The Mathematics of Set Predicates in Prolog*

Egon Börger[1] Dean Rosenzweig[2]

[1] Dip. di Informatica, Universita di Pisa
boerger@di.unipi.it
[2] FSB, University of Zagreb
dean@math.hr

Abstract. We provide a logical specification of set predicates *findall* and *bagof* of Prolog. The specification is given in proof theoretic terms, and pertains to any SLD–resolution based language. The order dependent aspects, relevant for languages embodying a sequential proof search strategy (possibly with side effects), can be added in an orthogonal way. The specification also allows us to prove that *bagof* cannot be defined by SLD–resolution alone. We show the correctness, wrt to our specification, of Demoen's definition of *bagof* for Prolog in Prolog. The specification of *bagof* allows us to throw some light on the logical problems with *setof*.

Introduction

The solution collecting predicates *findall, bagof, setof* of Prolog have been quite extensively discussed in the literature— [PerPor 81], [Warren 82], [Ueda 86], [Ueda 87], [O'Keefe 90], [Demoen 91], [Dodd 91], [WG17 92] and can be found, in different versions, in most Prolog systems (DEC-10, C, Quintus, BIM, Sicstus, IBM, LPA,...). Discussion has however mainly been about whether and why they should be used, and whether and how they are eliminable. The discussion never came to the point that these predicates are needed because they express (however imperfectly) *fundamental logical principles*, which have explicitly been with us since Frege. This may be because the predicates are usually explained through examples and defined, if at all, by specific algorithms, and not by a mathematical semantics which could be clearly seen as derived from those principles.

We provide a purely logical semantics of *findall* and *bagof* predicates, based on proof theory of SLD-resolution but independent of any particular proof search strategy. The specification thus pertains to any SLD-resolution based language. We relate the specification of *bagof* to the underlying fundamental mathematical principles of comprehension (abstraction) and parametrization. Some choices, made in (current practice and) the draft standard proposal [WG17 92] for *bagof*, turn out to be best justified by combining proof theoretical and model theoretical considerations. Once the specification is given, we can prove that *bagof* cannot be defined by SLD–resolution alone. We also prove that the algorithm, probably intended by the ISO Prolog standardization committee [WG17 92], and

* in: *Computational Logic and Proof Theory*, G.Gottlob, A.Leitsch, D.Mundici Eds., Springer LNCS 713, 1993, pp.1–13.

expressed by an elegant piece of Prolog code by [Demoen 91], is correct with respect to our specification.

The mathematical crux of the paper is section 2 which provides the logical semantics of *bagof/3*. Section 1 prepares the ground with a logical (order–independent) semantics for *findall/3*, giving also the methodological paradigm. In Section 3, we prove correctness of Demoen's Prolog code for *bagof*. In Section 4 we examine *setof*.

Terminology and notation

The notation for 'the set of all x such that $P(x)$', $\{ x \mid P(x) \}$, is understood as denoting application of *comprehension* (*abstraction,collection*) operator to variable x and expression $P(x)$, which *binds* all occurrences of x within its scope—in very much the same way as other variable–binding operators, such as $\forall, \exists$ in predicate calculus, λ in λ–calculus, $\int_a^b$ in integral calculus... bind all occurrences of a variable in an expression. Variable (occurrences)s which are not bound, in this sense, are *free*. In order to distinguish this *logical* notion of bound variable from the *computing* notion of variable being 'bound to a value', we shall call the latter, in this logic programming context, *instantiated*.

Every textbook on logic explains why bound variables can and must be renamed, to avoid clashes with variables which occur free in the same context. Most notable example in logic programming is probably renaming of a clause to be resolved, since all variables occurring in a clause are tacitly understood as being bound by a universal quantifier.

We shall have to deal with *multisets* (*bags*, 'sets with repetitions'). We adopt the following notation, for 'the bag of all x such that $P(x, i)$, so that, for each such $i \in A$, a copy of x is taken', where A is an ordinary set:

$$\langle\, x \mid P(x,i) \,\rangle_{i \in A}.$$

In such an expression x, i are both bound. For instance

$$\langle\, j \mid i^2 = j \,\rangle_{i \in \{-1,0,1\}} = \langle\, 0, 1, 1 \,\rangle.$$

where the form $\langle\, x_1, \ldots x_n \,\rangle$ will denote bags given by enumeration (obviously, the order of enumeration is irrelevant here). We shall drop the indication of index set, $i \in A$, when it is clear from the context. We use $\vee$ to denote multiset union, thus $\langle\, 1, 2, 1 \,\rangle \vee \langle\, 2, 2 \,\rangle = \langle\, 1, 1, 2, 2, 2 \,\rangle$.

We shall otherwise rely on standard notation and terminology of logic programming, cf. [Apt 90].

1 Semantics of *findall*

The predicate *findall(T, G, L)* has been introduced into Prolog in order to automate the process of finding (through repeated backtracking) and collecting into

a list, all values of the term T with which the goal G succeeds, unifying subsequently this list with L. Can we make some logical sense out of this procedural description, uncoupling its overdependence on Prolog backtracking (and hence on ordering)?

Let t be a term and g a goal. Let $\mathbf{X}$ be the (sequence of) variables occuring in t, and $\mathbf{Y}$ the variables occuring in g but not in t. During the computation of g, both $\mathbf{X}$-variables and $\mathbf{Y}$-variables get instantiated to some values. By *findall(t,g,l)* only the values of the $\mathbf{X}$-variables are collected into the list, while the collecting phase disregards the $\mathbf{Y}$ values. The mathematical idea underlying this collecting process can be, in the first approximation, expressed by comprehension

$$\{\, t(\mathbf{X}) \mid \exists \mathbf{Y}\, g(\mathbf{X}, \mathbf{Y}) \,\}.$$

Note that all variables are bound here, $\mathbf{Y}$'s by quantification, and $\mathbf{X}$'s by comprehension[3]. These bound variables must be treated as nameless dummies, i.e. as distinct from all variables occurring free in the same context—in l or in the calling environment—even if they have the same name. But note that being a free or a bound variable is here a runtime property; as long as $\mathit{findall}(t, g, l)$ is not called, all variable occurrences in t, g, l are instantiated uniformly. The necessity to distinguish bound from free variables, without in general using explicit variable–binding operators with syntactically clearly defined scope, does create some difficulties for logic programming, as we shall see in a more pronounced way in the section on *setof*.

In the context of usual **model theory** of logic programming, an approximate model-theoretic specification for *findall* would then be: compute (a finite representation of) the set

$$S = \{\, t(\mathbf{a}) \mid \exists\, \mathbf{b} \models g(\mathbf{a}, \mathbf{b}) \,\}$$

where $\mathbf{a}, \mathbf{b}$ are understood as ranging over (a finite power of) the Herbrand universe (i.e. sequences of ground terms). This approximate specification disregards both possible repetition of solutions and the order of their appearance.

We cannot do much better with model theory alone, since *findall* inherently involves **proof-theoretic** notions.

> ... although Prolog *reports* solutions, it is *looking for proofs*, and *findall*/3 is defined to return an instance of the t for every *proof* of g. [O'Keefe 90]

For the following proof theoretical analysis we have the assumption that the SLD-tree of g is finite (since otherwise the computation of *findall* will not terminate). In that case, as is well known, for terms $\mathbf{a}, \mathbf{b}$ of the Herbrand universe, $\models g(\mathbf{a}, \mathbf{b})$ iff $\vdash g(\mathbf{a}, \mathbf{b})$.

Let then $\pi_1, \ldots \pi_n$ be all SLD–*proofs* of (the original goal) g, and $\sigma_i = \sigma(\pi_i)$ the corresponding answer substitutions. The requirement, as formulated by O'Keefe, can be expressed by the bag

$$B \;=\; \langle\, t\sigma_i r_i \mid \sigma_i = \sigma(\pi_i),\ \vdash^{\pi_i} g \,\rangle_{i=1,\ldots,n}$$

[3] Remember that $\{\, t(\mathbf{X}) \mid \ldots \}$ is set theoretical shorthand for $\{\, U \mid \exists \mathbf{X}(U = t(\mathbf{X}) \,\&\, \ldots) \,\}$.

where each r_i is a renaming of $t\sigma_i$ by fresh variables (more exactly, we assume the ranges of r_i's to be disjoint, mutually and from the set of all variables present in the calling environment). The renamings may need some explanation. Only the $\mathbf{X}, \mathbf{Y}$ variables, which appear in the original t, g as well as in $t\sigma_i$, really need renaming, since all other variables, which may occur in $t\sigma_i$, are brought in by resolution, and therefore are always fresh. The renaming makes every bag element come with distinct variables. Intuitively this makes them independent—mutually, and from the environment—as they should be, since they come from independent computations. More formally, this is needed to distinguish between free and bound variable occurrences, since our multiset comprehension also binds all variables which occur there.

To sum up, a better approximate specification of *findall* is: compute a representation of B. It refines the first model-theoretic approximation by the obvious:

Proposition 1. For a ground term $t(\mathbf{a})$, $t(\mathbf{a}) \in S$ iff it is a (ground) instance of an element of B.

This makes every element s of B stand for, in classical model theory,

$$\{U \mid \models \exists \mathbf{Z}(U = s(\mathbf{Z}))\},$$

where $\mathbf{Z}$ are all variables occurring in s. The bag represents the union of (sets represented by) its elements. Note that, under this interpretation, all variables in s should be seen as *bound*.

A natural representation of a bag is a list of its elements. Such a representation, however, imposes an ordering on bag elements (as would any other simple representation). If we are to have unique representability, some ordering criterion must then be selected. Since our specification has so far remained order independent, we can adopt any choice whatsoever. Therefore we have the following

Specification of *findall(g,t,l)*. Given an ordering criterion, compute the representation of B in that ordering and unify the resulting list with l. This unifier is the answer substitution.

For languages based on some sequential proof search strategy, the natural choice of ordering is the *solution ordering* (the solutions come in the list in the order in which they appear during the computation). For Prolog this is the usual left-to-right preorder of the tree. For languages with side effects, a proviso should be added: the side effects of proofs π_i appear in the order in which they happen (in those proofs).

2 Semantics of *bagof*

Whereas nobody has problems with understanding or explaining *findall*—this fact is reflected in the straightforwardness of the preceding section—this does

not seem to be the case with *bagof*, as one can see from the discussion in the literature. In fact *bagof(t,g,l)* brings into the logic programming environment the fundamental mathematical operation of *parameterized* comprehension, which can be, in the first approximation, expressed by the comprehension

$$\{\, t(\mathbf{X}) \mid\ g(\mathbf{X}, \mathbf{Y}) \,\}.$$

where now however the variables $\mathbf{Y}$ are free—they are the parameters, and what needs to be represented is not only the collection but also its *dependence* on values of parameters. An approximate model theoretic specification is then: compute (a finite representation of) the set

$$S(\mathbf{b}) = \{\, t(\mathbf{a}) \mid \models g(\mathbf{a}, \mathbf{b}) \,\}$$

in its dependence on parameters $\mathbf{b}$.

A proof-theoretic analysis has to proceed more gradually here, keeping however the assumption of finiteness of the computation tree. If we fix parameter values $\mathbf{b}$, the proof-theoretic approach of the previous section gives us the multiset

$$B(\mathbf{b}) \quad = \quad \langle\, t\sigma_i\rho_i r_i \mid \sigma_i = \sigma(\pi_i),\ \stackrel{\pi_i}{\vdash}\ g\,,\ \mathbf{Y}\sigma_i\rho_i = \mathbf{b} \,\rangle_i$$

where t, π_i, σ_i are defined as in the previous section, ρ_i is the (minimal) ground substitution instantiating $\mathbf{Y}\sigma_i$ to $\mathbf{b}$ (i.e. their mgu). The renamings r_i rename (freshly) only those bound variables, occuring in $t\sigma_i$, which are not instantiated by ρ_i.This means those $\mathbf{X}$'s, that do not occur in $\mathbf{Y}\sigma_i$, i.e. are not linked to parameters by the answer substitutions. No $\mathbf{Y}$'s are renamed here, which reflects their role of *parameters*; their identity must be preserved—across alternative solutions and wrt to the calling environment—formally they are free in the call of *bagof*. As in the case of *findall*, of course, the distinction of free vs. bound is a runtime property.

For reference, let us note that domains and ranges of ρ_i, r_i, as defined above, are pairwise disjoint, thus

Lemma 1. Substitutions ρ_i, r_i, as defined in $B(\mathbf{b})$, commute.

We have the obvious

Proposition 2. A ground term is an element of $S(\mathbf{b})$ iff it is a (ground) instance of some element of $B(\mathbf{b})$.

If our approximate specification is refined to 'computing a finite representation of $B(\mathbf{b})$ in its dependence on $\mathbf{b}$', we are led to the following (tentative) requirements:

Requirement 1. If there are no free variables, *bagof* (t, g, l) behaves exactly as *findall* (t, g, l), given that g has a solution at all (cf. below).

Requirement 2. In general, *bagof* (t, g, l) may have alternative solutions.

Requirement 3. Alternative solutions to $bagof\,(t, g, l)$ should reflect dependence of $B(\mathbf{b})$ on $\mathbf{b}$.

Requirement 1 is due to the observation that, in case of no free variables, we are really talking about B from the previous section. Requirement 2 then follows, since it is easy to concoct examples yielding drastically different bags for different instantiations of g. Requirement 3, however, does have a model-theoretic flavour, which may not be quite appropriate in this proof-theoretic context—what is computed in proofs, namely, are not ground instances of parameters, $\mathbf{b}$, but 'computed parameter values' $\mathbf{Y}\sigma_i$. Hence requirement 3 may be moderated by

Requirement 4. Alternative solutions to $bagof\,(t, g, l)$ should reflect alternative computed parameter values.

From the wording of Requirement 4 it is clear why we had to put an additional condition on Requirement 1: if g had no solutions, Requirement 4 would prohibit us to return an empty bag, since it would not reflect *any* computed parameter value (see also [Warren 82]). Thus,

Requirement 4'. If g has no solution, $bagof\,(t, g, l)$ should fail.

Requirements 3 and 4 should not be taken too literally; they would namely be contradictory. Requirement 3, taken literally, would lead to the following collection procedure: take $B(\mathbf{b})$'s, strip away the ground substitutions ρ_i, and provide the finitely many (up to renaming of nonparametric variables) bags as alternative solutions. Equivalently, we could define alternative results to be (some renamings and unifications of, cf. Definition 3 below)

$$R \;=\; \langle\, t\sigma_i r_i \,\rangle_{i\in T}$$

where T ranges over all *maximal* nonempty index sets such that all computed parameter values $\mathbf{Y}\sigma_i,\ i \in T$ are mutually consistent, i.e. unifying. It is easy to see that every $B(\mathbf{b})$ is, elementwise, an instance of an R.

This would, however, destroy Requirement 4 completely. Same computed parameter values would reappear in different solutions, and it is possible for some *different* (but consistent) values to appear always together (if there is no further alternative to separate them).

How is Requirement 4 then to be understood? If we take 'alternative' computed parameter values literally, as coming from different proofs, then all collecting is lost (as all bags would be singleton), contradicting even Requirement 1. We thus have to abstract from proofs here, and look elsewhere for a criterion of 'being alternative'.

There seems to be (almost) a consensus in the Prolog community about where to look—to model theory. From model theoretic point of view, computed parameter values are different if they have different sets of (ground) instances. It is known [Apt 90] to be equivalent to the following criterion: computed parameter

values $\mathbf{Y}\sigma_i, \mathbf{Y}\sigma_j$ are alternative if they are not variants. The decision, to group into one alternative solution (bag) those proofs which yield variant parameter values, is then expressed by

Definition 1. Answer substitutions σ_i, σ_j are equivalent if $\mathbf{Y}\sigma_i, \mathbf{Y}\sigma_j$ are variants. Denote by Σ the equivalence class of σ.

For later use, let us record

Lemma 2. For a parameter Y

(a) $Y \in \mathit{vars}(\mathbf{Y}\sigma_i) \;\Rightarrow\; Y\sigma_i = Y$
(b) $Y \in \mathit{vars}(\mathbf{Y}\sigma_i) \;\Rightarrow\; Y \in \mathit{vars}(\mathbf{Y}\sigma_j)$
at exactly the same position, for $\sigma_i, \sigma_j \in \Sigma$.

Proof. Statement (a) follows from answer substitutions being idempotent—cf. [Apt 90]. Statement (b) follows from (a), observing that sequences

$$\ldots t(\ldots Y \ldots) \ldots Y \ldots \quad \text{and} \quad \ldots t(\ldots Z \ldots) \ldots Y \ldots$$

cannot be variants, for any Z which is not Y.

Definition 2. $B(\mathbf{b}, \Sigma) \;=\; \langle\, t\sigma_i\rho_i r_i \mid \mathbf{Y}\sigma_i\rho_i = \mathbf{b},\ \sigma_i \in \Sigma \,\rangle_i$

Proposition 3. $B(\mathbf{b}) \;=\; \bigvee\{B(\mathbf{b}, \Sigma) \mid \mathbf{b} \text{ is an instance of } \mathbf{Y}\sigma\}$

A solution to $\mathit{bagof}(t, g, l)$ will then be defined, up to ordering, by

Definition 3. $B(\Sigma) \;=\; \langle\, t\sigma_i r_i r_i'\theta \mid \sigma_i \in \Sigma \,\rangle_i$ where θ is the mgu of $\mathbf{Y}$ and all $\mathbf{Y}\sigma_i r_i'$ with $\sigma_i \in \Sigma$, where r_i' are fresh renamings of those bound variables occuring in $t\sigma_i$ and $\mathbf{Y}\sigma_i$ (and therefore not renamed by r_i).

This unification is necessary to preserve the identity of parameters—uninstantiated variables, distinct from $\mathbf{X}, \mathbf{Y}$ and possibly brought into $\mathbf{Y}\sigma_i$ by SLD-resolution, are all distinct, but should be matched across proofs contributing to the same bag. Thus $\mathbf{Y}\theta$ now provides a cannonical representative of 'computed parameter values'. The ρ_i's of Definition 2 and ρ of Definition 3 are linked by

Lemma 3. Let $\rho_i = \mathit{mgu}(\mathbf{Y}\sigma_i, \mathbf{b}),\ \rho = \mathit{mgu}(\mathbf{Y}\theta, \mathbf{b})$. Then the equation $\rho_i = r_i'\theta\rho$ holds when both sides are restricted to $\mathit{vars}(t\sigma_i) \cup \mathit{vars}(\mathbf{Y}\sigma_i)$.

Proof. A parameter Y not occurring in $\mathbf{Y}\sigma_i$, is necessarily included in the domain of σ_i, and therefore by idempotence cannot occur in its range, and hence not in $t\sigma_i$. Thus the lemma does not claim anything about Y, and the following cases remain to be proved:

Case	$Y \in vars(\mathbf{Y}\sigma_i)$	Then $Yr'_i = Y$ since Y is not in the domain r'_i; $Y\theta = Y$ by Lemma 2 (a) and the definition of θ. Therefore $Y\rho_i = Y\rho = Yr'_i\theta\rho$ by definition of ρ_i, ρ.
Case	$Z \notin vars(\mathbf{Y}\sigma_i)$	Then $Z\rho_i = Z$ by definition (and relevance) of ρ_i; $Zr'_i = Z$ since Z is not in the domain of r'_i; $Z\theta = Z$ by definition (and relevance) of θ; $Z\rho = Z$ by definition (and relevance) of ρ, θ, since $Z \notin \text{vars}(\mathbf{Y}\theta)$.
Case	$Z \in \text{vars}(\mathbf{Y}\sigma_i)$	Then $Zr'_i\theta$ is a variable (because all $\mathbf{Y}\sigma_i$ are variants) occurring, in $\mathbf{Y}\theta$, at exactly the same positions in which Z occurs, in $\mathbf{Y}\sigma_i$. Then the claim $Z\rho_i = Zr'_i\theta\rho$ follows by definition of ρ_i, ρ.

$B(\Sigma)$ covers exactly all the $B(\mathbf{b}, \Sigma)$ via instantiations of parameter values to ground terms, as shown by

Proposition 4. Each $B(\mathbf{b}, \Sigma)$ is identical to $B(\Sigma)\rho$ for $\rho = \text{mgu}(\mathbf{Y}\theta\,\mathbf{b})$.

Corollary. For each substitution τ such that $\mathbf{Y}\theta\tau$ is ground, $B(\Sigma)\tau$ is an instance of $B(\mathbf{Y}\theta\tau, \Sigma)$.

Proof. The equations

$$\begin{aligned} t\sigma_i\rho_i r_i &= t\sigma_i r_i\rho_i &&= t\sigma_i r_i r'_i\theta\rho \\ \mathbf{Y}\sigma_i\rho_i &= \mathbf{Y}\sigma_i r'_i\theta\rho = &&\mathbf{Y}\theta\rho \end{aligned}$$

follow, respectively, from Lemma 1, Lemma 3, Lemma 3 and the definition of θ.

In a call of *bagof*(t, g, l), g is usually allowed to be a *quantified goal*, i.e. a form $Z_1\hat{}\ldots Z_n\hat{}g_1$, where g_1 is a goal, understanding the variables $Z_1, \ldots, Z_n$ to be *existentially quantified*. That would, in a first model theoretical formulation, mean $S = \{\, t \mid \exists Z_1 \ldots Z_n\, g_1 \,\}$. The simple remark, that quantified variables are *bound*, suffices to make our treatment verbatim correct for quantified goals too.

The logical specification, resulting from the above decision, is then:

Specification of *bagof*(t, g, l)**.** Given an ordering criterion, for (alternative) solution corresponding to equivalence class Σ, compute the representation of $B(\Sigma)$ and unify it with $l\theta$. This unifier, composed with θ, is the answer substitution.

For languages based on some sequential proof search strategy, we must specify the *ordering of appearance* of alternative solutions—the natural choice is the solution ordering of first elements of their representations. For languages with side effects we must specify also the side effects of *bagof*(t, g, l). In case of Prolog,

the decision of ISO WG17 seems to be 'follow the usual implementations', i.e. execute all side effects of $\mathit{findall}(t, g, l)$, *before* reporting any solutions of *bagof* (cf. also Section 3).

Propositions 3 and 4 might help explain some of the usual difficulties in understanding *bagof*. It namely violates the *lifting property* of SLD-resolution [Apt 90]: a solution of an instance of a goal is an instance of a solution of that goal. Here, as shown, the solution $B(\mathbf{b})$ (given that g itself has the lifting property) for an instance of g may only be patched together from instances $B(\mathbf{b}, \Sigma)$ of alternative solutions for g. For instance, if predicate g, checking whether at least two of its three arguments unify, was defined by clauses

$$g(E, E, A).\ g(E, A, E).\ g(A, E, E).$$

a call of $\mathit{bagof}(1, g(Y_1, Y_2, Y_3), L)$ would yield three alternative solutions, each of them unifying L with $[1]$, whereas a call of $\mathit{bagof}(1, g(Y, Y, Y), L)$ would have only one solution, unifying L with $[1, 1, 1]$. Given the lifting property of SLD–resolution, we have

Proposition 5. Given any ordering criterion, a predicate satisfying the above specification of *bagof* cannot be defined by SLD–resolution alone.

This proposition confirms that some interleaving of model-theoretical and proof-theoretical arguments, in deriving a logical description of *bagof*, was inevitable. It also makes visible that the notion of *declarative* semantics, if we are to analyze real phenomena such as *bagof*, cannot be understood in a very narrow sense (say only in terms of simple fixpoint constructions like classical T_P)[4]

3 Correctness of Demoen's specification of *bagof*

In case of Prolog, several descriptions of *bagof* have been put forward in the context of the standardization effort in ISO WG17 [Demoen 91, Dodd 91, WG17 92]. The only specification which, to us, seems to be clear and precise enough to be related to our specification by a *proof*, is due to Bart Demoen, and comes in the form of the following elegant piece of Prolog code [Demoen 91].

$$\begin{array}{l} \mathit{bagof}(\mathit{Term}, \mathit{Goal}, \mathit{Bag}) :- \\ \quad \mathit{free\_variables}(\mathit{Goal}, \mathit{Term}, \mathit{Vars}), \\ \quad \mathit{findall}(\mathit{Vars} - \mathit{Term}, \mathit{Goal}, \mathit{Answerlist}), \\ \quad \mathit{produce}(\mathit{Answerlist}, \mathit{Answer}, \mathit{Vars}), \\ \quad \mathit{Bag} = \mathit{Answer}. \end{array}$$

[4] Proposition 5. applies, strictly speaking, to *findall* as well, but, as shown by the above example, 'less strikingly so'.

```
produce([Params − Term | Rest], Bag, Vars) :−
        split(Rest, Params, Terms, Bags),
        (Bag = [Term | Terms], Vars = Params
        ;
        produce(Bags, Bag, Vars)).


split([], _, [], []).
split([Params − Term | Rest], Params1, [Term | Terms], Bags) :−
        variants(Params, Params1),
        !,
        split(Rest, Params1, Terms, Bags).
split([Term | Terms], Params, Bag, [Term | Bags]) :−
        split(Terms, Params, Bag, Bags).
```

We have not listed the code for predicates *free_variables, variants* - of them we shall assume the following.

(i) *free_variables*$(Goal, Term, Vars)$ unifies $Vars$ with the list of all variables free in $Goal$ wrt $Term$, i.e. of those unquantified variables in $Goal$ which do not occur in $Term$;

(ii) *variants*(X, Y) fails if X and Y are not variants, and unifies them together otherwise.

Note that, in the algorithm, the free variables are detected only at runtime, after *bagof* has been called, as they should be. Since implementations of Prolog usually do not provide the *occur–check*, semantic reasoning about Prolog programs usually applies only to situations satisfying the following additional general assumption:

(iii) All unifications executed are not subject to the occur–check.

In view of the fact that the draft Prolog standard proposal [WG17 92] *does not* specify behaviour of systems when this assumption is violated, there is little that can be said in that case. In particular, since most implementations produce idempotent and relevant mgu's as soon as assumption (iii) holds, we will in this section rely on that. Under assumptions (i), (ii), (iii) we have

Proposition 5. Demoen's Prolog code is correct wrt to specification of *bagof* in Section 2.

Proof. We have to prove that, on alternative backtracking calls, the algorithm computes the representation of each $B(\Sigma)$ and unifies it with Bag. Given (the code under) usual operational understanding of Prolog, i.e of the way it searches the SLD–tree, following remarks are true:

(a) The only point where the SLD–tree can branch into alternative solutions (*choicepoint*, in WAM jargon) is the one indicated by the semicolon in code for *produce*—backtracking will use it only after a solution has been *produce*d;

(b)'Witnesses' *Vars* (generated by *free_variables*, cf. assumption (i)) represent the $\mathbf{Y}\sigma_i$'s. In case that some Y remains uninstantiated by (some iff all, cf. Lemma 2) σ_i's, it will, in this algorithm, get renamed by *findall*. To see this does not affect the computed $B(\Sigma)$, consider definition 3 with the renamings r'_i extended so as to rename such uninstantiated Y's as well. θ however undoes this extra renaming by unifying all such Yr'_i together and with the original Y's.

(c) θ is created incrementally, propagating unification along the list as it is being *split* by *variants* (cf. assumption (ii)), unifying $\mathbf{Y}\sigma_i$'s together, and completed after a solution is *produce*d by explicit unification $Bag = Answer$, unifying them with $\mathbf{Y}$.

Then a simple induction over the size of $B(\Sigma)$ proves that the algortihm finds the (next) $B(\Sigma)$ in the right ordering, and an equally simple induction over the number of Σ's (solutions) shows that it finds them all.

For the reader who finds this proof to be handwaving, and requires a 'more formal' argument, we would have to substitute 'the usual operational understanding' of Prolog (i.e. of the way it searches the SLD–tree) with a mathematical model. Our tree model [BoeRos 91, BoeRos 92] adheres so closely to this 'usual operational understanding' that a transfer of the preceding proof to the mathematical model is nothing but an excercise, which we may leave to the interested reader. The primary purpose of operational semantics was, after all, precisely to provide operational arguments of this kind with some certainty and dignity of mathematics.

4 Analysis of *setof*

From the set theoretical point of view, a bag is just a redundant representation of a set. The predicate *setof* is thus usually explained as being the same as *bagof*, with

(a) removing duplicates from solutions;

(b) sorting the solutions, providing unique list representation.

Both (a) and (b) are simple and well justified in case of ground terms. In case of uninstantiated variables in terms however, both (a) and (b) require definition.

Trying to find a common demoninator of current practice, ISO WG17 has decided [WG17 92] to interprete *duplicates* as *identical terms* in sense of Prolog predicate == /2. However obvious this choice may seem, in case of not fully instantiated terms it is impossible to justify logically, as sensed also by O'Keefe:

> (*setof* is) ... only sound when the free variables and template variables are bound to sufficiently instantiated terms ... [O'Keefe 90]

Consider a database with facts

$$p.\ p.\ q(Z,Z).\ q(Z,Z).$$

The calls $setof(1,p,L_1)$, $setof(X,p,L_2)$, $setof(X,q(X,Y),L_3)$ provide, under this semantics, the solutions $L_1 = [1]$, $L_2 = [X',X'']$, $L_3 = [Y]$. L_2 has two elements only due to renaming of bound variables (by r_i of section 2)—X', X'' are

just placeholders, nameless dummies which cannot occur anywhere else. In light of model theoretic discussion of Section 1, X' stands here for

$$\{U \mid \models \exists X'(U = X')\},$$

but then X', X'' stand for the same thing (even the same expression, cf. below). The fallacy of collecting them here as distinct objects may originate from the fact that the binding comprehension operator is not visible any more. It is notorious in logic that no meaningful distinction can be made between expressions which differ only wrt the names of their bound variables—they are considered as being *syntactically identical*. For instance, $\int_0^1 x^2\,dx = \int_0^1 y^2\,dy$ is *not* a theorem of the integral calculus—the two sides are simply the same expression. For systematic discussion in the context of λ–calculus, cf. [Barendregt 84].

To define duplicates by $== /2$ amounts to collecting sets of *names*. Even without drawing on the vast literature on the perils of confusing naming and meaning, mention and use, note that collecting names is surely not what the user is led to think of when writing a *setof* expression.

From the logical point of view then the terms being collected by *bagof* should at least be considered as duplicates when they are variants, modulo (variables linked by answer substitution to) parameters.

From the (classical) model theoretic point of view it would even be natural to consider as redundant also instances of terms which already exist in the representation—they add *only* (ground) elements which are already represented.

In both cases it is simple to adapt our specification of *bagof*, to yield an appropriate *setof*, without complicating the implementation excessively.

As to sorting criterion, it is really the fixation on *names* which must have led to *term ordering* as a supposedly natural choice [WG17 92]—even though different implementations cannot be reasonably expected to agree on ordering of uninstantiated variables. In some cases the ordering is thus left undefined, making it implementation dependent and not portable. In addition, term ordering is not preserved by instantiation, which is another, this time unnecessary, violation of the lifting property.

References

[Apt 90] K.R.Apt, Logic Programming, in: J. van Leeuwen (ed.), *Formal Models and Semantics*. Handbook of Theoretical Computer Science, Vol.B, Elsevier 1990, pp. 493–574

[Barendregt 84] H.P.Barendregt, *The Lambda Calculus*, Elsevier 1984

[BoeRos 91] E.Börger, D.Rosenzweig, A Formal Specification of Prolog by Tree Algebras, in: V.Čerić et.al. (eds.), *Proceedings of The Third International Conference on Information Technology Interfaces*, SRCE, Zagreb 1991, pp. 513–518

[BoeRos 92] E.Börger, D.Rosenzweig, The WAM—Definition and Compiler Correctness, Technical report TR–14/92, Dipartimento di Informatica, Università di Pisa 1992

[Demoen 91] B.Demoen, Code and Comments Regarding *bagof/3*, in: *PROLOG. Paris papers 2*, ISO/IEC JTC1 SC22 WG17 N.80, pp. 85-86

[Dodd 91] A.Dodd, The Predicates *bagof/3* and *setof/3*, a Proposal, in: *PROLOG. Paris papers 2*, ISO/IEC JTC1 SC22 WG17 N.80, pp. 75–84

[O'Keefe 90] R.A.O'Keefe, *The Craft of Prolog*, MIT Press 1990

[PerPor 81] L.M.Pereira, A.Porto, All Solutions, in: *Logic Programming Newsletter* 2, 1981, pp. 9-10

[Ueda 86] K.Ueda, Making Exhaustive Search Programs Deterministic, in: *Proceedings of the 3rd International Conference on Logic Programming*, pp. 270–282

[Ueda 87] K.Ueda, Making Exhaustive Search Programs Deterministic, Part 2, in: *Proceedings of the 4th International Conference on Logic Programming*, pp. 356–375

[Warren 82] D.H.D.Warren, Higher Order Extensions to Prolog: Are they Needed?, in: *Machine Intelligence* 10(1982), pp. 441–454

[WG17 92] *PROLOG. Part 1, General Core, Committee Draft 1.0*, ISO/IEC JTC1 SC22 WG17 N.92

This article was processed using the LaTeX macro package with LLNCS style