols, as is proposed by [8, 7], where the adornment strings provide hardly any abstraction, they give the actual condition. However, the size of the adorned program is prohibitive.

Refined *bf* and *bcf* Adornment Classes: We have defined two adornment classes, the refined *bf*, and the refined *bcf* class, to represent aliasing and conditions between arguments of a goal. Let X and Y be free arguments of a predicate p. The aliasing $X = Y$ is represented by an adornment symbol b_1 on both X and Y, causing X and Y to form an equivalence class of equated arguments. Similarly, the dependent condition $X > Y$ is represented by the symbol c_1 on both X and Y. Thus query Q

(Q): ?- $X > Y$ & $p(X, Y, Z, Z)$.

(r): $p(X, Y, Z, W) :\!-\ q1(X, U)$ & $q2(U, Y)$ & $q3((Z, U)$ & $q4(V, W)$.

is adorned as:

(AQ): ?- $X > Y$ & $p^{c_1 c_1 b_1 b_1}(X, Y, Z, Z)$.

(Ar): $p^{c_1 c_1 b_1 b_1}(X, Y, Z, W) :\!-\ q1^{\mathrm{ff}}(X, U)$ & $q2^{b_1 c_1}(U, Y)$ & $q3^{\mathrm{ff}}(Z, V)$ & $q4^{b_2 b_3}(V, W)$.

Related Work: This paper expands on the treatment of *c* adornment in [12]. Morris [10] formalizes adornments by the relations to which an adorned argument might be restricted. For instance, the *s* adornment is formalized as the set of relations that have just one constant tuple. The *b* adornment is formalized as the set of relations that have a finite number of constant tuples. The *f* adornment is formalized as the set of all relations including a special $\{\top\}$ relation that represents the infinite relation. This formalization is complementary to the legal set of goals used by us, with each goal corresponding to a relation. Morris does not have a counterpart for the representative sets we define. The goal of Morris's work is to define algorithms to adorn rules and goals with various adornment classes, rather than describe and compare merits of adornment schemes. Towards that end she defines adornment lattices, discusses several properties of adornment lattices, and gives adornment algorithms for use with any adornment lattice. She does not talk about desirable properties of adornments, or about various ways in which adornments may be

used.

Adornments are related to abstract interpretation [4], and we have pointed out the relationship at appropriate points in the paper. Computing adornments can be seen as an abstract interpretation of a program: compile-time characterization of goals generated for the program. In fact, an abstract interpretation similar to [4] can be used to compute adornments. Adornments can also be used to guide further abstract interpretation, since they characterize properties of all calls for a particular predicate, leading to more efficient abstract interpretation. Abstract interpretation tries to classify run-time goals, and the answers to the abstracted goals. Adornments classify goals, but do not classify the answers to the goals. However, methods for evaluation of the goals are classified using adornments. We have not discussed algorithms to compute adornments. Such algorithms would be special cases of abstract interpretation. Descriptiveness, legality, and tightness properties of adornments are related to coarseness, soundness, and preciseness of abstract interpretation techniques. An abstract domain of goals whose elements are two coarse is less descriptive. If the domain elements describe goals that do not occur at runtime, it is imprecise. In our framework it leads to non-tight, and thereby non-loyal adornments. Thus, if the abstract domain in an abstract interpretation is describing *call patterns*, it will help to chose a more descriptive and loyal adornment class.

11 Conclusions

We formalize adornments as a *language* to represent sets of run-time goals. The formalization is useful in defining and comparing the expressivities of the various adornment classes. For example, the expressivity of the *bf* adornment class can be increased by adding an *s* or *c* adornments.

We define composibility to check if an adornment class preserves restrictions when creating common subexpressions, and loyalty to check if an adornment class can predict accurately the restrictions on runtime goals. Faithfulness compares adornment on their accuracy of prediction. The *bf* adornment class is composible and loyal. More expressive classes, such as the *bcf* and *sbf* classes, are richer in representing restrictions, but one has to be careful when using them. For example, the *sbf* class cannot always accurately predict the larger class of restrictions, and is not composible. The *bcf* adornment class is loyal provided we don't deduce conditions at runtime.

The analysis and definitions in this paper are important to query optimization and abstract interpretation, and will lead to new optimization techniques and adornment classes. Composibility and loyalty properties guide the design of optimization techniques. For instance, an optimizer must not create common subexpressions of *s* adorned goals since *sbf* class is not composible. Descriptiveness describes the expressivity of an adornment, and is useful in (defining and) selecting an adornment class appropriate for the task at hand.

12 Acknowledgements

We thank Katherine Morris and Jeffrey D. Ullman for several discussions on adornments. The Starburst project at IBM Almaden Research Center and the *NAIL!* project at Stanford University provided a stimulating environment for this work. Ashish Gupta, Yatin Saraiya, Oded Shmueli, and Gio Wiederhold provided helpful comments on drafts of this paper.

References

[1] Isaac Balbin, David B. Kemp, Krishnamurthy Meenakshi, and Kotagiri Ramamohanarao. Propagating constraints in recursive deductive databases. In *North American Conference on Logic Programming (NACLP)*, October 16-20 1989.

[2] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth Symposium on Principles of Database Systems (PODS)*, pages 1–15, Cambridge, MA, March 1986.

[3] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *Journal of Logic Programming*, 10:255–300, 1991. Preliminary version appeared in the 6th ACM Symposium on Principles of Database Systems, 1987.

[4] Saumya Debray and Raghu Ramakrishnan. Canonical computation of logic programs. *Journal of Logic Programming*, 1993. To Appear.

[5] IBM. *IBM Database 2 Administration Guide*, March 1992.

[6] David B. Kemp, Kotagiri Ramamohanarao, and Zoltan Somogyi. Right-, left- and multi-linear rule transformations that maintain context information. In *Proceedings of the Sixteenth International Conference on Very Large Databases (VLDB)*, Brisbane, Australia, August 13-16 1990.

[7] Alon Levy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. Equivalence, query-reachability, and satisfiability in Datalog. In *Proceedings of the Twelfth Symposium on Principles of Database Systems (PODS)*, pages 109–122, Washington, DC, May 25-27 1993.

[8] Alon Levy and Yehoshua Sagiv. Constraints and redundancy in datalog. In *Proceedings of the Eleventh Symposium on Principles of Database Systems (PODS)*, pages 67–80, San Diego, CA, June 2-4 1992.

[9] Katherine A. Morris. An algorithm for ordering subgoals in NAIL!. In *Proceedings of the Seventh Symposium on Principles of Database Systems (PODS)*, pages 82–88, Austin, TX, March 1988.

[10] Katherine A. Morris. *Subgoal Order for Query Optimization in Logic Databases.* PhD thesis, Stanford University, Stanford, CA 94305, USA, April 1991.

[11] Inderpal Singh Mumick. *Query Optimization in Deductive and Relational Databases.* PhD thesis, Stanford University, Stanford, CA 94305, USA, December 1991. Technical Report No. STAN-CS-91-1400. Also available from University Microfilms International, 300 N. Zeeb Road, Ann Arbor, MI 48106. (313)761-4700.

[12] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic conditions. In *Proceedings of the Ninth Symposium on Principles of Database Systems (PODS)*, pages 314–330, Nashville, TN, April 2-4 1990.

[13] Inderpal Singh Mumick and Hamid Pirahesh. Extending the right-linear transformation. Research report, RJ7938, IBM Research Division, Computer Science, Almaden Research Center, San Jose, California 95120-6099, January 22 1991.

[14] Inderpal Singh Mumick and Hamid Pirahesh. Overbound and right-linear queries. In *Proceedings of the Tenth Symposium on Principles of Database Systems (PODS)*, Denver, CO, May 29-31 1991.

[15] Jeffrey F. Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D. Ullman. Arity reduction through factoring. In *Proceedings of the Fifteenth International Conference on Very Large Databases (VLDB)*, Amsterdam, Holland, 1989.

[16] Jeffrey F. Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D. Ullman. Efficient evaluation of right-, left-, and multi-linear rules. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data*, Portland, OR, 1989.

[17] Raghu Ramakrishnan. Magic templates: A spellbinding approach to logic programs. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 140–159, Seattle, WA, August 1988. MIT Press.

[18] Jeffrey D. Ullman. Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, 10(3):289–321, September 1985.

[19] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume 1.* Computer Science Press, 1988.

Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan
Adornments in Database Programs
Reprint of Figure 1 to be inserted on Page 14.

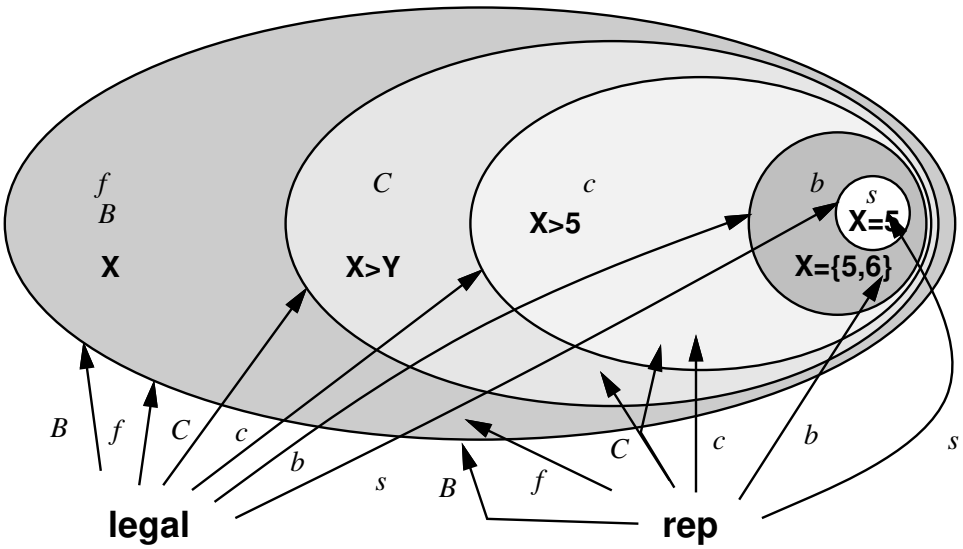

Figure 1: Relationships between legal and rep sets of s, b, c, C, B, and f adornments.

Inderpal Mumick. Reprint of Figure 2, Insert on Page 23.

Algorithm 1 (Find a full sips and adorn a rule.)

Input:

1. A Datalog rule r of the form:
   (r): $p :- q_1$ & q_2 & ... & q_m & c_1 & c_2 & ... & c_n.
   where each q_j is an ordinary subgoal, and each c_i is a condition.
2. A head adornment p^α.

Output:

A full sips, $\mathsf{sips}(r, \alpha) = (V, E, B)$, and an adorned rule ar.

Method:

% *Step 1: Initialize*
$C = \emptyset$. % *Set of conditions*
$\mathcal{B} = \mathsf{bvar}(p_h^\alpha)$; $\mathcal{C} = \mathsf{cvar}(p_h^\alpha)$; % *Set of bound and conditioned variables*
$V = \{q_1, \ldots, q_m\}$; $E = B = \emptyset$; % *Initialize* $\mathsf{sips}(r, \alpha)$
for $(j = 1$ **to** $m)$
{% *Step 2.A: Determine the* j^{th} *subgoal*
$u_j = \mathsf{next}(r, (u_1, \ldots, u_{j-1}), C, \mathcal{B}, \mathcal{C})$; % *Find next subgoal*

% *Step 2.B: Derive* $C(j-1)$*; propagate effect to* C*,* $\mathcal{B}$*, and* $\mathcal{C}$
$C(j-1) = \{c_i \mid \mathsf{eligibility}(c_i) = j-1\}$; % *Conditions eligible at position* $j-1$
$C = C \cup C(j-1)$; % *Conditions eligible at or before position* $j-1$
$\mathcal{B} = \mathsf{bvar}(\{p_h^\alpha, C, u_1, u_2, \ldots, u_{j-1}\})$; % *bound variables before subgoal* u_j
$\mathcal{C} = \mathsf{cvar}(\{p_h^\alpha, C, u_1, u_2, \ldots, u_{j-1}\})$; % *conditioned variables before subgoal* u_j

% *Step 2.C: Extend* $\mathsf{sips}(r, \alpha)$ *to record information passed into* u_j
Insert the edge $e = (\{p_h^\alpha, C, u_1, \ldots, u_{j-1}\} \rightarrow u_j)$ into the edge set E;
Insert the node $\{p_h^\alpha, C, u_1, \ldots, u_{j-1}\}$ into the vertex set V;
$\beta(e) = \mathcal{B} \cap \mathsf{var}(u_j)$; $\kappa(e) = \mathcal{C} \cap \mathsf{var}(u_j)$; % *labels for edge* e
% *Step 2.D: Adorn* u_j
Compute adornment α_j for u_j due to the bound and condition labels $\beta(e)$ and $\kappa(e)$;

% *Step 2.E: Determine* $\mathcal{B}$ *and* $\mathcal{C}$ *for the next subgoal*
$\mathcal{B} = \mathcal{B} \cup \mathsf{var}(u_j)$; % *As* u_j *is evaluated, its variables get bound*
$\mathcal{C} = \mathcal{C} - \mathsf{var}(u_j)$;
}

% *Step 3: Determine conditions evaluable only after the last subgoal*
$C(m) = \{c_i \mid \mathsf{eligibility}(c_i) = m\}$;
The adorned rule ar is:
(ar): $p^\alpha :- C(0)$ & $u_1^{\alpha_1}$ & $C(1)$ & $u_2^{\alpha_2}$ & ... & $C(m-1)$ & $u_m^{\alpha_m}$ & $C(m)$.

◇

Figure 2: A tight and legal adornment algorithm for the *bcf* adornment class.