Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

Recursive Queries Evaluation: A Constraint Based Top-down/Botton-up Method

P. Asirelli, P. Inverardi, V. Raffaelli

Nota Interna B4-19
Maggio 1990

Recursive Queries Evaluation: A Constraint Based top-down/bottom-up Method

P. Asirelli, P. Inverardi, V. Raffaelli

Istituto di Elaborazione dell'Informazione CNR, via S. Maria n.46, I-56100 PISA

DRAFT

Abstract

The paper presents an evaluation method for recursive queries based on an integration of the bottom-up and the top-down evaluation strategies.

The proposal tries to eliminate the major drawbacks of the top-down Prolog-like evaluation strategies. Namely, repetition of identical accesses to the EDB and the exploration of resolution branches that lead to the same answer substitution. Forward reasoning (bottom-up phase) is performed by keeping memory of (some) demonstrated "lemmas" while the introduction of constraints avoids visiting already inspected portions of the search space when backtracking (top-down phase).

1. Introduction

The paper presents an evaluation method for DATALOG recursive queries based on an integration of the bottom-up and the top-down evaluation strategies.

The main goal of the proposal is to eliminate, in a satisfactory manner, the major drawbacks of recursive query evaluation methods, that are based on a top-down Prolog-like evaluation strategy. Such drawbacks can be summarized as follows:

1) Many, repeatedly executed, identical accesses to the database. In particular, this can be made worst when, in the goal being evaluated, one, or more, base predicates are "independent", i.e. they share no variables; for those predicates, accesses to the database will be repeated, identically, for each deduction that will be generated.

2) Many, distinct, resolution branches that lead to the same answer substitution. These resolutions are redundant, since the query evaluation process is looking for all distinct answers, not for all possible ways of finding the answers. This problem can also increase 1) above.

The general idea of the approach can be summarized as follows:

Given an IDB, P, an EDB, E, and a query-goal Q, in order to reduce the above mentioned two drawbacks, we identify and use a *set of potentially relevant facts* ϕ [Bancilhon&Ramakrishnan '86] to answer the query. Let us now suppose that ϕ is given, as a generical set of ground atoms enclosed in the least Herbrand model for $P \cup E$.

We consider P∪E as the logic program and Q as the goal. Then we define a modification of the SLD-tree definition [Lloyd '87] that represents the given search space, and that it is built by using as input clauses both the clauses in ϕ and those in P∪E. More precisely, we will consider first ϕ and, only afterwards, consider P∪E, introducing *constraints* to preserve memory of (and avoid to repeat) what has been already done.

We have chosen to call the new tree *Mixed-SLD-tree* because of the, mixed, nature of the use we make of P, E and ϕ.

In section 2 an informal overview of the method is presented. Section 3 is devoted to (boring) definitions, section 4 describes what a mixed SLD-tree is; in section 5 the mixed search strategy is presented together with an example, section 6 concludes. In the Appendix a more complex example is given.

2. Overview of the method

As we have already said, one of the ideas the method we propose is based on is that of having a set of relevant facts ϕ to be used for the bottom-up steps. However, finding out this set in such a way that it is *minimal (or sufficient)* is, in terms of accesses to the database, as expensive as answering to Q. Thus, we have chosen a compromise: finding a subset, ϕ, of a sufficient set of relevant facts R, or, at least, a super*sub*set of ϕ, i.e. a set that includes a part of R plus a set of irrelevant facts, as fewer as possible. The real nature of the set ϕ will be described later on. By means of ϕ, the drawbacks (1-2 in the introduction) can be greatly reduced. That is, the idea is to try, until it is possible, to proceed forward from ϕ. Then, re-start the backward computation when:

i) ϕ does not provide sufficient information to end the computation, or else,

ii) when it is possible to recognize that it is necessary to look for an other deduction from P∪E not locked by constraints, i.e. a deduction that has not been tried forwardly yet.

Because of this bottom-up/top-down way of proceeding in the computation we will call the computation steps from ϕ, as *bottom-up* steps, while we call the others *top-down.*

In the following Example, we give an informal description of the method:

Example.1

IDB: r_1: A(x, y):-B1(x, z, w), A(z, x), B2(y, x).
r_2: A(x, y):-B2(x, y).

query-goal: A(a, y)?

EDB: $B1(a, b_1, c_1), B1(a, b_2, c_2), \ldots, B1(a, b_n, c_n), B2(d_1, a), \ldots, B2(d_m, a)$

$\phi = \{A(b_1, a), \ldots, A(b_k, a), B1(a, b_1, c_1), \ldots, B1(a, b_k, c_k)\}$, $k<n$.

We use conjunctions of facts in ϕ as input clauses, if possible; otherwise, we go top-down as a Prolog-like strategy (see Figure 1).

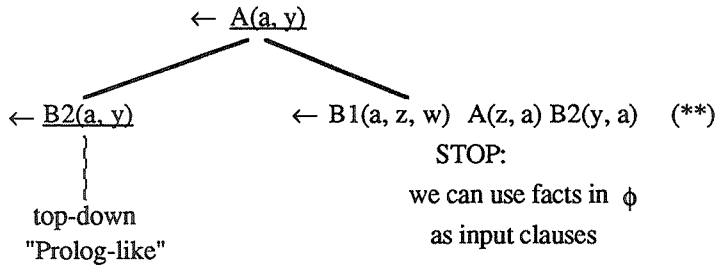


Figure 1

Using facts B1(a, b_i, c_i) and A(b_i, a), $1 \leq i \leq k$, we get k identical sub-computations from the subgoal ←B2(y, a) obtained from (**). These computations generate m answers $y = d_1, \ldots,$ $y = d_m$ to the query-goal. Note that, because these sub-computations are identical, we might execute only one of them.

After executing the sub-computations, we consider again P∪E, where P is the IDB defining A and E is the EDB (i. e. B1 and B2). In order to be able to avoid repeating the already executed computations (starting from ϕ), we rewrite the node (**) in Figure 1 inserting some constraint according to the information acquired executing the subcomputations (see the resulting node in Figure 2, where the rewriting step is indicated by a double arc).

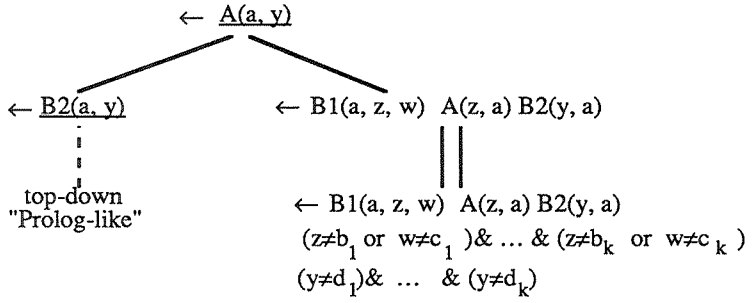


Figure 2

This allows us to avoid:

i) repeating all deductions associated to the substitution $[z/b_1, w/c_1], \ldots, [z/b_k, w/c_k]$;

ii) repeating redundant resolutions for the answers $y=d_1, \ldots, y=d_m$ (we will return on this point in the following). ♦

The crux point of the method is, of course, the way the set ϕ is chosen. We will see, in the description of the search strategy, how to conveniently specify ϕ.

3. Preliminary definitions

In the following we are going to define constraints and clarify their relations with substitutions.

We assume the usual definition of substitution, composition of two substitutions and instance Eθ obtained by applying the substitution θ to an expression E [Lloyd '87].

Definition 1 (constraint) Let $I=\{x_1,\ldots, x_n\}$, $n\geq 1$, be a set of variables,

$v = v_1 \& v_2 \& \ldots \& v_k$ is a *constraint on I* iff each v_i, $1\leq i\leq k$, is such that:

– $v_i = (x_j \neq a)$, where $x_j \in I$ and a is a constant, **or**

– $v_i = (x_{j1}\neq a_1$ or ... or $x_{js}\neq a_s)$, where $x_{jp}\in I$, $1\leq p\leq s$, and $a_1, \ldots, a_s$ are constants, **or**

– v_i = false, **or**

– v_i = true. ♦

From now on, we suppose that constraints are always simplified according to the tautologies of first order logic.

Definition 2 Let $v = v_1 \& \ldots \& v_k$, $k\geq 1$, be a constraint on a set of variables I and θ be a substitution on the set of variables $J = \{x_1, \ldots, x_n\}$, $I \supseteq J$, then *v* is the constraint v simplified by θ*, if v* is derived from v, repeatedly applying the following rules:

1. if there is a substitution $[x_j/a]$ in θ, where a is a constant, and there exists a v_i in v s.t.:

i) $v_i = (x_j\neq b$ or $y_1\neq c_1$ or ... or $y_s\neq c_s)$, $1\leq i\leq k$, $k>1$, $s\geq 0$, with b, $c_1, \ldots, c_s$ constants, then v is replaced by: $v_1 \& \ldots \& v_{i-1} \& v_{i+1} \& \ldots \& v_k$; if k=1, then v* = true.

ii) $v_i = (x_j\neq a$ or $y_1\neq c_1$ or ... or $y_s\neq c_s)$, $1\leq i\leq k$, $k\geq 1$, $s\geq 1$, with $c_1, \ldots, c_s$ constants, then v is replaced by: $v_1 \& \ldots \& v_{i-1} \& (y_1\neq c_1$ or ... or $y_s\neq c_s) \& v_{i+1} \& \ldots \& v_k$; if s=0 (i. e. $v_i = (x_j\neq a)$), then v* = false.

2. if there is the substitution $[x_j/y]$ in θ, where y is a variable, and there exists a v_i in v s.t.

$v_i = (x_j\neq a$ or $y_1\neq c_1$ or ... or $y_s\neq c_s)$, $1\leq i\leq k$, $k\geq 1$, $s\geq 0$, with a, $c_1, \ldots, c_s$ constants, then v is replaced by: $v_1 \& \ldots \& v_{i-1} \& (y\neq a$ or $y_1\neq c_1$ or ... or $y_s\neq c_s) \& v_{i+1} \& \ldots \& v_k$. ♦

Definition 3 Let $\theta = [x_1/a_1, \ldots, x_n/a_n]$ be a substitution, then $v = (x_1\neq a_1$ or ... or $x_n\neq a_n)$ is the *constraint derived from the substitution θ*. If $\theta = \varepsilon$ (i. e.the empty substitution), then v = false. ♦

Since the mixed-SLD-tree that we are going to define will contain constraints as above defined, it is necessary to extend the definition of the unification algorithm and all notions related to it (such as: $C\theta$ istance of a pair C by the substitution θ, unifier, mgu, disagreement set), so that unification can be applied to pairs: <simple expression, constraint>. The extension is defined in [Raffaelli '90] and presents analogous characteristics to the unification given by [Chan '88] for the treatment of non ground negative information.

The following definition introduces a partitions rule that permits to select one or more subset of elements of ϕ that can be unified with a given goal, of course, taking into account constraints, as required by the new unification algorithm. The result of this rule is a set of pairs <list-of-element of the goal, list-of-element of ϕ>. Thus, this partitions rule, for bottom-up steps, plays the role of the selection rule, generally defined for top-down steps.

Definition 4 (Partitions Rule) Let S be a set of triples (g, v, ϕ), where g is a goal, v a

constraint and ϕ a set of ground atoms. Moreover let I be a set of pairs C:(c_1, c_2), where c_1 and c_2 are lists of atoms of the same length and such that the atoms in c_1 are in g and the ones in c_2 are in ϕ. Thus, each pair identifies a bi-partition of all the atoms in g: the ones in c_1, and the ones in g but not in c_1 (if any).

We say that R* is a *partitions rule* if it is a function from S to 2^I such that the value of the function for a triple (g, v, ϕ) is a set of pairs in I $\{C_1, \ldots, C_n\}$, $n \geq 0$. Moreover, for each pair C_i = (c_{i1}, c_{i2}), there exists a substitution θ_i which results from the composition of k mgu $\sigma_1, \ldots, \sigma_k$, k = $|c_{i1}| = |c_{i2}|$, such that σ_j is a unifier for ($c_{i1\text{-}j}$, v) and ($c_{i2\text{-}j}$, true), where $c_{i1\text{-}j}$ ($c_{i2\text{-}j}$) is the j-atom in c_{i1} (c_{i2}, respectively). Finally, θ_i is such that ($Cong_{i1}\theta_i$) = ($Cong_{i2}\theta_i$) where $Cong_{i1}$ ($Cong_{i2}$) is the conjunction of all the atoms in c_{i1} (c_{i2}, respectively). ♦

4. Mixed SLD-tree.

We are now going to define the mixed-SLD-tree. The definition is given recursively since we have chosen to associate a new sub-tree to each possible node obtained by a bottom-up step using ϕ. This seems to be a suitable choice since it allows us to distinguish between nodes obtained by an SLD-like procedure, from those obtained by using the atoms contained in ϕ.

Definition 5 Let P be a set of clauses (the IDB), E a set of facts (the EDB), G_i a pair (g_i, v_i), where g_i is a goal and v_i is a constraint. Let Θ be the set of answer substitutions for $P \cup E \cup \{g_i\}$. Then, the *set of answers for G_i*, given P and E, is the set

$\Theta^* = \{\theta \,/\, \theta \in \Theta$ and, if v_θ is the constraint derived from θ, then $(v_\theta \,\&\, v_i) \neq$ false$\}$

(i. e. we ignore the answer substitutions that violates the constraint v_i). ♦

Definition 6 (Mixed-SLD-tree) Let P be a set of clauses (the IDB), E a set of facts (the EDB), Q the query-goal and v a constraint. Moreover, let ϕ be a set of ground atoms enclosed in the least Herbrand model for $P \cup E$, R a selection rule and R* a partitions rule. Then a *Mixed-SLD-tree* (with identifier id and root (Q, v)), derived via R, R* and ϕ, is recursively defined as follows:

a) Each node in the id tree is a pair (goal, constraint) and has a unique identifier G. Both the elements of the pair can be empty (where an empty goal is denoted by $\square$ and the empty constraint is the constraint true).

b) The root of the id tree is a pair (goal, v). We denote by I(id) the set of answers for the pair (goal, v), given P and E (see Definition 5).

c) Each node ($\square$, true) has no descendants.

d) Let G: ($\leftarrow(A_1, \ldots, A_m, \ldots, A_k)$, v), $k \geq 1$, be a node in id, such that the result of applying R* is the empty set, we apply R which selects an atom A_m, then:

i) if A_m is not a base atom, then G has a descendant for each clause in P, $B \leftarrow B_1, \ldots, B_q$, such that ($A_m$, v) and (B, true) have an m.g.u. θ. The descendants are as follows: ($\leftarrow(A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)$, v)$\theta$.

ii) if A_m is a base atom, then G has a descendant for each fact in E, B_j, such that (A_m, v) and (B_j, true) have an m.g.u. θ_j. The descendants are as follows: $(\leftarrow(A_1,\ldots, A_{m-1}, A_{m+1},\ldots, A_k), v)\theta_j$

e) Let G: $(\leftarrow(A_1, \ldots, A_k), v)$, $k\geq 1$, be a node in id, such that, applying R* to $(\leftarrow(A_1, \ldots, A_k), v, \phi)$ we get the pairs $C_1 = (c_{11}, c_{12}), \ldots, C_n = (c_{n1}, c_{n2})$, $n\geq 1$. Moreover, let $\theta_1, \ldots, \theta_n$ be n substitutions, such that each θ_i is the composition of the m.g.u.*s* for the elements in c_{i1} and c_{i2} (see the R* definition). Then G has a single descendant, the node $G^* = (\leftarrow(A_1, \ldots, A_k), v^*)$, computed as follows:

Let us consider the n Mixed SLD *sub*-trees whose identifier is a function *f*(id, G, θi), of the identifier of the father-tree, id, of the identifier of the node G, and of the substitution θ_i, $1\leq i\leq n$, that generated the root of that subtree. Thus the n trees have the following identifiers *f*(id, G, θ_1), ..., *f*(id, G, θ_n) and roots $(\text{goal}_1, v)\theta_1, \ldots, (\text{goal}_n, v)\theta_n$, respectively, where goal_i, $1\leq i\leq n$, is the goal $\leftarrow(A_1, \ldots, A_k)$ where the atoms in c_{i1} (and unified with those in c_{i2}), i=1,...n, have been respectively eliminated.

Let us now consider, for each tree *f*(id, G, θ_i), $1\leq i\leq n$, the set I(*f*(id, G, θ_i)) of answers for the pair $(\text{goal}_i, v)\theta_i$ (the root of the tree), given P and E.

Then $v^* = v \,\&\, v_\theta \,\&\, v_{gen}$, where:

– v_θ is the conjunction of constraints $v_{\theta 1}, \ldots, v_{\theta p}, \ldots, v_{\theta n}$, for $v_{\theta p}$ constraint obtained by substitutions θ_p, $1\leq p\leq n$ on the variable of G;

– v_{gen} is the conjunction of constraints $v_{gen\text{-}1}, \ldots, v_{gen\text{-}p}, \ldots, v_{gen\text{-}n}$, where

$v_{gen\text{-}p} = h(\theta_p, I(f(\text{id}, G, \theta_i))$ and *h* is a function of the answers computed by the i-th sub-tree. The constraint $v_{gen\text{-}p}$ is such that those answers will not be computed anymore.

♦

Definition 5, refers to the usual notion of answer substitution and it has been given to clarify the nature of constraints that are introduced in the nodes of the mixed-SLD-tree. In fact, for every node G, whenever it is possible to go bottom-up, by means of φ, we generate n sub-trees for each one of the n distinct unification with elements of φ (see Figure 3). Then it is necessary to give a new constraint that is a function of the possible answers computed by those n sub-trees. The new constraint has to be such that those answers will not be computed anymore.

The n nodes generated by a bottom-up step will become the roots of n new sub-trees, and each root has associated the set of possible answer substitutions as defined in Definition 5. Given such set of answer substituions, the constraint to be associated to the only descendant of G, the node G*, can be computed. In fact, the node G* is identical to G but it has a constraint v* that is able to exclude the answers already computed by the n bottom-up sub-trees obtained from G.

In particular, given G and its contraint v, the constraint v* of G* is given in terms of the

variables in G and it is such that:

a) it preserves conditions already stated for G (**v**);

b) it excludes repetitions of substitutions that have generated the n bottom-up computations (v_θ);

c) it exlcudes repetition of the answers substitutions generated by the n bottom-up computations (v_{gen})

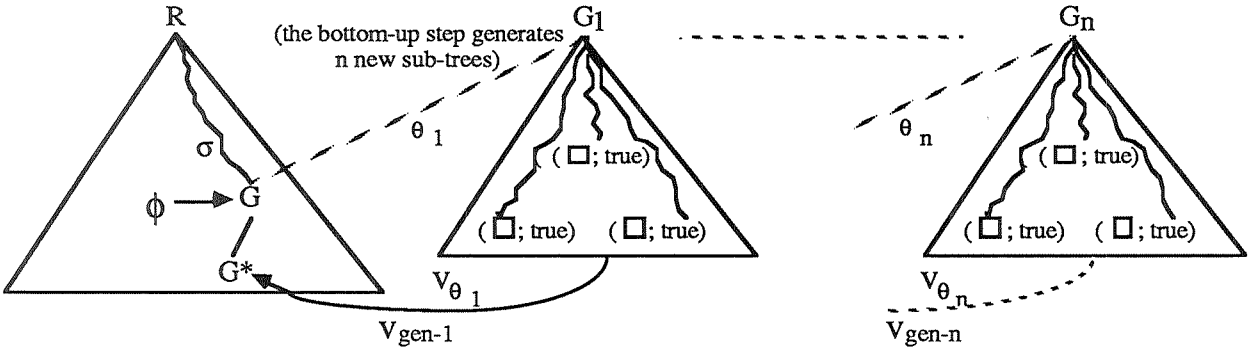


Figure 3

Similarly, according to the usual definition of refutation and of answer substitutions, *mixed-SLD-refutation* and *mixed-SLD computed answer* can be defined, as well.

Next, we give the two theorems that prove soundeness and completeness of the mixed SLD resolution. They are, of course, based on the same results for SLD resolution. Furthermore, given the generality of selection rules R and R*, the second theorem can also be used to prove strong completeness of mixed-SLD. In the paper we only give a sketch of proofs that are instead completely given in [Raffaelli '90].

Teorema.1 (Soundeness of Mixed-SLD Resolution): Let P be a set of clauses (IDB), E a set of facts (EDB), G a pair (goal, vincolo), ϕ a set of ground atoms contained in the minimal Herbrand model of P$\cup$E, R a selection rule and R* a partitions rule. Every mixed-answer substitution for G, that is obtained in P$\cup$E by means of R, R* and ϕ, is correct.

Sketch of the proof: By induction on the refutation level to compute the mixed substitution. To each mixed answer substitution computed by a mixed SLD refutation for the pair G, it is possible to find a corresponding SLD refutation, for the goal in G, that computes the same answer substitution, thus it is correct, for the soundness of SLD. ♦

Teorema.2 (Completeness of Mixed SLD-Resolution): Let P be a set of clauses (IDB), E a set of facts (EDB), G a pair (goal, vincolo), ϕ a set of ground atoms contained in the minimal Herbrand model of P$\cup$E. For every correct answer substitution θ for G in P$\cup$E, there exists a selection rule R, a partitions rule R*, a mixed-answer substitution σ obtained by means of R, R* and ϕ for G in P$\cup$E and a substitution γ such that $\theta = \sigma\gamma$.

Sketch of the proof: Straightforwardly by contradiction. ♦

5. Mixed SLD-Strategy

In this section we describe the search strategy that allows to collect all solutions associated to a given mixed-SLD tree. In particular, we will define a selection rule R, a partitions rule R*, the set ϕ and the search rule.

5.1 Selection and Partitions rules

Selection Rule

The selection rule we have adopted tries to privilege the most instanciated atoms in the goal. In this way it is possible to sensibly reduce the number of accesses to the base. To this respect the usual Prolog-like leftmost selection rule, as pointed out in [Henschen&Naqvi '84, Ullman '85, Vieille '89], is inadequate.

Thus we introduce a measure of the instantiation degree of an atom.

Definition 7 Let A be an atom. Then, the *instantiation degree* of A is the quotient (n/k) where:

– n is the number of constant arguments of A;

– k is the total number of arguments of A.

The instantiation degree of an atom without constant arguments is equal to zero. ♦

Definition 8 Let $I=\{A_1, \ldots, A_n\}$ be a set of atoms, *A_i is the most instantiated* atom if it has the maximum instantiation degree (note that there can be more than one most instantiated atom). ♦

Selection Rule R

R: Given a goal g, select the most instantiated base atom, if any, otherwise the most instanciated derived one. If no atom is instanciated select a base atom, if any, otherwise a derived one. If more than one most instanciated atom exists select the leftmost one. ♦

Partitions Rule

In order to determine the partitions rule R*, criteria similar to the above discussed ones have been followed. In particular, the idea is to let R* to select those nodes whose goals have minimal (in terms of residual atoms) *dimension* and are extensively instantiated.

Definition 9 Let $I=\{A_1, \ldots, A_n\}$ be a set of atoms. Then, the *instantiation degree of I* is the quotient $(\Sigma_{i=1..n}\, r_i)/n$ where r_i, $1 \le i \le n$, is the *instantiation degree* of A_i. ♦

Definition 10 Let $I=\{g_1, \ldots, g_n\}$ be a set of goals, then *g_i is the most instantiated* goal if the set of atoms in g_i has maximal instantiation degree with respect to the set of atoms in the other goals (note that there can be more than one most instantiated goal). ♦

Before introducing the partitions rule we have adopted, it is worthwhile to recall that a partitions

rule is a function from a set of triples (g, v, ϕ), where g is a goal, v a constraints and ϕ a set of ground terms, to the multiset 2^I where I is a set of pairs (c_1, c_2) and c_i is a list of atoms which exhibits the properties expressed in Definition 4.

*Partitions Rule R**

R*: Given a triple (g, v, ϕ), return the set of pairs $\{C_1, \ldots, C_n\}$, $n \geq 0$, built in the following way:

– for every C_i, let r_i be the instantiation degree associated to the conjunctions of atoms $g_{i1}\theta_i$ s.t. g_{i1} is the conjunction of the atoms in g which are not in c_{i1} and θ_i is the substitution associated to the pair $C_i = (c_{i1}, c_{i2})$.

– select all the pairs C_i whose r_i is > p, $0<p<1$, where p is a constant whose value depends on the specific nature of the application under consideration (cardinality of the base relations, existence of indexes on certain attributes, etc.).

– if there are more pairs C_i s.t. the roots of their associated trees are identical, only one of such computation will be executed while all the other will contribute to the constraint construction (see example.2). ♦

5.2 The set ϕ and the search rule

During the search of the tree, the set ϕ will not be fixed once and for all but, as we will see, it will be dinamically updated starting from the empty set.

Search Rule

The goal is to visit the mixed-SLD-tree in order to get every (distinct) solution.

Init: Tree_id =1; $\phi = \varnothing$; g=←$(A_1, \ldots, A_k)$; v= true; Suspended_nodes = list ([nil], Tree_id)

Tree: If R* (g, v, ϕ) $\neq \varnothing$

then

Bottom-up:

Let R*(Gi, ϕ) = $\{C_1 = (c_{11}, c_{12}), \ldots, C_n = (c_{n1}, c_{n2})\}$, $n \geq 1$. Let $\theta_1, \ldots, \theta_n$ be n substitutions, such that each θ_i is the composition of the m.g.u.*s* for the elements in c_{i1} and c_{i2} (see the R* definition). Let $G_i\theta_1, \ldots, G_i\theta_n$ be the roots of the n subtrees.

for all $G_i\theta_j$ $1 \leq j \leq n$

do Tree_id = *f*(Tree_id, G_i, θ_j); Suspended_nodes = append (Suspended_nodes, ([nil], Tree_id)); g= goal ($G_i\theta_j$); v = constraint ($G_i\theta_j$); goto **Tree** od.

Let G_i* be the new derived node whose associated goal is identical to the G_i one but whose constraint has been properly updated; let g= goal(G_i*); v= constraint(G_i*); goto **Tree**.

else

Top-Down:

If R(g = ←(B_1, ..., B_m)) = ∅ then

If first(Suspended_nodes, Tree_id) = [nil] then remove (Suspended_nodes, ([nil], Tree_id)); return

else let G be a node in the Tree_id Suspended_nodes sublist whose instantiation degree is maximal; let g= goal(G); v= constraint(G); goto **Tree**;

else let R(g = ←(B_1, ..., B_m)) = B_j and let G1, ..., Gk be the k descendant nodes;

If k>0 then select G_i whose instantiation degree is maximal and it is the leftmost in the tree; insert((G_1, ..., G_{i-1}, G_{i+1},..., G_k), Tree_id) in Suspended_nodes; let $\phi' = \phi \cup \{B_j\,\theta_r \mid B_j$ is a base predicate and θ_r a ground substitution with $1 \le r \le k\} \cup \{A_h \mid A \leftarrow C_1, \ldots, C_n$ is in the IDB and, it exists a σ s.t. $C_1\sigma, \ldots, C_n\sigma$ are in ϕ and $A\sigma = A_h\}$ else return fail;

If ($\phi' = \phi$ or it does not exist a Gj in the Tree_id Suspended_nodes sublist s.t. R* (Gj, ϕ) ≠ ∅) then g = goal(Gi); $\phi = \phi'$; v = v; go to **Tree**;

else insert (Gi, Tree_id) in Suspended_nodes; forall Gj in Suspended_nodes s.t. R*(Gj, ϕ) ≠ ∅ do g=goal(Gj); v= v; go to **Bottom-up** od;

♦

Init: The root node of the (initial) tree is the pair (Q, true) where Q is the query-goal←(A_1, ..., A_k) and true is the initial constraint. In the initialization phase the set ϕ is empty, and two control data structures are defined: Tree_id serves the purpose of uniquely identifying the trees while the list Suspended_nodes records all those nodes, in the search space, that have still to be examined.

Tree: This is the main loop, where the main idea is reflected: If it is possible to go forward, then go ahead, otherwise resort to the usual top-down strategy.

Bottom-Up: In this branch a bottom-up step is performed generating n new subtrees that are iteratively inspected. Then, the control goes back to the computation that generated the n subtrees, keeping track of the already done work by updating the constraint of the original node. Figure 4 illustrates the situation.

Top-Down: In this branch the top-down strategy is applied if it is not possible to proceed forward in the inspection of the current subtree. During this phase the set ϕ is updated.

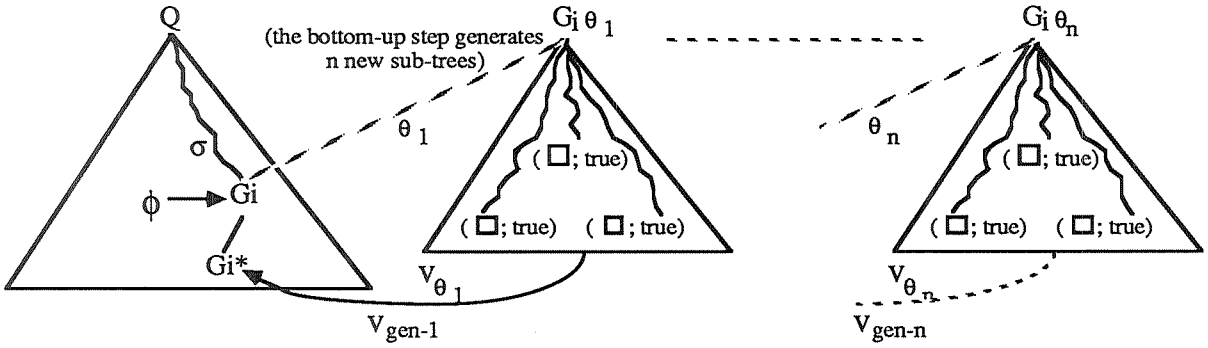


Figure 4

5.3 Some search strategy optimizations

Here we describe three different optimizations that can be applied to the above described search strategy. Actually, the final strategy contains the first two of these optimizations and it is only for the sake of simplicity (sic!) that we have decided to present them separately, from the bare strategy structure.

The optimizations can be summarized as follows:

i) the first one is related to the set ϕ. In the Top-Down phase above, the set ϕ is updated by adding to the previous one all the base and derived facts demonstrated in that step. This may be very costly since, as long as the computation proceeds, ϕ becomes larger and larger.

Thus, we have decided to keep only the part that is added to ϕ, that is we put $\phi' = \{B_j\,\theta_r \mid B_j$ is a base predicate and θ_r a (top-down derived) ground substitution with $1 \le r \le k\} \cup \{A_h \mid A \leftarrow C_1, \ldots, C_n$ is a clause in the IDB and it exists a σ s.t. $A\sigma = A_h$, $A\sigma, C_1\sigma, \ldots, C_m\sigma$ are instances of $A, C_1, \ldots, C_m$ respectively, with $m \le n$, $C_1\sigma, \ldots, C_m\sigma$ are in ϕ' and $C_{m+1}\rho_1, \ldots, C_n\rho_k$ belonged some previously computed ϕ's$\}$. Informally, this means that we keep track, during the computation, of the information that still has to be proved to demonstrate the fact A_h. This will be clear in the example were the partially computed clauses decorate the nodes.

ii) The second optimization is related to the propagation of constraints. We deal with two cases:

a) Referring to the strategy presented above, in the bottom-up step, the constraint v, relative to the n subtrees computations, originated by the current node G, contributes only to the generation of the constraint v* associated to the unique descendant node G* of G. Now, it is easy to see that it is much more convenient to propagate the most general information "v_{gen}" of v to all the suspended nodes in the tree containing G. See Figure 5 below.

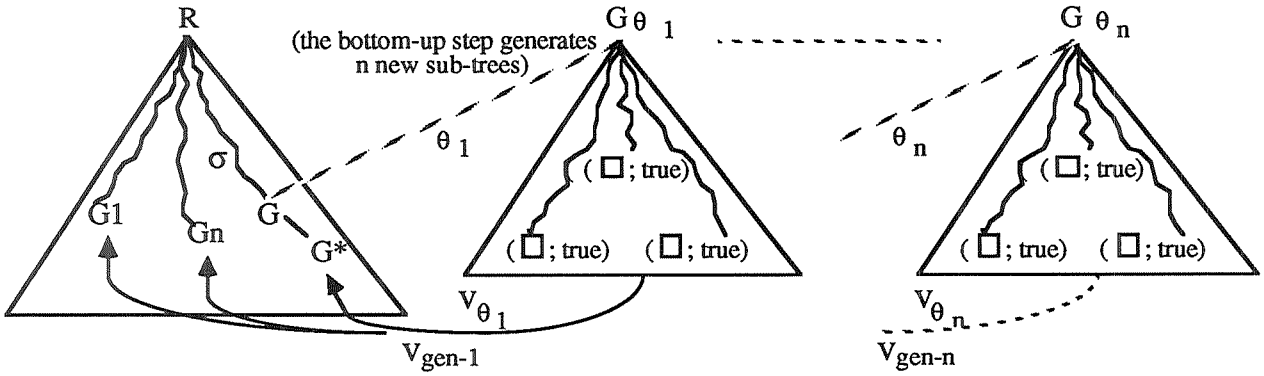


Figure 5 case a)

b) Similarly, it is possible to improve the effect of the propagation of general constraint as soon as possible *inside* the current tree. That is, if during the computation, a solution for the initial query-goal is obtained, it is useful to propagate such information to all the suspended nodes in the current tree. In this case, in order to propagate the constraint, it is not necessary to reach a specific point of the computation, as for example, in case a). This introduces, in the strategy, a sort of asynchronous mechanism of general constraint propagation. Note that such asynchrony applies only inside the current tree. Of course, in order to perform such propagation it is necessary to manage the various variable renamings/instantiations that may occur to the variables of the query-goal along the considered paths. See Figure 6 below.

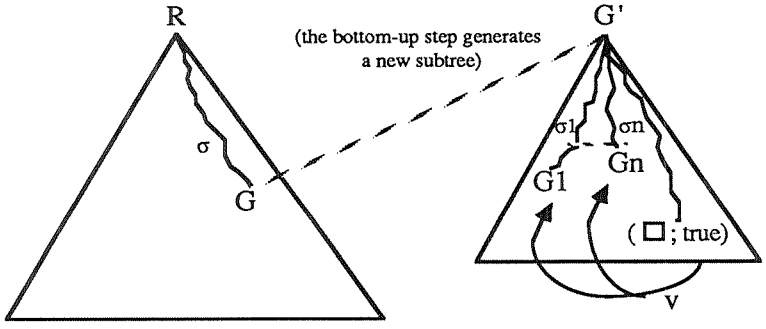


Figure 6 case b)

iii) The third possible optimization concerns the potential parallel execution of the n subtrees during the bottom-up step. It is, in fact, possible to change the strategy structure as presented above in a new one in which the iterative step implemented with a forall construct can be replaced with a parallel construct, thus exploiting the inherent parallelism of the bottom-up step.

Example.2

In the following, we present a simple example in order to illustrate how the method works. The example is given in two versions: In the former, the strategy with only the i) optimization is adopted; in the latter also optimization ii) is introduced.

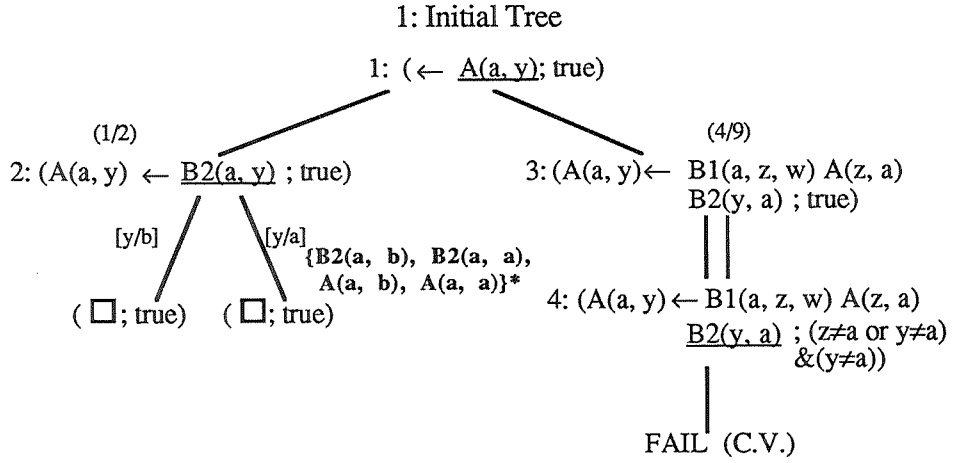
We refer again to the IDB of example 1 of pag. 2.

Let the EDB consists only of: B2(a, b), B2(a, a), B1(a, d, b) e B1(d, c, d).

In discussing the example we adopt a number of conventions, that are briefly summarized:

Nodes are denoted with a goal and a constraint. In order to keep track of the aumont of information that have to be proved to deduce a derived fact, instances of the head of the clauses used during the resolution, are in the goal mantained also. Parenthesis serve the purpose of separating atoms introduced in the goal by different clauses. In this way it is easy to see when a derived fact becomes true. The unique tree identifier is computed by using an obvious integer numerical convention. The quotients that decore some of the nodes indicate the instantiation degrees that are used by the selection rule R. The ground assertions derived in a top-down step are indicated in curled parenthesis and label the arcs between the father node and its descendants. Every time one of such sets originates new subtrees, one or more stars decore it. Thus, a presence of a star means that the analysis switched from the current subtree to the subtree(s) generated from the star decored set. In this example we fix the value of the constant p= 1/4.

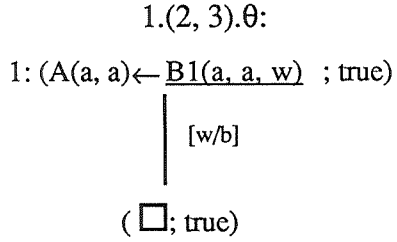
When a node derives from the completion of a bottom-up step a double arc is used. Furthermore, if a derivation fails because a constraint has been violated, it will be recorded by a C.V. mark (Constraint Violated) beside the keyword fail.



Note that, relatively to node 4, the selection rule chooses the more instantiated atom B2(y, a), instead of B1(a, z, w).

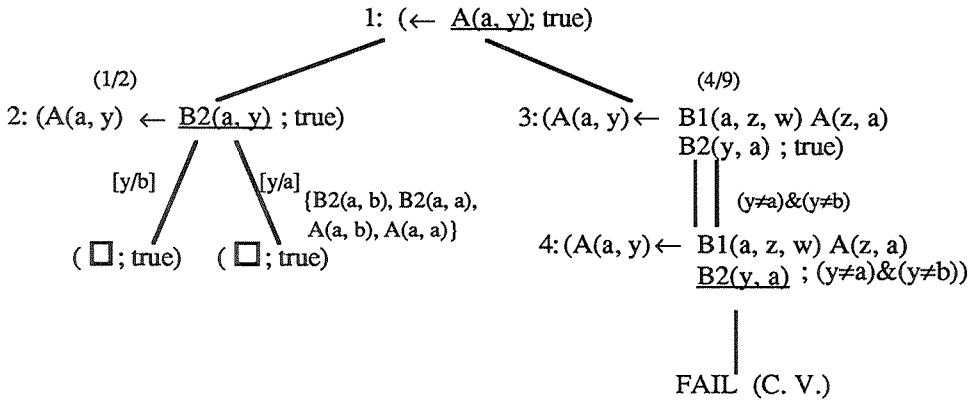
ϕ = {**B2(a, b), B2(a, a), A(a, b), A(a, a)**}*.

The only suspended node is the node 3. For that node, R* gives the substitution θ = [z/a, y/a] with instantiation degree 2/3 >p, obtained by using the two facts B2(a, a) and A(a, a). The corresponding subtree is:



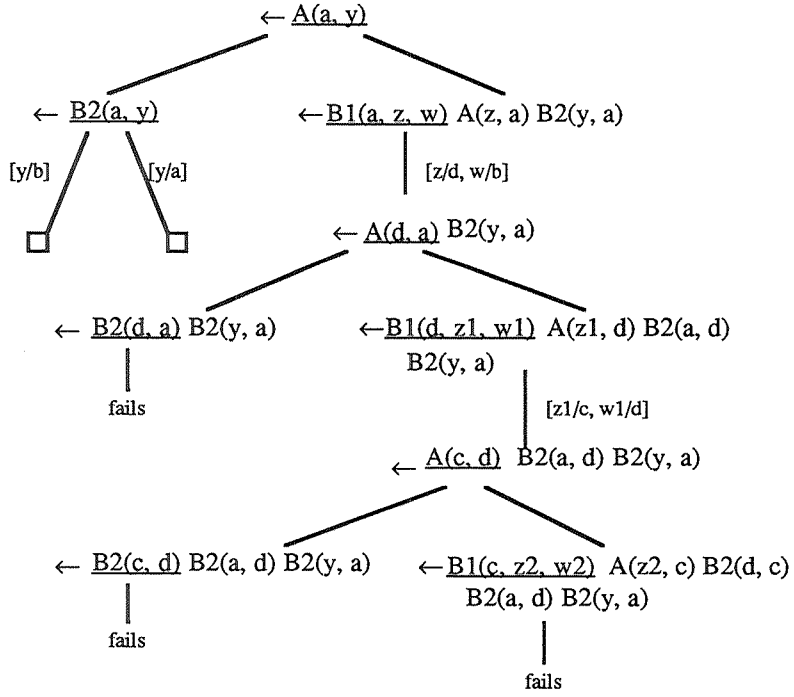
By visiting this tree the constraint (z≠a or y≠a)&(y≠a), composed by the local constraint (z≠a or y≠a) and by the general constraint (y≠a) is obtained. ♦

Let us now see what happens if we introduce the optimization ii) in the search rule:



The general constraint (y≠a) is propagated to the node 3 with the effect of avoiding the subcomputation. The generation of the new node 4, derived from the propagation of the general constraints (y≠a)&(y≠b) is depicted by using again a double arc this time labelled by the constraints.

For the sake of completeness we report also the standard Prolog-like computation:



It is immediate to see the high redundancy with respect to the database accesses, even in this very poor EDB.

We may try to quantify, very roughly, such redundancy by considering the number of calls to the two base predicates in the two strategies:

	B1	B2
Mixed-SLD-Strategy + optimizations	0	2
Prolog-like strategy	3	3

It is worthwhile noticing that the Mixed SLD Strategy, by means of constraints has also a beneficial influence on the size of the involved intermediate relations. ♦

5.4 About the correctness and completeness of the search strategy

The correctness of the strategy is preserved since it does not depends on the way the set ϕ is computed. The fact that we do not manipulate, during the whole computation, always the same set of facts is equivalent to ideally adopting a specific "local" partition rule which operates on the set of all the ground logical consequences of $P \cup E$ by selecting, time at a time, only the pair related to the last deduced facts.

As concerns completeness, it has to be noted that the selection rule R is not fair, as for example the leftmost selection rule. This means that it is possible to remain inspecting an infinite branch for ever. In this respect, constraints play a positive role. In a large number of situation it is, in fact, possible to detect and to block the inspection of infinite branches.

6. Related works and Conclusions

The adopted use of atomic lemmas for generating answers to recursive queries is not a completely new idea. The QSQ methods [Vieille '86], do adopt a similar use of lemmas, but only related to derived predicates. In particular, [Vieille '89] describes the SLD-AL Resolution, a theorical framework for the QSQ methods. The SLD-AL Resolution is derived from the SLD Resolution by the introduction of an *admissibility* test and of a *lemma* resolution (AL = Admissibility + Lemma). The derived lemmas are used for resolving non-admissible subgoals, i.e. "similar" to "previous" one.

[Lozinskii '85] presents a mixed top-dowm/bottom-up evaluation strategy, called APEX. This method adopts a forward use of the clauses, starting from sets of relevant facts, and resumes backward reasoning to select new subqueries to be answered. At compile time a *Migration Set* is computed for each variable in the derived predicates. Then, it is this set that drives the access to EDB and the construction of the set of relevant facts. Thus, it differes from our approach in the way the set of relevant facts is determined and there does not exist anything like our constraints to reduce the amount of redundant backward reasoning.

On the side of Logic Programming it is interesting to notice that constraints are becoming a very promising tool either as a general programming methodology [Jaffar&Lassez '87] or as a mean to provide a more powerful framework for the treatment of negative non ground information [Chan '88, Turi '89]. In this respect note that the problem of non ground negative information in LP is mainly the one of an exhaustive search of the SLD-tree as it happens for the query evaluation process.

Concluding, we have presented a new evaluation method which combines bottom-up and top--down strategy to improve the efficiency of a top-down based strategy. Its main novel feature is the use of a constraint mechanism to drive the computation and this approach seems promising since it may benefits of the great amount of work which is currently undertaken in the LP area. Furthermore, it is possible to foresee a further use of the negative information (disequalities) contained in the constraint to perform selective accesses to the base predicates.

An other interesting aspect of our method is its potential inherent parallelism that could be properly exploited.

References

[Bancilhon&Ramakrishnan '86] Bancilhon, F. and Ramakrishnan, R. "An Amateur's Introduction to Recursive Query Processing Strategies", *Proc. ACM SIGMOD, Int. Conf. on Management of Data*, Washington, DC, 16-52, 1986.

[Chan '88] Chan, D. "Constructive Negation Based on the Completed Database", in *Proc. 5th Int. Conf. and Symp. on Logic Programming*, Seattle, (Kowalski, R.A. and Bowen, K. Eds.), Mit Press, Cambridge, Mass., 111-125, 1988.

[Henschen&Naqvi '84] Henschen, L. and Naqvi, S. "On Compiling Queries in Recursive First-Order Databases", *J. of the ACM*, Vol. 31, N. 1, 47-85, 1984.

[Jaffar&Lassez '87] Jaffar, J. and Lassez, J.L. "Constraint Logic Programming", *Proc. 14th ACM Principles of Programming Languages Conf.*, 111-119, 1987.

[Lloyd '87] Lloyd, J. W, *Foundations of logic programming*, Second Edition, Symbolic Computation Series, Springer, 1987.

[Lozinskii '85] Lozinskii, E.L. "Evaluating Queries in Deductive Databases by Generating", *Proc. 9th Int. Joint Conf. on Artificial Intelligence*, 173-177, 1985.

[Raffaelli '90] Raffaelli, V. "La Programmazione Logica ed i Metodi di Valutazione di Query Logiche Ricorsive: Confronti e Proposte", Tesi di Laurea in Scienze dell'Informazione, Univ. di Pisa, 1990.

[Turi '89] Turi, T. "Programmi Logici con Negazione", Tesi di Laurea in Scienze dell'Informazione, Univ. di Pisa, 1989.

[Ullman '85] Ullman, J.D. "Implementation of Logical Query Languages for Databases", *ACM Trans. on Database Systems*, Vol. 10, N.3, 289-321, 1985.

[Vieille '86] Vieille, L. "Recursive Axioms in Deductive Databases: the Query/Subquery Approach", *Proc. First Int. Conf. on Expert Database Systems*, Charleston, 253-267, 1986.

[Vieille '89] Vieille, L. "Recursive Query Processing: the Power of Logic", in *Theoretical Computer Science*, Vol. 69, 1-53, 1989.

Appendix

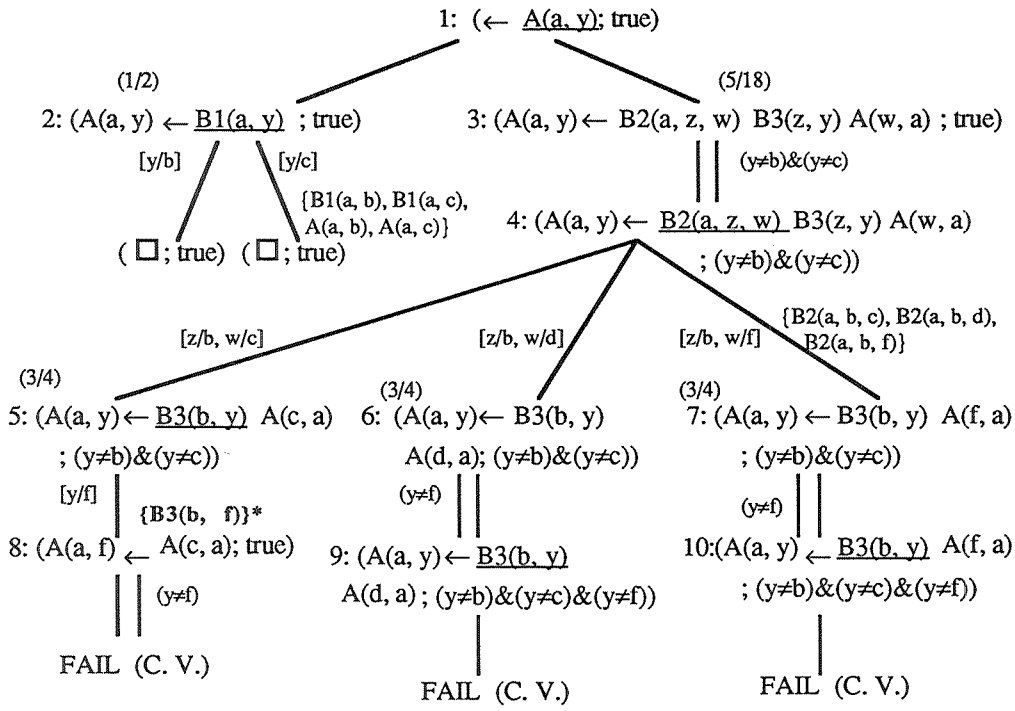
In this example a more complex case is treated. The strategy is considered with both optimization i) and ii). As before we first present the IDB and the EDB, and then the mixed-SLD-trees that are part of the computation. At the end the SLD tree obtained with the standard prolog strategy follows.

Note that, in this case, the instantiation degrees of the atoms returned by the R* rule, are always

equal to 1, thus independently from which value the constant p assumes all the trees are visited.

r_0: A(x, y):-B1(x, y).
r_1: A(x, y):-B2(x, z, w), B3(z, y), A(w, x).
query-goal: A(a, y)?

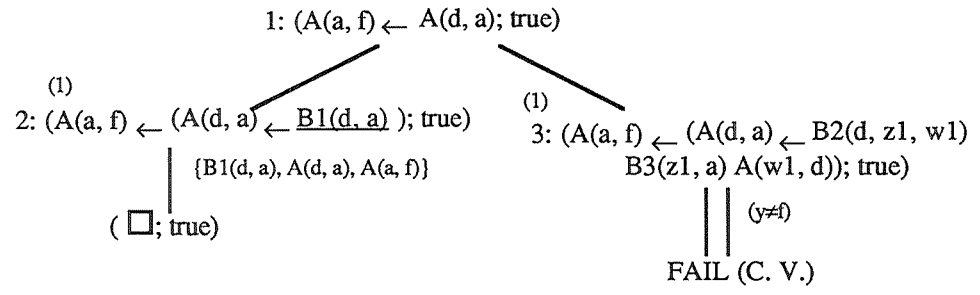
B1	B2	B3
(a, b)	(a, b, c)	(b, c)
(a, c)	(a, b, d)	(b, f)
(c, a)	(a, b, f)	(f, a)
(d, a)	(d, a, b)	(g, h)
(f, a)	(f, g, h)	

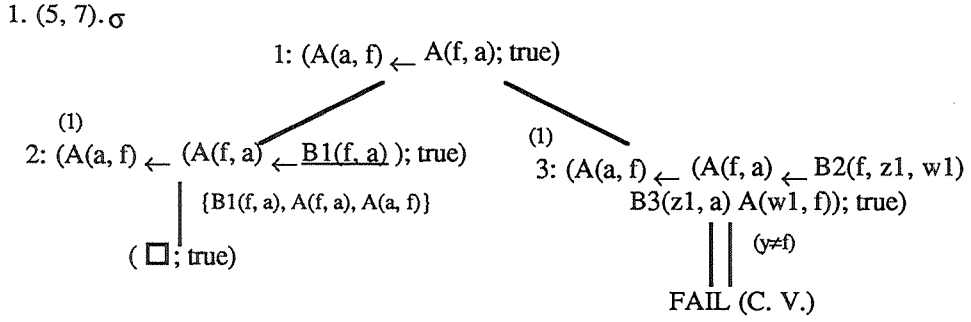


*{B3(b, f)}. Suspended nodes in the tree 1.: 6 e 7.

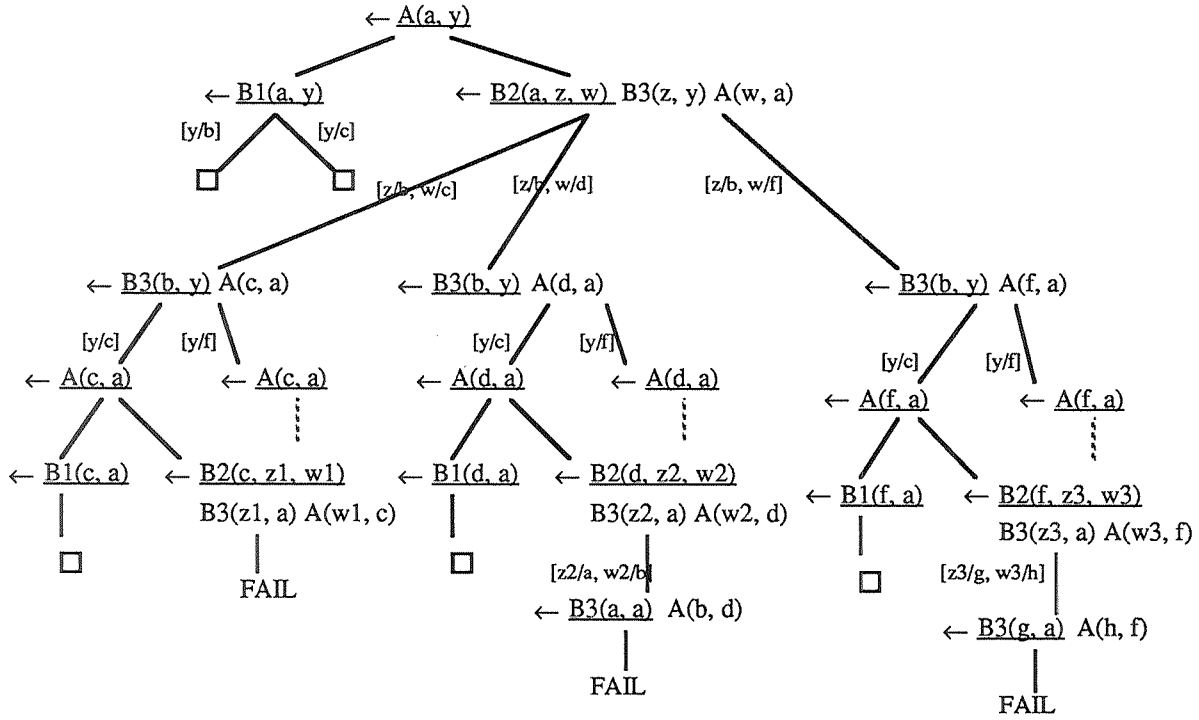
For node 6, θ = [y/f] and the instantiation degree is 1/1; for node 7, σ = [y/f] and same instantiation degree. The corresponding subtrees are the following:

1. (5, 6). θ





From the two subtrees, 1.(5, 6).θ e 1.(5, 7).σ, the constraint (y≠f) is obtained which is a general constraint and then can be propagated into the tree 1 thus letting node 8 to fail. In the nodes 8 and 9 the two attempts to access to the base predicate B3 do not succed because of the presence of constraints, thus ending the computation. The computed answer set is relative to the variable y and is the following: {b, c, f}.
Now, it follows the Prolog_like tree:



	B1	B2	B3
Mixed-SLD-strategy	3	1	3
Prolog-like strategy	7	7	7