Theory and Practice of Logic Programming **1** (5): 539–590, September 2001.

Printed in the United Kingdom © 2001 Cambridge University Press

*On redundancy elimination tolerant scheduling rules**

FILOMENA FERRUCCI, MARIA I. SESSA
DMI Universita' di Salerno - via S. Allende, 84081 Baronissi (SA), Italy
(*e-mail:* {filfer,mis}@unisa.it)

GIULIANO PACINI
Accademia Navale, Viale Italia 72, 57100 Livorno, Italy
(*e-mail:* pacini@di.unipi.it)

Abstract

In Ferrucci, Pacini and Sessa (1995) an extended form of resolution, called Reduced SLD resolution (RSLD), is introduced. In essence, an RSLD derivation is an SLD derivation such that redundancy elimination from resolvents is performed after each rewriting step. It is intuitive that redundancy elimination may have positive effects on derivation process. However, undesiderable effects are also possible. In particular, as shown in this paper, program termination as well as completeness of loop checking mechanisms via a given selection rule may be lost. The study of such effects has led us to an analysis of selection rule basic concepts, so that we have found convenient to move the attention from rules of atom selection to rules of atom scheduling. A priority mechanism for atom scheduling is built, where a priority is assigned to each atom in a resolvent, and primary importance is given to the event of arrival of new atoms from the body of the applied clause at rewriting time. This new computational model proves able to address the study of redundancy elimination effects, giving at the same time interesting insights into general properties of selection rules. As a matter of fact, a class of scheduling rules, namely the specialisation independent ones, is defined in the paper by using not trivial semantic arguments. As a quite surprising result, specialisation independent scheduling rules turn out to coincide with a class of rules which have an immediate structural characterisation (named stack-queue rules). Then we prove that such scheduling rules are tolerant to redundancy elimination, in the sense that neither program termination nor completeness of equality loop check is lost passing from SLD to RSLD.

KEYWORDS: redundancy elimination, selection rule, scheduling rule, termination, loop check, Stack-Queue scheduling rule

1 Introduction

Several different approaches have been considered so far to enrich the SLD resolution in order to improve the performance of top-down interpreters. The usual

* This work has been partially supported by the Italian National Research Council (CNR) research project "Tecniche di taglio dei cicli e loro implementazione in ambiente di Programmazione Logica", Grant No. 97.02432.CT12.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

objective is to reduce the search space without loss of results of the refutation process, possibly obtaining a finite search space. Among the proposed methods, the loop check mechanisms (Apt, Bol and Klop, 1989; Bol, Apt and Klop, 1991; Smith, Genesereth and Ginsberg, 1986; Van Gelder, 1987) and the tabulation technique (Bol and Degersted, 1998; Dietrich, 1987; Ramakrishnan *et al*., 1999; Tamaki and Sato, 1986; Vieille, 1989) aim to eliminate redundant computations and to enforce the termination of a query over a logic program.

Loop check mechanisms provide the interpreter with the capability of pruning certain nodes of the SLD tree. The pruning is based on excluding some kinds of structural repetitions for the goals in a derivation path. When suitable structure repetitions are found, further rewritings of the current node are ignored, because any solution possibly existing in the cut sub-tree is also present in other parts of the SLD tree. Different forms of loop checks are proposed in the literature. In particular, Bol et al. have defined several *simple* loop checks, i.e. loop checks whose pruning mechanisms do not depend on the considered logic program, and have analysed them against the basic property of soundness and completeness (Bol, Apt and Klop, 1991). The completeness property concerns with the capability of pruning every infinite derivation. In contrast, soundness concerns with the preservation of the computed answer substitutions.

The main idea of tabulation originates from functional programming and consists in building a table during the search of answers in an SLD tree. The table contains entries for atoms with the corresponding answers so far computed. These answers are to be used later, when instances of such atoms should be recomputed. Such instantiated occurrences are named *non-admissible atoms* (or *consumer*). In essence, non-admissible atoms are not resolved against clauses but against answers computed in other parts of the SLD tree. The re-using approach exploited by the tabulation technique was already mentioned by Kowalski (1979), and has been proposed several times under different names, such as memo-isation (Dietrich, 1987) and the AL-technique (Vieille, 1989).

The conceptual differences between loop checks and tabulation are reflected in several interesting aspects. In particular, tabulation requires a local selection rule to guarantee the answer preservation, while no missing of solution is possible with (sound) loop checks independently of the used selection rule. On the other hand, the tabulation technique ensures termination for any function-free program and for any program with a finite Herbrand model, while the completeness of loop checks takes place for specific classes of programs possibly with respect to given selection rules (Bol, 1992; Bol, Apt and Klop, 1991; Pacini and Sessa, 2000). Finally, loop checks exploit no auxiliary data structure and the pruning decision usually depends on the current derivation only, while tabulation needs a table to store the answers of atoms solved in the previously traversed portion of the tree.

Proposals can be also found in literature for a synergistic use of different techniques aiming to optimise the query evaluation procedure. In particular, in Vieille (1989), a loop checking mechanism is combined with the tabulation technique in order to eliminate some redundant parts of the search space. In Ferrucci, Pacini and Sessa (1995), the simple loop check mechanisms proposed in Bol, Apt and Klop (1991)

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

are combined with another form of redundancy elimination which is named (goal) *reduction*. Goal reduction is conceptually analogous to the *condensing* technique proposed by Joyner for the proof of the unsatisfiability of first-order formulas (Joyner, 1976). In both cases, redundant atoms are eliminated from resolvents, in order to avoid useless computations and to contain the size of the resolvents at the same time. The main idea of reduction originates from the observation that if there exists a refutation for an atom, then a refutation exists also for any more general version of that atom. In this sense, such more general versions can be seen as potentially redundant and we can imagine to remove them from the resolvent, though suitable cares are to be taken as discussed in Ferrucci, Pacini and Sessa (1995). By goal reduction, a generalised form of SLD resolution (named RSLD) can be obtained, where a reduction of the resolvent is performed after each rewriting step.

A goal reduction technique has a modus operandi which shows evident affinity with the one of loop checking mechanisms. Indeed, with reduction redundant atoms are definitively ignored, as it is done with loop checks for pruned nodes. This is not the case with tabulation, in the sense that non-admissible atoms, which are indeed solved against previously tabulated answers, are not redundant. Such different philosophy between tabulation and RSLD is highlighted also by the fact that the reduction technique eliminates atoms in their *more general version*, while non-admissible atoms are *instances* of previously solved goals. It is evident that RSLD does not need any auxiliary data structure because it considers only the current goal (not even the current derivation path). The soundness of RSLD is shown in Ferrucci, Pacini and Sessa (1995) independently of the selection rule used. This means that RSLD does not require particular selection rules in order to ensure answer preservation.

It is intuitive that redundancy elimination may have positive effects on derivation process. In Ferrucci, Pacini and Sessa (1995), advantageous combinations are shown with respect to loop checking mechanisms. In particular, it is proven that a well known simple loop check mechanism, namely Equality Variant check of Resultant as Lists (EVR_L), becomes complete for several classes of programs, provided that RSLD is exploited instead of usual SLD. The specific reason is that the length of resolvents can be maintained within the limit of a finite value through systematic elimination of redundant atoms. In essence, there is clear evidence that the strength of equality loop checks can augment if RSLD resolution is used.

However, even though not completely intuitive, redundancy elimination can produce undesirable effects, too. In fact, as exemplified later, problems can arise with program termination, as well as with the completeness of loop checking mechanisms. The rationale behind this is that redundancy elimination can affect the actual sequence of atom rewriting with respect to given selection rules. This can (infinitely) delay the selection of failing atoms, so that termination is missed. On the other hand, the structure of the obtained resolvents can be altered by redundancy elimination, so that loop checks may become unable to detect infinite derivations.

As shown in this paper, missing termination and loop detection depends critically on the used selection rule. We say in the sequel that a selection rule is *redundancy*

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

elimination tolerant if no loss in termination and/or loop detection comes out, passing from SLD to RSLD.

In Section 2, we prove that termination and EVR_L completeness are preserved if they hold in SLD with respect to all possible selection rules. Then, a more accurate analysis of redundancy elimination tolerance is performed. To this aim, a careful reconsideration of selection rule basic concepts will be required, so that we will be led to a reformulation of selection rule ideas in terms of their operational counterparts, namely *scheduling mechanisms*, so that we will prefer to talk of tolerant scheduling rules. As a matter of fact, in Section 3 we provide a highly expressive execution model based on priority mechanism for atom selection. A priority is assigned to each atom in a resolvent, and primary importance is given to the event of arrival of new atoms from the body of the applied clause at rewriting time. Indeed, new atoms can be freely positioned with respect to the old ones in the resolvent, through the assignment of priority values according to a given scheduling rule. Then, at any derivation step, the atom with optimum priority is simply selected.

This new computational model proves able to address the study of redundancy elimination effects, giving at the same time interesting insights into general properties of selection rules. As a matter of fact, in Section 4 a class of scheduling rules, namely the *specialisation independent* ones, is defined by using not trivial semantic arguments. Several properties of specialisation independent scheduling rules are also proven. As a quite surprising result, in Section 5 we show that specialisation independent scheduling rules coincide with *stack-queue rules*, which have an immediate structural characterisation. Indeed, the stack-queue scheduling technique is simply defined so that, in order to obtain the new resolvent at rewriting time, part of new atoms are stacked at the beginning of the old resolvent while the remaining ones are queued. Then in Section 6 we prove that such scheduling rules are tolerant to redundancy elimination, in the sense that neither program termination nor completeness of equality loop check is lost passing from SLD to RSLD. The proof is largely based on properties which we have established for specialisation independent (and stack-queue) scheduling rules.

2 Goal reduction, program termination and EVR_L completeness

Throughout the paper we assume familiarity with the basic concepts of Logic Programming (Apt, 1990; Apt, 1998; Lloyd, 1987).

Here, only some notations are given about SLD derivation procedure, which can be described as follows. Let $G = a_1, a_2, ...a_k$ be a *goal*, constituted by a conjunction of k atoms, and $c = (ht \longleftarrow B)$ a *clause*, where ht is an atom and B is a goal. The goal G' is a *resolvent* of G and c by a renaming ξ and a substitution θ, if an atom a_i exists, with $1 \preceq i \preceq k$, such that $G' = (a_1, ...a_{i-1}, B\xi, a_{i+1}, ...a_k)\theta$, where θ is an idempotent and relevant *mgu* of $(ht)\xi$ and a_i. In the sequel, given an expression E, the notation $var(E)$ will indicate the set of variables in E. Moreover, we will denote by $(G \xrightarrow{c\xi,\theta} G')$ the fact that G' is a resolvent of G and c by ξ and θ. Given an *initial goal* G_o and a logic program P, an *SLD derivation of* G_o *in* P is a possibly infinite sequence of the type:

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

$G_o \xrightarrow{c_o\xi_o,\theta_o} G_1 \ldots G_j \xrightarrow{c_j\xi_j,\theta_j} G_{j+1}\ldots$

such that, for any $j \geq 0$, each clause c_j belongs to P and each $c_j\xi_j$ is *standardised apart*, i.e.

$var(c_j\xi_j) \cap (var(G_o) \cup var(c_o\xi_o) \cup \ldots \cup var(c_{j-1}\xi_{j-1})) = \emptyset$.

A *selection rule* is a function which chooses the atom to be rewritten in the last resolvent of any finite SLD derivation. Given a selection rule S, an SLD derivation is *via* S if all the selections of atoms are performed in agreement with S. An SLD *refutation* is a finite SLD derivation such that the last resolvent is empty.

Now we can introduce the definitions of *goal reduction* and RSLD derivation. The reduction technique aims to eliminate redundant atoms from the resolvents in order to contain their size. Analogous issue was already been faced for the proof of the unsatisfiability of first-order formulas. Indeed Joyner (1976) noted that the increase in size of resolvents is a factor which prevents resolution strategies being decision procedures for solvable classes of first-order formulas (i.e. classes of formulas for which the question of satisfiability or unsatisfiability can be effectively decided). To limit the growth of the number of literals, Joyner introduced a technique for simplifying resolvents, called *condensing*. The condensation of a clause is defined as the smallest subset of the clauses which is also an instance of it. In other words, the condensation of a clause can be obtained by applying a substitution α and eliminating all the atom repetitions.

With reference to SLD derivations, the most evident form of redundancy corresponds to multiple occurrences of the same atom in a resolvent. It is obvious that this kind of atom repetition is essentially redundant. However, this is not the only possible case of redundancy. Indeed, the reduction technique, which is introduced in Ferrucci, Pacini and Sessa (1995) as a variant of Joyner's condensing technique, is able to perform quite general actions of redundancy elimination from resolvents while preserving the soundness and the completeness of RSLD resolution. By condensation, Joyner obtains a complete and sound resolution procedures, which work as decision procedures for several solvable classes of first order formulas (Joyner, 1976). By reduction, the well known sound EVR_L loop check becomes complete for several classes of logic programs (Ferrucci, Pacini and Sessa, 1995).

Intuitively, the basic idea of goal reduction technique can be explained as follows. Suppose having to refute a resolvent which contains $p(x)$ and $p(a)$, where x is a variable and a is a constant. Obviously, any refutation for $p(a)$ implies a refutation for the atom $p(x)$, as $p(x)$ is more general than $p(a)$. In this sense, the atom $p(x)$ may appear as a redundant one. Actually, in order to ensure the soundness of the derivation process, the elimination of redundant atoms (such as $p(x)$ above) is conditioned in two aspects which can be sketched through the following simple examples:

(a) Consider a resolvent like $p(x), q(x), p(a)$. In this case, the atom $p(x)$ cannot be eliminated, because the connection between the atoms $p(x)$ and $q(x)$, by the variable x, is lost.

(b) Suppose that x is a variable in the initial goal of a derivation, and the actual

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

resolvent is $p(x), p(a)$. In this case $p(x)$ cannot be dropped, because possible instantiations of x in computed answers could be lost. So we would obtain computed answers which are too general with respect the correct answers, thus missing soundness.

Now we present a formal definition of goal reduction which takes into account the observations (a) and (b) and follows the line of Definition 2.1 presented in Ferrucci, Pacini and Sessa (1995). We will denote by $\subseteq_L$ the inclusion relation between goals, and $G - N$ will indicate the goal obtained from G by eliminating the atoms which are present in N. In both cases the goals are regarded as lists.

Definition 2.1 (*Reduced goal*)
Let X be a set of variables, τ a substitution and G a goal. A goal N is a *reduced goal* of G by τ up to X, denoted by $G >>^{\tau} N$, if the following conditions hold:

(i) $N \subseteq_L G$,

(ii) $\forall b \in (G - N), \quad b\tau \in N$,

(iii) $\forall x \in (var(N) \cup X)$ it is $x\tau = x$.

In agreement with the above definition, a part $(G - N)$ of atoms of G can be eliminated if a substitution τ exists such that $b\tau \in N$, for any atom $b \in (G - N)$, provided that τ does not affect neither the variables in N nor those in X. The imposition that τ does not affect the variables in N prevents the kind of difficulties which are exemplified in (a).

Example 2.1
Let:

$G = p(z, v), q(w), p(w, v), p(w, x), p(w, y), q(v), q(y)$,

$X = \{x, w\}$.

The following goal N is a reduced goal of G by $\tau = \{z/w, y/v\}$ up to X:

$N = q(w), p(w, v), p(w, x), q(v)$. □

Performing reductions in the resolvents of an SLD derivation corresponds to an actual extension of the SLD resolution process. Then, a generalised version of SLD resolution can be introduced, i.e. the *Reduced SLD resolution* (RSLD in the sequel), where at any resolution step a reduction of the resolvent is allowed. The following is the formal definition of RSLD derivations.

Definition 2.2 (*Reduced SLD derivation*)
Let P be a program and G_o a goal. A *Reduced SLD derivation* of G_o in P (RSLD in the following) is a possibly infinite sequence of the form:

$$G_o >>^{\alpha_o} N_o \overset{c_o\xi_o,\theta_o}{\longrightarrow} G_1 \ldots G_h >>^{\alpha_h} N_h \overset{c_h\xi_h,\theta_h}{\longrightarrow} G_{h+1} >>^{\alpha_{h+1}} N_{h+1} \ldots$$

where, for any $j \geq 0$,

(i) c_j is a clause in P,

(ii) $var(c_j\xi_j) \cap (var(G_o) \cup var(c_o\xi_o) \cup \ldots \cup var(c_{j-1}\xi_{j-1})) = \emptyset$,

(iii) $G_j >>^{\alpha_j} N_j$ up to $var(G_o\theta_o \ldots \theta_{j-1})$.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

It is evident that an SLD derivation is a particular case of RSLD derivation where $G_j = N_j$, for any j. Each N_j is called a *reduced resolvent*. Condition (ii) above is the usual standardisation apart requirement. Condition (iii) prevents the kind of difficulties which are exemplified in (b), guaranteeing the soundness of the mechanism. The soundness and completeness of RSLD resolution are proven in Theorems 2.1 and 2.2 of Ferrucci, Pacini and Sessa (1995).

2.1 Program termination

The completeness of RSLD resolution ensures that missing computed answers is impossible when we pass from SLD to RSLD. This is not the case with termination, as shown by the following Example 2.2. In the example a selection rule S and a program P are given, such that any SLD derivation of P via S terminates independently of the initial goal. However, we show that termination is lost, if reduction of resolvents is performed.

Example 2.2
Let us consider a selection rule S such that, given a goal G, the first atom is chosen for rewriting if the length of G is odd, and the last atom is chosen otherwise. Let us consider the logic program P consisting of the following clause:

$$c = p(x, y) \longleftarrow q, p(x, z_1), p(z_1, z_2), p(z_2, y).$$

It can be easily seen that all SLD derivations in P via S terminate, independently of the initial goal. Indeed, suppose that the initial goal has an odd number of atoms. It is evident that either the derivation via S fails immediately or the initial goal has the form '$p(..), Y$', so that the first step of the derivation produces a resolvent of an even length as follows:

$$p(..), Y \xrightarrow{c} q, p(..), p(..), p(..), Y.$$

Now, either the derivation fails immediately or $Y = Z, p(..)$, so that a second derivation step is performed:

$$q, p(..), p(..), p(..), Z, p(..) \xrightarrow{c} q, p(..), p(..), p(..), Z, q, p(..), p(..), p(..),$$

and the process fails anyway, since the last resolvent has an odd length. Then, suppose on the contrary that the initial goal has an even number of atoms. Either the derivation fails immediately or the initial goal has the form '$T, p(..)$'. In the second case, the first derivation step gives place to a resolvent with an odd length, so that the derivation fails. Now, let us verify that termination can be lost if reduction of resolvents is performed. Indeed, let us consider the RSLD derivation of the goal $(q, p(x, x))$ in P via S given in Figure 1. It is evident that the number of atoms is even in any reduced resolvent. Thus, the last atom is always selected and the derivation is infinite. □

As shown by the example in Figure 1, termination with respect to a given selection rule can be missed, if we pass from SLD to RSLD resolution. On the contrary, we show in this section (Theorem 2.1) that termination is preserved, when any SLD derivation of G in P is finite independently of the used selection rule. Theorem 2.1 will be proven as an immediate consequence of the following Lemma 2.1

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Resolvents	Reduced Resolvents
$q, p(x,x)$	
$>> q, p(x,x)$	$\xrightarrow{c}$
$q, q, p(x,z_1), p(z_1,z_2), p(z_2,x)$	
$>> q, p(x,z_1), p(z_1,z_2), p(z_2,x)$	$\xrightarrow{c}$
$q, p(x,z_1), p(z_1,z_2), q, p(z_2,z_3), p(z_3,z_4), p(z_4,x)$	
$>> q, p(x,z_1), p(z_1,z_2), p(z_2,z_3), p(z_3,z_4), p(z_4,x)$	$\xrightarrow{c}$
.....	.....

Fig. 1

Lemma 2.1
Let P be a program and G_o a goal. For any possibly infinite RSLD derivation D of G_o in P, an SLD derivation D' of G_o in P exists, such that every reduced resolvent of D is included in the corresponding resolvent of D' up to renamings.

Proof
Consider a possibly infinite RSLD derivation D of G_o in P

$$D = (G_o >>^{\alpha_o} N_o \xrightarrow{c_o\xi_o,\theta_o} G_1 ... \\ ... G_h >>^{\alpha_h} N_h \xrightarrow{c_h\xi_h,\theta_h} G_{h+1} >>^{\alpha_{h+1}} N_{h+1} ...) \quad (1)$$

Intuitively, the SLD derivation D' is obtained by choosing, step by step, the same clause and the same atom as in D. This way, redundant atoms are not eliminated from resolvents of D', but they have no real influence on the derivation process. More formally, suppose that an SLD derivation of G_o in P is already constructed like

$$G_o \xrightarrow{c_o\xi'_o,\theta'_o} G'_1 ... \longrightarrow G'_i, \quad (2)$$

such that, for any $0 \preceq j \preceq i$, a renaming τ_j exists with $N_j\tau_j \subseteq_L G'_j$. It is easy to show that derivation (2) can be extended of one step in agreement with the lemma. Let a be the atom which is rewritten in the step $N_i \xrightarrow{c_i\xi_i,\theta_i} G_{i+1}$ of derivation (1). It is evident that the clause c_i is applicable to the atom $a\tau_i \in N_i\tau_i \subseteq_L G'_i$, so that we have an SLD derivation step of the form:

$$G'_i \xrightarrow{c_i\xi'_i,\theta'_i} G'_{i+1}. \quad (3)$$

Now let F denote the sublist of atoms in G'_{i+1} which derives from $N_i\tau_i$. It is obvious that the subgoal $(G'_i - N_i\tau_i)$ has no active role in derivation step (3). So, we have that F is a variant of G_{i+1}, i.e. a renaming τ_{i+1} exists with $F = G_{i+1}\tau_{i+1}$, which means that $G_{i+1}\tau_{i+1} \subseteq_L G'_{i+1}$. But, by definition of goal reduction we have $N_{i+1} \subseteq_L G_{i+1}$. As a consequence

$$N_{i+1}\tau_{i+1} \subseteq_L G_{i+1}\tau_{i+1} \subseteq_L G'_{i+1}. \quad \square$$

Theorem 2.1
Let P be a program and G a goal. If every SLD derivation of G in P is finite independently of the used selection rule, then every RSLD derivation of G in P is finite too.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Proof
Suppose that an infinite RSLD derivation of G in P exists. By Lemma 2.1, an infinite SLD derivation of G in P also exists, which contradicts the hypothesis. □

2.2 EVR_L loop check completeness

The termination issue of a query to a logic program has attracted much attention over the past few years, both in the logic programming field, and in the deductive database field (see De Shreye and Decorte (1994) for a survey).

A well known approach to the termination problem of a query in a logic program consists in modifying the computation mechanism by adding a capability of pruning, i.e. at some point the interpreter is forced to stop its search through a certain part of the SLD tree (Apt, Bol and Klop, 1989; Bol, 1992; Bol, Apt and Klop, 1991; Pacini and Sessa, 2000; Smith, Genesereth and Ginsberg, 1986; Van Gelder, 1987). These mechanisms are called *loop checks*, as they are based on discovering some kinds of repetitions in derivation paths. The purpose of a loop check is to reduce the search space for top-down interpreters in order to prune infinite derivations, without loss of results of the refutation process. Thus, two basic properties are considered for loop checks. The *completeness* property of a loop check concerns the capability of pruning every infinite derivation. In contrast, the *soundness* property has to do with the preservation of computed answer substitutions.

Different forms of loop checking are considered in literature. A systematic analysis of loop checking for SLD resolution is given in Bol, Apt and Klop (1991). *Simple loop checks* have deserved special interest, because the decision of pruning does not depend on the logic program we are confronted with. The more immediate form of simple and sound loop check is the so called *Equality Variant of Resultant* check, which requires the detection of equal (up to renaming) resultants in the derivation. Such a loop check is formulated with respect to RSLD derivations in the following Definition 2.3 which recalls the essence of the analogous Definition 3.19 in Ferrucci, Pacini and Sessa (1995). The notation ($F =_L G$) is used, which means that the goal F is equal to G, where the goals are regarded as lists.

Definition 2.3 (*Equality Variant Check for Resultants*)
An RSLD derivation

$$G_o >>^{\alpha_o} N_o \overset{c_o\xi_o,\theta_o}{\longrightarrow} G_1...G_{h-1} >>^{\alpha_{h-1}} N_{h-1} \overset{c_{h-1}\xi_{h-1},\theta_{h-1}}{\longrightarrow} G_h >>^{\alpha_h} N_h...$$

is *pruned* by *Equality Variant of Resultant as Lists* loop check (EVR_L in the following), if for some i and j, with $0 \preceq i < j$, a renaming τ exists such that:

(i) $G_o\theta_o...\theta_{j-1} = G_o\theta_o...\theta_{i-1}\tau$,

(ii) $N_j =_L N_i\tau$.

Given an RSLD tree T, the application of EVR_L yields a prefix Tp of T which is obtained in this way. The descendants of a node are thrown away iff the derivation associated with the path from the root to the node is pruned.

Any couple $Rs_h = [N_h, G_o\theta_o...\theta_{h-1}]$ is a *reduced resultant*. Given two resultants

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

$Rs_j = [N_j, G_o\theta_o ... \theta_{j-1}]$ and $Rs_i = [N_i, G_o\theta_o ... \theta_{i-1}]$, for which requirements (i) and (ii) of Definition 2.3 hold, we will write $Rs_i \cong_L Rs_j$. In other words, Definition 2.3 expresses that EVR_L check is based on detecting that a resultant is obtained which is related by $\cong_L$ to a preceding one in the same derivation. It is worth noting that the relationship $\cong_L$ is an equivalence relationship. It is evident that, if reduction of resolvents is always ineffective (i.e. $G_j = N_j$, for any j), the usual EVR_L loop check for SLD derivations is found again. It is well known that EVR_L is a sound loop check in the case of SLD resolution. The soundness of EVR_L is extended to the more general case of RSLD by Theorem 4.1 of Ferrucci, Pacini and Sessa (1995).

Let us observe that if we do not consider condition (i) in Definition 2.3 we obtain the EVG_L loop check which is based on detecting that a resolvent is obtained which is a variant of a preceding one in the same derivation. It is worth noting that EVG_L is a *weakly sound* loop check, in sense that it preserves at least a successful, but it does not ensure the preservation of the computed answer substitutions (Bol, Apt and Klop, 1991).

The completeness of a loop check is usually referred to given selection rules and classes of programs. A loop check is complete for a program P with respect to a selection rule S if all infinite derivations of P via S are pruned. A loop check is complete for a class C of programs, if it is complete for every program in C. Several classes of logic programs are characterised in literature for which complete loop checks can be found. Actually, most of them are classes of function free programs, i.e. programs whose clauses contain no function symbol (Bol, 1992; Bol, Apt and Klop, 1991; Ferrucci, Pacini and Sessa, 1995; Pacini and Sessa, 2000). In the following part of this section, and later in Section 6, we consider the problem of preserving the completeness of EVR_L check, passing from SLD to RSLD resolution, in the case of function free programs.

Let us first show how the completeness of equality loop checks, with respect to a given selection rule, can be lost passing from SLD to RSLD. Indeed, it is sufficient reconsider Example 2.2. In that case EVR_L loop check is obviously complete, since no infinite SLD derivation exists. On the other hand, it is obvious that EVR_L loop check cannot prune the infinite RSLD derivation developed in the same example, because the length of resolvents increases at each derivation step. Actually, it is immediate to verify that the infinite derivation in Example 2.2 cannot even be pruned by using more complex and powerful checks (like SIR_M) which are based on *subsumption* relationships between resultants (Bol, Apt and Klop, 1991).

Now we prove that EVR_L loop check completeness is preserved for function free programs, in the case that EVR_L is complete with respect to all selection rules. Precisely, Theorem 2.2 states that, if EVR_L prunes every infinite SLD derivation of a goal G in a function free program P, then EVR_L prunes also every infinite RSLD derivation of G in P. In order to show this result, let us provide a condition which holds whenever EVR_L prunes every infinite derivation of G in P. Lemma 2.2 states that, if EVR_L check prunes all infinite derivations of G in P, then the length of resolvents in all possible derivations is limited. In the proof of Lemma 2.2 we exploit the notion of S-tree (Apt and Pedreschi, 1993). Given a program P and a goal G, an S-*tree* of G in P is a tree where the descendants of a goal are its resolvents with

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

respect to all selection rules and all input clauses. In other words, an S-tree groups all SLD derivations of G in P. The notation $\#R$ represents the number of atoms in the goal R.

Lemma 2.2
Let P be a program and G a goal. Suppose that all infinite SLD derivations of G in P are pruned by EVR_L. Then, a finite bound l exists such that, for each resolvent R in any SLD derivation of G in P, it is $\#R \preceq l$.

Proof
Let T be an S-tree of G in P. Given a node n in T, let $Dr(n)$ denote the derivation associated to the path from the root of T to n, and $R(n)$ the final resolvent of $Dr(n)$. Then, let Tp be the prefix of T which is obtained by applying the EVR_L check to T, i.e. the prefix where the descendants of any node n of T are thrown away if and only the derivation $Dr(n)$ is not pruned by EVR_L. By hypothesis, all infinite SLD derivations of G in P are pruned by EVR_L, which means that Tp has no infinite path. As a consequence, since T is a finitely branching tree, by Konig's lemma (see Theorem K in Knuth (1997)) the prefix Tp is finite. Now, let d be the depth of Tp, and l the maximum of the set $\{\#R(n)|\ n$ is a node in Tp$\}$. We prove that:

$\#R(n) \preceq l$, for any node n in T.

The proof is by induction on the value of $depth(n)$. For $depth(n) \preceq d$ the thesis is trivial. Then consider an integer $h > d$, and suppose that $\#R(n') \preceq l$, for any node n' with $depth(n') < h$. Given a node n of T such that $depth(n) = h$, we show that also $\#R(n) \preceq l$ holds. Since $n \notin$Tp, the derivation $Dr(n)$ is pruned by EVR_L, so that two nodes n_1 and n_2 exist in the path from the root of T to n with:

– $depth(n_1) < depth(n_2)$, (1)

– $R(n_2)$ is a variant of $R(n_1)$. (2)

Now, consider the sequence of clauses which has determined the path from n_2 to n in T. Since T contains all SLD derivations of G in P, the same derivation steps can be repeated in T starting from n_1. As a consequence, by (1) and (2), a path from n_1 to a node n' exists such that:

– $depth(n') = depth(n) - (depth(n_2) - depth(n_1)) < depth(n) = h$,

– $R(n')$ is a variant of $R(n)$.

By inductive hypothesis it is $\#R(n') \preceq l$. But $R(n')$ is a variant of $R(n)$, so that $\#R(n) = \#R(n') \preceq l$.

In conclusion, the thesis holds for every node n in T. □

Theorem 2.2
Let P be a function free program and G a goal. If EVR_L prunes every infinite SLD derivation of G in P independently of the used selection rule, then EVR_L prunes every infinite RSLD derivation of G in P.

Proof
Let D be an infinite RSLD derivation of G in P. By Lemma 2.1, an SLD derivation D' of G in P also exists such that every reduced resolvent of D is included in a

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

resolvent of D' (up to renamings). Since EVR_L prunes every infinite SLD derivation of G in P, by Lemma 2.2 the length of resolvents of D' is limited. Then, the length of reduced resolvents and resultants of D is also limited. Now, since the language of P is function free and has finite many predicate symbols and constants, the relationship denoted by $\cong_L$ has only finitely many equivalence classes on resultants of D. As a consequence, for some $0 \preceq i < k$ we have that the k^{th} and the i^{th} resultants of D are in $\cong_L$ relationship. This implies that D is pruned by EVR_L. □

In this section, redundancy elimination tolerance has been proven on the basis of a rather strong hypothesis, i.e. termination and completeness of loop checking for all possible selection rules. In Section 3 we will introduce a new computational model which will allow us to characterise a class of selection rules which are shown to be redundancy elimination tolerant. As a matter of fact, in Section 6 we will prove that program termination and EVR_L loop check completeness are maintained for that class of rules, passing from SLD to RSLD.

3 Priority scheduling rules

As shown in Section 2, redundancy elimination can determine missing termination and loop check detection. This fact depends critically on the used selection rule, because redundancy elimination can affect the actual sequence of atom rewriting. As a matter of fact, it is widely acknowledged that the analysis of interdependence between derivation processes and the used selection rules is a difficult task. In our study, the necessary insights have been provided by a computation model which is based on a novel mechanism of atom choice, which works in terms of *scheduling rules* rather than in terms of conventional selection rules. Through this new computational model, a class of scheduling rules is identified in Section 4, which is *redundancy elimination tolerant* in the sense that no loss in termination and/or loop detection comes out, passing from SLD to RSLD.

We start the analysis with an observation about selection rules, as they are normally conceived in literature and used in practice. In SLD derivations, resolvents are usually regarded as lists, nevertheless selection rules are given complete free choice ability of the atom to rewrite. In this sense, two different philosophies are superimposed, because a scheduling (i.e. an ordering) must coexist with an atom choice which can actually overcome the scheduling. Now, in the case that resolvents are viewed as unstructured multisets instead of lists, the obvious solution is that a free choice ability is provided at rewriting time. However, if scheduling policies (i.e. an ordering or a priority assignment) are exploited, it may appear natural that priorities are obeyed at rewriting time, so that the atom with optimum priority is always selected. Indeed, if a scheduling policy is used, the moment of addition of new atoms in the resolvent may be recognised as the really important event, when suitable priority values must be established and assigned.

In the following, we consider execution mechanisms for logic programs which are based on priority scheduling policies. In particular, we characterise *scheduling rules* informally as follows:

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

- a priority value is assigned to each atom in the actual resolvent,
- assigned priorities are not modified in the following of the derivation,
- the atom with optimum priority is always taken for rewriting.

In essence a scheduling rule is a rule that defines a priority values for any new atom which enters the actual resolvent. It is crucial that atoms from the body of the applied clause can be freely scheduled with respect to the ones already present in the resolvent, which maintain their own priority values. It is intuitive that this can be easily done if a set of 'dense' priority values is adopted. Indeed, as formalised in Section 3.1, we will use rational numbers as priority values.

Now, in analogy with Lloyd's definition of selection rules (Lloyd, 1987), we consider the subclass of scheduling rules where the schedule of new atoms is determined only by the last resolvent in the derivation, i.e. by the *current state* of the computation. Such rules will be named state *scheduling rules*. A state scheduling rule can be seen as a rule which, for any resolvent G and clause c (that is applied to the optimum priority atom), determines the schedule positions of the new atoms in the resolvent, through the assignment of appropriate priority values.

In other words, a state scheduling rule determines new resolvents, starting from the old ones and from applied clauses. The rewritten atom is necessarily the one with the optimum priority value. It is evident that the transformation from a resolvent to a new one, which is obtained by the addition of new atoms from the applied clause, is nothing more than a step of an SLD derivation. In this sense, we can say that a state scheduling rule characterises a set of derivation steps. Indeed, as formalised in Section 3.5, a state scheduling rule can be straight conceived as a *set of derivation steps*, that is: the set of derivation steps which are allowed according to the scheduling rule itself. Formal definition of state scheduling rules is provided in Section 3.5.

3.1 Atoms, goals and priorities

To characterise *state scheduling rules* in a formal way, we introduce the notions of *priority goal* and *priority clause*. A priority goal is a goal where each atom has an associated priority value. Thus, a priority goal G can be thought as a set of couples, where any couple is named *priority atom*. In the following formal definition, priority atom will be denoted by $a[p]$, where a is an usual atom and p is a rational number which establishes the priority of a in G. The symbol $\Longrightarrow$ will be frequently used in the rest of the paper to denote logical implication.

Definition 3.1

(i) A *priority goal* G (*p-goal* in the sequel) is defined by a set of *priority atoms* (or simply *p-atoms*) of the form:

$G = \{a_1[p_1], ...a_k[p_k]\}$, with $\forall i, j : i \neq j \Longrightarrow p_i \neq p_j$,

where each a_m is an usual atom and each p_m is a rational number, $1 \preceq m \preceq k$.

(ii) A *priority clause* (or simply a *p-clause*) has the form $c = ht \longleftarrow B$, where ht is an atom (without priority) and B is a priority goal.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

In the sequel, priority clauses will be referred as clauses for the sake of simplicity. Capital letters will be used in the following to represent p-goals. To denote p-atoms, we will use notations like $a[p]$, as well as simple small letters (as a, b, etc.) when explicit reference to priority values is not important. As a slight abuse of notation, p-goals made of only one p-atom a will be often denoted by a. Given a p-goal G, the notations $\#G$ will indicate the number of p-atoms in G.

In the sequel, we will exploit very frequently a basic operation which corresponds to the union of two p-goals with no common priority values. This operation is denoted by '+' and is said p-goal *merging*. During merging operations, atoms retain their priority values. We introduce also the idea of *concatenation*, which is a particular case of merging. Concatenations will be denoted by the symbol '|' (vertical bar). The following are the formal definitions of merging and concatenation. It is worth noting that both these operations are associative.

Definition 3.2

(i) A p-goal M is the *merging* of F and G (denoted by $M = F+G$) if F and G have no common priority values and $M = F \cup G$.

(ii) Given two p-goals F and G, we write $F \dashv G$ to denote that all priorities in F are less than any priority in G. A p-goal N is the *concatenation* of F and G (denoted by $N = F|G$), if $N = F + G$ and $F \dashv G$.

The fact that equal priority values are not admitted in a p-goal has two principal effects. The first one is that a complete ordering (i.e. a scheduling) is imposed on the atoms of a p-goal. In particular we assume that atoms with less priorities precede atoms with greater ones. The second effect is that possible multiple occurrences of atoms are distinguished by different priority values. On the basis of the above observations, the following evident properties of concatenation can be stated.

Property 3.1

Given the p-goals A_1, A_2, A_3, B_1, B_2, and B_3, the following propositions hold:

(i) $A_1|A_2 = B_1|B_2,\ \#A_1 = \#B_1 \quad \text{or} \quad \#A_2 = \#B_2 \Longrightarrow A_1 = B_1,\ A_2 = B_2.$

(ii) $A_1|A_2|A_3 = B_1|B_2|B_3,\ A_2 \neq \emptyset,\ A_2 = B_2 \Longrightarrow A_1 = B_1,\ A_3 = B_3.$

3.2 Shifting and positioning

Throughout the paper, we will exploit a basic operator for handling priority values. It will be called (*priority*) *shifting*, and corresponds to a modification of priority values which does not alter the scheduling of the atoms in a p-goal. The following is the formal definition of shifting. In the sequel, shiftings will be always denoted by underlined Greek letters.

Definition 3.3 (*shifting*)

A *shifting* $\underline{\pi}$ is an increasing one-to-one application of the type:

$\underline{\pi}$: Rational $\longrightarrow$ Rational.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Given a *shifting* $\underline{\pi}$, and two p-goals G and F such that:

$G = \{a_1[p_1], ...a_k[p_k]\}$ and $F = \{a_1[\underline{\pi}(p_1)], ...a_k[\underline{\pi}(p_k)]\}$,

we say that F is a *shifting* of G and write $F = G\underline{\pi}$.

It is evident that the composition of two shiftings is a shifting, too, as well as the inverse of a shifting. Shifting operations enjoy the following four basic properties. All properties are plain consequence of the definition. The first two properties will be used very often in the sequel without explicit reference.

Property 3.2

(Ax-i) $(A_1 + A_2 + ... + A_k)\underline{\pi} = A_1\underline{\pi} + A_2\underline{\pi} + ... A_k\underline{\pi}$,

(Ax-ii) $(A_1|A_2|... A_k)\underline{\pi} = A_1\underline{\pi}|A_2\underline{\pi}|... A_k\underline{\pi}$,

(Ax-iii) $G = A_1\underline{\tau}_1|A_2\underline{\tau}_2|... A_k\underline{\tau}_k, \quad F = A_1\underline{\pi}_1|A_2\underline{\pi}_2|... A_k\underline{\pi}_k$
$\Longrightarrow \exists\ \underline{\sigma}$ *such that* $F\underline{\sigma} = G$,

(Ax-iv) $(A_1 + A_2 + ... A_k)\underline{\pi} = (A_1 + A_2 + ... A_k)\underline{\tau}$
$\Longrightarrow A_1\underline{\pi} = A_1\underline{\tau}, \ A_2\underline{\pi} = A_2\underline{\tau}, \ ... \ A_k\underline{\pi} = A_k\underline{\tau}$.

Finally, let us consider a combination of shifting and merging which provides the convenient tool to formalise our ideas about scheduling of atoms in resolvents. As outlined in previous section, at any step of derivation, atoms coming from the body of the applied clause are assigned new priority values, while priorities of old atoms are left unchanged. This way, new atoms are positioned (i.e. scheduled) with respect to the old ones. In general, the *positioning* of atoms from a p-goal B, with respect to the atoms of another p-goal F, can be described through a composition of shifting and merging. Indeed, consider an expression like $F + B\underline{\pi}$. The effect of the shifting $\underline{\pi}$ is twofold. First, possible conflicts of priority values between F and B can be removed, so that the merging $F + B\underline{\pi}$ is correctly performed. At the same time, yet more important, $\underline{\pi}$ allows us to establish the positions which atoms from B go to occupy. Since priorities are represented by rational values, it is evident that all possible allocations of atoms from B, with respect to those in F, can be described through suitable choices of $\underline{\pi}$.

3.3 Priority SLD Derivations

Now we are ready to frame well known Logic Programming concepts, as the ones of resolvent and SLD derivation, in terms of priority atoms, goals and scheduling. We start with the following Definition 3.4, which formalises the idea of *priority derivation step*. Given a p-goal $a|F$, in agreement with our concept of scheduling the atom a with minimum priority is always rewritten and atoms coming from the body of the applied clause are positioned with respect to old ones to form the new resolvent. The positioning is obtained through a combination of shifting and merging, as discussed at the end of the previous Section 3.2. With reference to Definition 3.4, the body B of the applied clause is first shifted by $\underline{\pi}$ and then merged with F, i.e. with the initial p-goal $a|F$ minus the rewritten atom.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Definition 3.4 (*priority derivation step*)
Consider a p-goal $G = a|F$ and a clause $c = (ht \longleftarrow B)$. Let:

– ξ be a renaming such that $var(G) \cap var(c\xi) = \emptyset$,

– θ be an idempotent and relevant mgu of a and $(ht)\xi$,

– $\underline{\pi}$ be a shifting such that F and $B\underline{\pi}$ have no common priority value.

We say that R is a *resolvent* of G and c by ξ, θ and $\underline{\pi}$, if:

$R = (F + B\xi\underline{\pi})\theta$.

The transformation from $a|F$ to $(F + B\xi\underline{\pi})\theta$ will be called a *priority derivation step*. It is denoted by:

$a|F \stackrel{c}{\longrightarrow} (F + B\xi\underline{\pi})\theta$.

The notation $G \stackrel{c\xi,\theta}{\longrightarrow} R$ will be used to represent a derivation step by θ and ξ, where the shifting $\underline{\pi}$ is not pointed out. Analogously, we will write $G \stackrel{c\xi}{\longrightarrow} R$ to represent a derivation step by the renaming ξ without specifying the mgu θ. By $G \stackrel{c}{\longrightarrow} R$ we denote a derivation step which generically produces R as a resolvent of G and c. Iterating the process of computing resolvents, we obtain a priority SLD derivation, that is a sequence of priority derivation steps as formalised by the following definition.

Definition 3.5 (*priority SLD derivation*)
Let P be a program and G_o a p-goal. A *priority SLD derivation* of G_o in P is a possibly infinite sequence of priority derivation steps

$G_o \stackrel{c_o\xi_o,\theta_o}{\longrightarrow} G_1 \longrightarrow ... \; G_k \stackrel{c_k\xi_k,\theta_k}{\longrightarrow} G_{k+1} \longrightarrow ...$

where, for any $j \geq 0$,

(i) c_j is a clause in P,

(ii) $var(c_j\xi_j) \cap (var(G_o) \cup var(c_o\xi_o) \cup ... \cup var(c_{j-1}\xi_{j-1})) = \emptyset$.

Given a finite priority SLD derivation (p-SLD *derivation* in the following) of the form:

$G_o \stackrel{c_o\xi_o,\theta_o}{\longrightarrow} G_1 \longrightarrow ...G_h \stackrel{c_h\xi_h,\theta_h}{\longrightarrow} G$,

the sequence $M = c_1, c_2, ...c_h$ of applied clauses will be called *template*. The whole derivation will be denoted by $G_o \stackrel{M,\theta}{\longrightarrow} G$, where $\theta = \theta_1\theta_2...\theta_h$, or simply $G_o \stackrel{M}{\longrightarrow} G$, if the substitution θ does not need to be pointed out. We use the notation $G_o \stackrel{M}{\longrightarrow} \bullet$, when there is not interest in specifying the final resolvent. Given a template M, the notation $\#M$ will indicate the number of clauses in M. In many cases, we will consider concatenation of templates, which is denoted by a vertical bar '|'.

It is intuitive that, given a derivation, any subset of atoms in the current resolvent *derives from* other specific atoms in preceding resolvents. As it will be clear in the sequel, this idea plays an important role in the development of this paper. Thus, it is convenient to give some formal definitions. Precisely, let us consider a p-SLD derivation of the form $Dr = (F + G \stackrel{H}{\longrightarrow} Q)$. The following two intuitive concepts will be characterised:

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

(a) the *sub-resolvent of F in Dr*, i.e. the subset of p-atoms in Q which derive from the subgoal F (denoted by Q/F),

(b) the *sub-template of F in Dr*, i.e. the sequence of clauses which are applied to p-atoms of F and p-atoms derived from F, extracted in the order from the template H (denoted by H/F).

Definition 3.6 (*sub-resolvents and sub-templates*)

(i) Given a derivation step of the following form, where $c = (ht \longleftarrow B)$:

$$a|(F + G) \xrightarrow{c} (Q = ((F + G) + B\xi\underline{\pi})\alpha), \tag{1}$$

let us define *sub-resolvents* and *sub-templates* in (1) as follows:

$$Q/a = B\xi\underline{\pi}\alpha, \quad Q/F = F\alpha, \quad Q/(a|F) = Q/a + Q/F$$
$$c/a = c, \quad c/F = \emptyset, \quad c/(a|F) = c.$$

(ii) Given a derivation of the form:

$$F + G \xrightarrow{c} Q \xrightarrow{K} R, \tag{2}$$

let us recursively define *sub-resolvents* and *sub-templates* in (2) as follows:

$$R/F = R/(Q/F),$$
$$(c|K)/F = (c/F)|(K/(Q/F)).$$

It is worth noting that the notation relative to sub-templates and sub-resolvents can be ambiguous. Indeed consider:

$$G + F \xrightarrow{D} Q \tag{3}$$

$$G + F' \xrightarrow{D} Q'. \tag{4}$$

It is possible that D/G with respect to (3) is different from D/G with respect to (4). In the following of the paper, when such a kind of ambiguity will possibly arise, we exploit a refined notation of evident meaning, like $D/^3/G$ and $D/^4/G$. As an example, let us consider $G = aF' = d$ and $D = c$ such that

$$G + F = a|b \xrightarrow{c} Q \tag{3b}$$

$$G + F' = d|a \xrightarrow{c} Q'. \tag{4b}$$

Then, $D/^{3b}/G = c$ and $D/^{4b}/G = empty$.

3.4 Congruent lowering of derivation steps

This section introduces some important ideas. Precisely, the concepts of *specialisation*, *lowering*, and finally, *congruent lowering* are defined and analysed. Congruent lowering is basic for the characterisation of the general concept of scheduling rule, as well as of the class of specialisation independent scheduling rules (see Section 4) to which the results about redundancy elimination tolerance of Section 6 refer.

Substitutions and renamings are basic concepts in Logic Programming. In agreement with usual terminology, if a substitution is applied to a goal, an *instance* is obtained, while, if a renaming is used, a *variant* of the original goal is produced. Goals which are equal up to renamings are in essence equivalent goals. Practically all the results of Logic Programming are insensible to renamings. An instance may be

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

considered as a specialised version of the original goal, while any goal is more general with respect to its instances. The above concepts are easily adjusted in the frame of priority goals. Intuitively, the application of a renaming/substitution corresponds to the application of a renaming/substitution together with a shifting. Actually, as it will be clear in the following, we are interested in an idea of *specialisation* of a given p-goal which extends the traditional concept of instantiation. In essence, we will consider couples of p-goals such that the second is obtained from the first by performing in the order:

- the application of a generic substitution λ and a shifting $\underline{\sigma}$,
- the embedding in a generic context X of other p-atoms.

Definition 3.7 (*specialisation*)
A p-goal F is a *specialisation* of a p-goal $a|K$ by X, if a shifting $\underline{\sigma}$ and a substitution λ exist such that

$$F = a\lambda\underline{\sigma}|(K\lambda\underline{\sigma} + X).$$

It is worth noting that our idea of specialisation is essentially symmetric to the concept of subsumption by an instance (see Bol, Apt and Klop, 1991). A goal G *subsumes* (as list) a goal F *by an instance*, if a substitution λ exists such that $G\lambda \subseteq_L F$. Indeed, considering that any shifting preserves the order of the atoms, it is evident that, if F is a specialisation of $a|K$ by X, i.e. $F = a\lambda\underline{\sigma}|(K\lambda\underline{\sigma} + X)$, then $a|K$ subsumes (as list) F by the instance $(a|K)\lambda$.

The term 'lifting' is used in Logic Programming to express that a derivation step (or a whole derivation) which is possible from a goal $A\lambda$ is repeated starting from the more general goal A. Analogously, we use the term lifting to mean that a derivation step (or a whole derivation) which is possible from a specialisation of $a|K$, i.e. from a p-goal $a\lambda\underline{\sigma}|(K\lambda\underline{\sigma} + X)$, is repeated starting from $a|K$. In the sequel of the paper, we will use the dual concept of 'lowering'. In other words, the term lowering will mean that a derivation step (or a whole derivation) from a p-goal $a|K$ is repeated, when possible, starting from a specialisation $a\lambda\underline{\sigma}|(K\lambda\underline{\sigma} + X)$ of $a|K$. Then, let us give the following definition which refers to single derivation steps.

Definition 3.8 (*lowering of derivation steps*)
Let us consider two priority derivation steps of the type $G \xrightarrow{c} \bullet$ and $F \xrightarrow{c} \bullet$. We will say that the second step is a *lowering of* the first one *by* X, if the p-goal F is a specialisation of G by X.

Let us consider two derivation steps (Ds1) and (Ds2), such that (Ds2) is a lowering of (Ds1) by X, and let $c = (ht \longleftarrow B)$. By definition of derivation step, they have the following form:

$$a|K \xrightarrow{c} (K + B\xi'\underline{\theta}')\alpha' \tag{Ds1}$$

$$a\lambda\underline{\sigma}|(K\lambda\underline{\sigma} + X) \xrightarrow{c} (X + K\lambda\underline{\sigma} + B\xi''\underline{\theta}'')\alpha''. \tag{Ds2}$$

The definition of lowering of derivation steps does not impose any similarity in

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

the way priority values are handled in couples of derivation steps like (Ds1) and (Ds2). In particular, no analogy is required about the positions new atoms go to occupy with respect to old ones in the resolvents produced by (Ds1) and (Ds2). Indeed the shifting $\underline{\theta}'$ and $\underline{\theta}''$ are completely independent, so that the positions of atoms of $B\xi''\underline{\theta}''$, with respect to atoms of $K\lambda\underline{\sigma}$, will be in general different from the positions occupied by atoms of $B\xi'\underline{\theta}'$ with respect to atoms of K. Nevertheless, in the rest of the paper special importance will be given to derivation step lowering such that the positioning of new atoms, with respect to the old ones in K and $K\lambda\underline{\sigma}$, is maintained passing from (Ds1) to (Ds2). In such hypothesis, we will say that the lowering is a *congruent lowering*.

As an elementary example, let us consider a clause like $c = a \longleftarrow b_1|b_2$ and the following derivation steps, such that (2) is a lowering of (1) by $x_1|x_2$:

$$a|k_1|k_2 \overset{c}{\longrightarrow} b_1'\underline{\theta}'|k_1|b_2\underline{\theta}'|k_2 \tag{1}$$

$$a|x_1|k_1|x_2|k_2 \overset{c}{\longrightarrow} x_1|b_1\underline{\theta}''|k_1|x_2|b_2\underline{\theta}''|k_2 \tag{2}$$

In (1) and (2) the relative positions of atoms b_1 and b_2 with respect to k_1 and k_2 are the same, then (2) is a congruent lowering of (1). Now, let us consider the following other derivation step (3):

$$a|x_1|k_1|x_2|k_2 \overset{c}{\longrightarrow} x_1|k_1|b_1\underline{\tau}|x_2|b_2\underline{\tau}|k_2 \tag{3}$$

Also, (3) is a lowering of (1) by $x_1|x_2$. However, in this case the positioning of atoms b_1 and b_2 with respect to k_1 and k_2 is not maintained passing from (1) to (3), so that (3) is not a congruent lowering of (1). Variable substitutions are not considered in the above examples. Indeed, in agreement with the following formal Definition 3.9, they are not really influent for a lowering to be congruent or not.

Definition 3.9 (*congruent lowering*)
Let us consider two derivation steps of the form (Ds1) and (Ds2) above, i.e. two derivation steps such that the second one is a lowering of the first one by X. We will say that step (Ds2) is a *congruent lowering* of step (Ds1) *by* X if a shifting $\underline{\rho}$ exists with:

$$K\underline{\rho} = K\underline{\sigma} \text{ and } B\underline{\theta}'\underline{\rho} = B\underline{\theta}''. \tag{c1}$$

It is apparent that the desired analogy, in positioning new atoms in the two derivation steps (Ds1) and (Ds2), is imposed by means of condition (c1) above in Definition 3.9. Indeed, condition (c1) says that the shifting $\underline{\rho}$ creates a correspondence between atoms of $K + B\underline{\theta}'$ and atoms of $K\underline{\sigma} + B\underline{\theta}''$, such that old atoms are mapped in old atoms (see $K\underline{\rho} = K\underline{\sigma}$) and new atoms in new ones (see $B\underline{\theta}'\underline{\rho} = B\underline{\theta}''$). Since any shifting maintains atom precedence, it is intuitive that congruent allocation of new atoms is imposed. More specifically, let us consider the generic atom b of B and assume:

$$K + B\underline{\theta}' = M'|b\underline{\theta}'|N',$$
$$K\underline{\sigma} + B\underline{\theta}'' = M''|b\underline{\theta}''|N''.$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

It is immediate to verify that[1]:

$$M''|b\underline{\theta}''|N'' = K\underline{\sigma} + B\underline{\theta}'' =^{(c1)} K\underline{\rho} + B\underline{\theta}'\underline{\rho} = (K + B\underline{\theta}')\underline{\rho}$$
$$= (M'|b\underline{\theta}'|N')\underline{\rho} = M'\underline{\rho}|b\underline{\theta}'\underline{\rho}|N'\underline{\rho}.$$

Now, by $B\underline{\theta}'\underline{\rho} = B\underline{\theta}''$ in (c1) and Ax-iv in Property 3.2, we have that $b\underline{\theta}'\underline{\rho} = b\underline{\theta}''$. Then, by Property (3.1-ii) it is $M'\underline{\rho} = M''$, and then also $\#M' = \#M'\underline{\rho} = \#M'' = n$, for n positive integer. In essence, considered the generic atom b of B, it is found in the $(n+1)$th position in $K + B\underline{\theta}'$ as well as in $K\underline{\sigma} + B\underline{\theta}''$. In other words, new atoms from B are positioned in (Ds1) with respect to old ones (i.e. atoms of K) exactly as it happens in (Ds2) with respect to $K\underline{\sigma}$. It is evident that the presence of various substitutions in (Ds1) and (Ds2) does not interfere with the above positional considerations.

Example 3.1 (*lowering and congruent lowering*)
Let us consider a clause of the form $c = (a \longleftarrow q[1])$ and the two following derivation steps:

$$a[2]|\{b[\mathbf{3}]\} \xrightarrow{c} \{b[\mathbf{3}], q[\underline{10}]\}, \tag{1}$$

$$a[9]|\{b[12], b[13], d[15]\} \xrightarrow{c} \{b[12], q[12.5], b[13], d[15]\}. \tag{2}$$

In step (1), old atoms are pointed out in bold and new ones are underlined.

(a) In agreement with Definition 3.9, step (2) is a lowering of (1) by $X = \{b[13], d[15]\}$, with $K\underline{\sigma} = \{b[12]\}$. Pointing out old and new atoms, derivation step (2) can be written as follows:

$$a[9]|\{b[\mathbf{12}], b[13], d[15]\} \xrightarrow{c} \{b[\mathbf{12}], q[\underline{12.5}], b[13], d[15]\}.$$

It is evident that (2) is a congruent lowering of (1) by X, with any shifting $\underline{\rho}$ such that $\underline{\rho} \supseteq \{3/12, 10/12.5\}$.

(b) Step (2) is a lowering of (1) also by $X' = \{b[12], d[15]\}$, with $K\underline{\sigma}' = \{b[13]\}$. However (2) is not a congruent lowering of (1) by X'. In fact, in agreement with this second viewpoint, derivation step (2) can be written as follows:

$$a[9]|\{b[12], b[\mathbf{13}], d[15]\} \xrightarrow{c} \{b[12], q[\underline{12.5}], b[\mathbf{13}], d[15]\}.$$

As a consequence, for step (2) being a congruent lowering of step (1) by X', a shifting $\underline{\rho}'$ might exist such that $\underline{\rho}' \supseteq \{3/13, 10/12.5\}$, which is not an increasing function. □

We close this section considering a couple of p-goals F and G such that they are specialisations of each other, i.e. F is a specialisation of G by a subgoal X and G is a specialisation of F by Y. In this case it must be $F = G\lambda\underline{\sigma} + X$ and $G = F\tau\underline{\rho} + Y$, which yields:

$$G = F\tau\underline{\rho} + Y = (G\lambda\underline{\sigma} + X)\tau\underline{\rho} + Y = G\lambda\tau\underline{\sigma}\underline{\rho} + X\tau\underline{\rho} + Y.$$

As a consequence λ must be a renaming for G and $X = Y = \varnothing$ must hold, which means that $F = G\lambda\underline{\sigma}$ where λ is a renaming. It is evident that the relation '$F = G\lambda\underline{\sigma}$,

[1] The notation "$K\underline{\sigma} + B\underline{\theta}$" $=^{(c1)}$ $K\underline{\rho} + B\underline{\theta}'\underline{\rho}$" expresses that the formula (c1) must be used to establish the equality. Similar advising will be used frequently in the sequel.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

for a renaming λ and a shifting $\underline{\sigma}$' can be seen as the translation of the usual notion of 'F being variant of G' in the frame of p-SLD resolution. In this sense, we will usually say that F is a *p-variant* of a G, to mean that F and G are specialisations of each other.

Analogously, two derivation steps may be *lowerings of each other*, as well as *congruent lowerings of each other*. Two derivation steps Ds_1 and Ds_2 are lowerings of each other if the initial goals are p-variants and the same clause is applied, i.e. it is $Ds_1 = (A \xrightarrow{c} \bullet)$ and $Ds_2 = (A\lambda\underline{\sigma} \xrightarrow{c} \bullet)$, where λ is a renaming. Two derivation steps are congruent lowerings of each other if they have the form:

$$Ds_1 = a|K \xrightarrow{c} (K + B\xi'\underline{\theta'})\alpha' \text{ and } Ds_2 = (a|K)\lambda\underline{\sigma} \xrightarrow{c} (K\lambda\underline{\sigma} + B\xi''\underline{\theta''})\alpha'',$$

where $c = (ht \longleftarrow B)$, λ is a renaming, and the equalities $K\underline{\rho} = K\underline{\sigma}$ and $B\underline{\theta'}\underline{\rho} = B\underline{\theta''}$ hold for a shifting $\underline{\rho}$.

It is worth noting that by the preceding argument if two derivation steps are lowerings of each other the contexts must be empty.

3.5 State priority scheduling rules

Now, we use the notion of being congruent lowerings of each other to define the ideas of *determinism* and *completeness* of a set of derivation steps. Both concepts are basic for the definition of state priority scheduling rules.

Definition 3.10 (*determinism*)
A set S of priority derivation steps is *deterministic* if, for each couple of derivation steps Ds_1 and Ds_2 in S, the following implication holds:

Ds_1 and Ds_2 are lowerings of each other

$\Longrightarrow$ Ds_1 and Ds_2 are congruent lowerings of each other.

In other words, the definition of determinism imposes that two derivation steps, which apply the same clause to p-variant initial goals, give place to congruent allocations of new atoms. Now let us give the definition of completeness of a set of derivation steps.

Definition 3.11 (*completeness*)
A set S of priority derivation steps is *complete*, if the following assertions hold:

(i) $\exists$ Ds derivation step of the type $G \xrightarrow{c} \bullet$,

$\Longrightarrow$ $\exists$ Ds' of the type $G \xrightarrow{c} \bullet$ with $Ds' \in S$,

(ii) $\forall Ds', Ds$ derivation steps with $Ds \in S$,

Ds' and Ds are congruent lowerings of each other $\Longrightarrow$ $Ds' \in S$.

Assertion (i) of the above definition states that, if a clause c is applicable to a p-goal G, i.e. a derivation step exists of the type $G \xrightarrow{c} \bullet$, the application of the clause c to G is indeed possible in any complete set of derivation steps. Assertion (ii) assures that S is closed with respect to being congruent lowerings of each other. In other words, let $Ds' = (G \xrightarrow{c} Q) \in S$ be a derivation step, then every

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

other $Ds'' = (F \xrightarrow{c} R)$ must belong to S, if F is a p-variant of G and new atoms are allocated in R as it is done in Q. Now, the formal definition of state priority scheduling rules can be easily given, by combining the properties of determinism and completeness.

Definition 3.12 (*state priority scheduling rules*)
A *state priority scheduling rule* is a complete and deterministic set of priority derivation steps.

It can be easily verified that the leftmost selection rule, adopted by the Prolog execution mechanism, is a state priority scheduling rule. The very nature of a state scheduling rule is characterised by the following Definition 3.13. Indeed, the definition simply says that a p-SLD derivation *is via* a state scheduling rule S if all derivation steps are admitted in the rule S, i.e. they all belong to the set of derivation steps which S is constituted by.

Definition 3.13 (*derivations via S*)

(i) Given a set S of derivation steps, the notation $\Delta(S)$ represents the whole of p-SLD derivations which are composed of derivation steps in S.

(ii) Given a state scheduling rule S, the set $\Delta(S)$ is the set of *p-SLD derivations via S*.

In the sequel of the paper we only consider state priority scheduling rules, which therefore will be called just *scheduling rules*. The following notations will be used frequently. Given a set S of derivation steps, a clause c and a template M, we will denote by

$$G \xrightarrow{S,c} R \quad \text{and} \quad G \xrightarrow{S,M} R$$

the fact that the derivation step $(G \xrightarrow{c} R) \in S$ and the p-SLD derivation $(G \xrightarrow{M} R) \in \Delta(S)$, respectively. In the case that the exploited logic program must be pointed out, a notation like

$$(G \xrightarrow{S,M,P} R)$$

will be used to specify that the derivation is via S in the program P, i.e. every clause of the template M belongs to P. The notion of p-SLD tree via S could be characterised in complete analogy with the usual one of SLD tree.

Let us close this section with a property, which can be easily shown on the basis of completeness and will be used several times in the sequel. Property 3.3 asserts that if a clause c can be applied to a p-goal $a\gamma|G$, every complete set of derivation steps allows c to be applied to any p-goal of the form $a|F$. Since the atom a is more general than $a\gamma$, the property may also be interpreted as a sort of lifting of derivation steps. However, the subgoals G and F are left unrelated at all. The evident explanation is that they have no active role in rewriting operations. Moreover, the property recalls that new variables can be always chosen so that conflicts are avoided with arbitrary pre-established sets of variables. The formal proof of this rather intuitive property can be found in Appendix A.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

In the statement of Property 3.3 and in the sequel of the paper, given a p-SLD derivation Dr, the notation $nvar(Dr)$ will represent the set of standardisation apart variables which are introduced during the derivation Dr. In the case of a single derivation step $Ds = (A \xrightarrow{c\xi} \bullet)$, it is $nvar(Ds) = var(c\xi)$.

Property 3.3
Let S be a complete set of derivation steps. Given two p-goals $a\gamma\underline{\tau}|G$ and $a|F$, let us fix arbitrarily a finite set V of variables. The following implication holds:

$\exists Ds$ derivation step of the type $a\gamma\underline{\tau}|G \xrightarrow{c} \bullet$

$\Longrightarrow$ $\exists Ds'$ of the type $a|F \xrightarrow{c} \bullet$, with $Ds' \in S$ and $nvar(Ds') \cap V = \emptyset$.

4 Specialisation independent scheduling rules

Now, we will exploit the notion of congruent lowering in order to introduce the concept of *specialisation independence*. This concept will be used to characterise the class of scheduling rules that are the main object of the paper (*specialisation independent scheduling rules*). In fact, all our results for termination and loop check completeness preserving will refer to such a class of scheduling rules. In Section 5, a second characterisation of the same class is given which has an operational nature and is surprisingly different in appearance.

The definition of *specialisation independence* enforces the idea of determinism. Indeed, in agreement with the Definition 4.1 below, every lowering is required to be a congruent lowering. In other words, the congruence in the allocation of new atoms must hold any time the initial goals of two derivation steps are related by specialisation and the same clause is used. This can be interpreted saying that the positioning of new atoms with respect to old ones is *independent of goal specialisation*, which means independent of goal instantiation as well as of the addition of a group X of other atoms.

Definition 4.1 (*specialisation independence*)
A set S of priority derivation steps is *specialisation independent* if, for every couple of steps Ds_1 and Ds_2 in S, the following implication holds:

Ds_2 is a lowering of Ds_1 by X

$\Longrightarrow$ Ds_2 is a congruent lowering of Ds_1 by X.

Definition 4.2 (*Specialisation independent scheduling rules*)
A *specialisation independent scheduling rule* is a complete and specialisation independent set of priority derivation steps.

In the next two sections, we provide some results about p-SLD derivations via specialisation independent scheduling rules. The results will be frequently exploited in the sequel.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

4.1 Derivation lowering

In this section we give results which relate resolvents coming from a couple of derivation steps in the congruent lowering relationship. Then, by Lemma 4.1, the analysis is extended to couples of whole derivations, developed via specialisation independent scheduling rules. We start by presenting a preliminary statement (Property 4.1) which holds for every couple of derivation steps that are in the lowering relationship. In reference to derivation steps (1) and (2) below, the preliminary property says that, if we abstract from atom positioning and ignore the additional subgoal X, the resolvent of (2) is an instance of the resolvent of (1). Property 4.1 can be shown following the line exploited for proving the Variant Lemma (see Apt, 1990), which is done in Appendix A for the sake of completeness of the paper.

Property 4.1
Let $c = (ht \longleftarrow B)$ be a clause. Let us consider two derivation steps like (1) and (2), where (2) is a lowering of (1) by X. The following implication holds:

$$a|K \xrightarrow{c\xi'} (K + B\xi'\underline{\theta'})\mu', \tag{1}$$

$$a\tau\underline{\sigma}|(K\tau\underline{\sigma} + X) \xrightarrow{c\xi''} (K\tau\underline{\sigma} + B\xi''\underline{\theta''} + X)\mu'' \tag{2}$$

$$\Longrightarrow \exists\delta \text{ such that } K\tau\mu'' = K\mu'\delta \text{ and } B\xi''\mu'' = B\xi'\mu'\delta,$$

where δ is a renaming, if τ is a renaming.

Property 4.2 completes Property 4.1, taking into account the preservation of atom scheduling in the case of congruent lowering. It states that, if we ignore the additional subgoal X, resolvents are preserved up to a substitution and a shifting. In reference to derivation steps (1) and (2) below, this means that, apart from R/X, the resolvent R in (2) is an instance of Q such that also atom scheduling is maintained.

Property 4.2
Let $c = (ht \longleftarrow B)$ be a clause. Let us consider two derivation steps of the type (1) and (2), such that the second one is a congruent lowering of the first one by X:

$$a|K \xrightarrow{c} Q, \tag{1}$$

$$a\tau\underline{\pi}|(K\tau\underline{\pi} + X) \xrightarrow{c} R. \tag{2}$$

The following assertion holds:

$$\exists\delta, \underline{\rho} \text{ such that } R/((a|K)\tau\underline{\pi}) = Q\delta\underline{\rho},$$

where δ is a renaming if τ is a renaming.

Proof
Let $c = (ht \longleftarrow B)$, so that Q and R may be written as follows:

$$Q = (K + B\xi'\underline{\theta'})\mu',$$

$$R = (K\tau\underline{\pi} + X + B\xi''\underline{\theta''})\mu''.$$

Since step (2) is a congruent lowering of (1) by X, a shifting $\underline{\rho}$ exists such that:

$$K\underline{\rho} = K\underline{\pi}, \quad B\underline{\theta'}\underline{\rho} = B\underline{\theta''}. \tag{3}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

By definition of sub-resolvent and (3), we have:

$$R/((a|K)\tau\underline{\pi}) = B\xi''\underline{\theta}''\mu'' + K\tau\underline{\pi}\mu'' =^{(3)} B\xi''\mu''\underline{\theta}'\underline{\rho} + K\tau\mu''\underline{\rho}. \tag{4}$$

Now, we apply Property 4.1 to (1) and (2), deriving that a substitution δ exists such that:

$$K\tau\mu'' = K\mu'\delta \quad \text{and} \quad B\xi''\mu'' = B\xi'\mu'\delta, \tag{5}$$

where δ is a renaming if τ is a renaming.

As a consequence, we have that:

$$R/((a|K)\tau\underline{\pi}) =^{(4)} B\xi''\mu''\underline{\theta}'\underline{\rho} + K\tau\underline{\rho}\mu'' =^{(5)} B\xi'\mu'\delta\underline{\theta}'\underline{\rho} + K\mu'\delta\underline{\rho} = Q\delta\underline{\rho},$$

where δ is a renaming if τ is a renaming. □

The following lemma may be seen as the extension of Property 4.2 to whole derivations, provided that the used scheduling rule is specialisation independent. Note that, given a derivation like (1) in the statement below, if a derivation like (2) exists, it can be considered as a lowering of (1). Indeed, the initial p-goal $X + G\gamma\underline{\tau}$ is a specialisation of G by X, and the sequence E of clauses is applied in the same order to atoms deriving from $G\gamma\underline{\tau}$ in derivation (2). In this sense we will regard Lemma 4.1 as a 'specialisation independent lowering lemma'.

Lemma 4.1 (*Specialisation independent lowering lemma*)
Let S be a specialisation independent scheduling rule and consider two p-SLD derivations like (1) and (2). The following implication holds:

$$G \xrightarrow{S,E} Q, \tag{1}$$

$$G\gamma\underline{\tau} + X \xrightarrow{S,D} R, \text{ with } D/(G\gamma\underline{\tau}) = E \tag{2}$$

$$\Longrightarrow \exists\sigma, \underline{\rho} \text{ such that } R/(G\gamma\underline{\tau}) = Q\sigma\underline{\rho}, \text{ where } \sigma \text{ is a renaming if } \gamma \text{ is a renaming and } D/X = \varnothing. \tag{p1}$$

Proof
Let us first prove the thesis, apart from the fact (p1). The proof is by induction on the length of D. If $\#D$ is equal to zero, the thesis is trivially true. Let us suppose that $\#D$ is greater than zero. Two different cases must be considered, i.e. the first clause of D (say c) is applied either to an atom of X or to an atom of $G\gamma\underline{\tau}$.

First case (*The clause c is applied to an atom of* X).
In this case derivation (2) may be rewritten as:

$$X + G\gamma\underline{\tau} \xrightarrow{S,c\eta,\alpha} G\gamma\underline{\tau}\alpha + X' \xrightarrow{S,D'} R, \tag{3}$$

$$\text{with } D'/(G\gamma\underline{\tau}\alpha) = D/(G\gamma\underline{\tau}) = E.$$

By inductive hypothesis, applied to the tail of derivation (3) and derivation (1), we have:

$$\exists\sigma, \underline{\rho} \text{ such that } R/(G\gamma\underline{\tau}) = R/(G\gamma\underline{\tau}\alpha) =^{(ind.hyp.)} Q\sigma\underline{\rho}.$$

Second case (*The clause c is applied to an atom of* $G\gamma\underline{\tau}$).
In in this case derivations (1) and (2) may be rewritten as (4) and (5), respectively:

$$G \xrightarrow{S,c\xi} Y \xrightarrow{S,E'} Q \tag{4}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

$$X + G\gamma\underline{\tau} \xrightarrow{S,c\eta,\alpha} X\alpha + Z \xrightarrow{S,D'} R \tag{5}$$

$$\text{with } D'/Z = E', \quad c/^4/G = c/^5/G\gamma\underline{\tau} = c. \tag{6}$$

Since S is specialisation independent, the first step of (5) is a congruent lowering of the first one of (4) by X. Then, by Property 4.2, we have:

$$\exists \sigma', \underline{\rho}' \text{ such that } Z = (X\alpha + Z)/(G\gamma\underline{\tau}) =^{(Prop.4.2)} Y\sigma'\underline{\rho}'. \tag{7}$$

As a consequence, recalling the first fact in (6), the inductive hypothesis can be applied to the tails of derivations (4) and (5). Then, we have:

$$\exists \sigma, \underline{\rho} \text{ such that } R/Z = Q\sigma\underline{\rho}. \tag{8}$$

In conclusion, we have that:

$$R/(G\gamma\underline{\tau}) = R/Z = Q\sigma\underline{\rho}.$$

In order to show the fact (p1), i.e. σ is a renaming if γ is a renaming and $D/X = \emptyset$, it is sufficient to note that:

– the 'first case' does not occur at all,
– the substitutions σ' and σ, mentioned in (7) and (8), are renamings. □

The following example shows that the hypothesis of specialisation independence is crucial for the validity of Lemma 4.1.

Example 4.1

Let us consider a scheduling rule S such that new atoms are positioned in the centre of the old resolvent. New atoms are positioned immediately before the centre if the length of the resolvent (the rewritten atom excluded) is odd. It is easy to recognise that lowering Lemma 4.1 does not hold for such a rule. Indeed, let P be the following program:

$$c1 = p(x) \longleftarrow q(x)[1]$$
$$c2 = s \longleftarrow p(b)[1].$$

Now, in reference to the statement of Lemma 4.1, let:

$$G = s[1], p(a)[2]$$
$$G\gamma\underline{\tau} = s[1], p(a)[1.5] \quad \text{and} \quad X = r[2].$$

The following are two derivations of G in P and $(G\gamma\underline{\tau} + X)$ in P, respectively:

$$\{s[1], p(a)[2]\} \xrightarrow{S,c2} \{p(b)[1], p(a)[2]\} \xrightarrow{S,c1} (\{q(b)[1], p(a)[2]\} = Q)$$

$$\{s[1], p(a)[1.5], r[2]\} \xrightarrow{S,c2} \{p(a)[1.5], p(b)[1.7], r[2]\} \xrightarrow{S,c1} (\{p(b)[1.7], q(a)[1.8], r[2]\} = R).$$

Thus, no σ and $\underline{\rho}$ can exist such that:

$$R/(G\gamma\underline{\tau}) = \{q(a)[1.8], p(b)[1.7]\} = \{q(b)[1], p(a)[2]\}\sigma\underline{\rho} = Q\sigma\underline{\rho}.$$

Note that $R/(G\gamma\underline{\tau})$ and Q are essentially different, even if they are considered as multisets abstracting from priority values. It is easy to check that the used scheduling rule is not specialisation independent, in agreement with Definition 4.2. □

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

4.2 Derivation lifting and combining

The following Lemma 4.2 is a result about p-SLD derivation lifting which is valid for specialisation independent scheduling rules. In reference to derivation (1) below, the lemma asserts that the sub-template of clauses, applied to the part $G\gamma\underline{\tau}$ of the initial p-goal $(X + G\gamma\underline{\tau})$ in (1), can be applied again in the order starting from the more general goal G, via the same scheduling rule. The lemma also recalls that standardisation apart variables can be chosen in order to avoid conflicts with any fixed finite set of variables. The lemma does not relate resolvents. Indeed, Lemma 4.1 can be exploited to this purpose.

Lemma 4.2 (*specialisation independent lifting lemma*)
Let S be a specialisation independent scheduling rule. Given any finite set V of variables, the following implication holds:

$$X + G\gamma\underline{\tau} \xrightarrow{S,D} \bullet \tag{1}$$
$$\Longrightarrow \quad \exists Dr = (G \xrightarrow{S,D/G\gamma\underline{\tau}} \bullet), \quad \text{with} \quad nvar(Dr) \cap V = \varnothing.$$

Proof
The proof is by induction on the length of the template D. If $\#D$ is zero, the assert is evident. Let us suppose that $\#D > 0$. Two cases must be considered, i.e. either the first clause in D (say c) is applied to an atom of X or the clause c is applied to an atom of $G\gamma\underline{\tau}$.

First case (*The clause c is applied to an atom of* X).

Derivation (1) may be rewritten as:

$$X + G\gamma\underline{\tau} \xrightarrow{S,c\eta,\beta} X' + G\gamma\underline{\tau}\beta \xrightarrow{S,D'} \bullet. \tag{2}$$

By inductive hypothesis applied to the tail of (2), for any finite set V of variables, a derivation Dr exists such that:

$$Dr = (G \xrightarrow{S,D'/G\gamma\underline{\tau}\beta} \bullet), \quad \text{with} \quad nvar(Dr) \cap V = \varnothing.$$

But, by construction of (2), it is $D'/G\gamma\underline{\tau}\beta = D/G\gamma\underline{\tau}$, so that the thesis is verified.

Second case (*The clause c is applied to an atom of* $G\gamma\underline{\tau}$).

Derivation (1) may be rewritten as follows:

$$X + G\gamma\underline{\tau} \xrightarrow{S,c\eta,\beta} X\beta + G' \xrightarrow{S,D'} \bullet, \tag{3}$$
$$\text{where } c|(D'/G') = D/G\gamma\underline{\tau}. \tag{4}$$

Let $G = a|Z$, that is $X + G\gamma\underline{\tau} = a\gamma\underline{\tau}|(X + Z\gamma\underline{\tau})$. By (3) and Property 3.3, we can assert that a derivation step exists like:

$$Ds' = ((G = a|Z) \xrightarrow{S,c} R'), \tag{6a}$$
$$\text{with } nvar(Ds') \cap V = \varnothing. \tag{6b}$$

Since, by hypothesis S is specialisation independent, the first step of derivation (3) is a congruent lowering of step (6a) by X. As a consequence, by Property 4.2, a substitution π' and a shifting $\underline{\rho}$' exist with:

$$G' = (X\beta + G')/(G\gamma\underline{\tau}) = R'\pi'\underline{\rho}'. \tag{7}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Then, by inductive hypothesis applied to the tail of (3), we may assert that, a derivation Dr'' exists:

$$Dr'' = (R' \overset{S,D'/G'}{\longrightarrow} \bullet) \tag{8a}$$

$$\text{with } nvar(Dr'') \cap (nvar(Ds') \cup var(G) \cup V) = \emptyset. \tag{8b}$$

So, derivation (8a) is standardised apart with respect to (6a). Since S is a state scheduling rule, (6a) and (8a) can be combined in order to give place to an unique derivation Dr such that:

$$Dr = (G \overset{c}{\longrightarrow} R' \overset{D'/G'}{\longrightarrow} \bullet) \in \Delta(S),$$

where, by (6b) and (8b), we have also that:

$$nvar(Dr) \cap V = (nvar(Ds') \cup nvar(Dr'')) \cap V = \emptyset.$$

By (4), the thesis is proven. □

It is worth noting that lowering Lemma 4.1 and lifting Lemma 4.2 consider couples of p-goals in a specialisation relationship, i.e. p-goals of the form G and $(G\gamma\underline{\tau} + X)$. The distinctive point is that a group X of additional atoms may be present in the second p-goal, besides the instantiation of G by γ. The correspondence is obvious with the fact that Definition 4.1 requires that positioning of new atoms is independent of goal specialisation. As it will be clear in the following, this kind of independence is basic in order to assure tolerance to redundancy elimination.

In Gabrielli, Levi and Meo (1996) a class of selection rules is introduced for which independence of atom choices from goal instantiation is assured. These rules are named *skeleton selection rules*. Indeed, they are sensible only to a specific structural extract (the skeleton) of the applied clauses and the initial goal in the story of a derivation. As shown in Gabrielli, Levi and Meo (1996), instantiation independence is sufficient to prove a Strong Lifting Lemma which asserts that, for any skeleton rule S, an SLD derivation of a goal $G\gamma$ via S can be lifted to a derivation of G via the same rule S, relating in a quite strong sense the mgu's and the resolvents. On the other hand, instantiation independence seems not sufficient to assure redundancy elimination tolerance. For example, in agreement to the definition in Gabrielli, Levi and Meo (1996), the selection rule of Example 2.2 is a skeleton rule, because choices only depend upon the length of the initial goal and the ones of applied clauses. Really, choices are performed on the unique basis of the length of the actual resolvent, so that the rule of Example 2.2 can be seen as a case of *state* skeleton selection rule. Anyhow, the rule is not tolerant to redundancy elimination.

To point out the role of the hypothesis of specialisation independence with respect to derivation lifting, let us give the following example where lifting Lemma 4.2 does not hold. Note that the used scheduling rule is instantiation independent, but it is not specialisation independent.

Example 4.2

Let us consider again the scheduling rule of Example 4.1. It is easy to recognise that lifting Lemma 4.2 does not hold for such a rule. Indeed, let P be the following program:

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

$\{p[1.1], r[1.5], r[1.6], s[2], s[2.5]\} \xrightarrow{S,c1}$
$\{r[1.5], r[1.6], p[1.7], r[1.8], r[1.9], s[2], s[2.5]\} \xrightarrow{S,(c2,c2)}$
$\{p[1.7], r[1.8], r[1.9], s[2], s[2.5]\}$
.....

Fig. 2

$c1 = p \longleftarrow p[1], r[2], r[3]$
$c2 = r \longleftarrow .$

Now, in reference to the statement of Lemma 4.2, let:

$G = \{p[1], s[2], s[3]\}$
$G\gamma\underline{\tau} = \{p[1.1], s[2], s[2.5]\}$ and $X = \{r[1.5], r[1.6]\}$.

In Figure 2 an infinite p-SLD derivation of $(G\gamma\underline{\tau} + X)$ in P is shown.
On the contrary, the only p-SLD derivation of G in P is the following one

$\{p[1], s[2], s[3]\} \xrightarrow{S,c1} \{s[2], p[2.5], r[2.6], r[2.7], s[3]\}.$

which fails at the second resolvent. □

From the proofs of Lemmas 4.1 and 4.2, the proof of two corresponding assertions can be easily drawn. They are given in Lemma 4.3 below, and are valid for all scheduling rules in the case of two p-SLD derivations which are lowerings of each other. Part (a) of the lemma may be viewed as a form of variant Lemma.

Lemma 4.3 (*determinism lemma*)
Let S be any scheduling rule and V any arbitrary finite set of variables. Then let G and G' be two p-goals such that G' is a p-variant of G. The following implications hold:

(a) $G \xrightarrow{S,D} Q$ and $G' \xrightarrow{S,D} R$
$\Longrightarrow$ R is a p-variant of Q,

(b) $G' \xrightarrow{S,D} \bullet$
$\Longrightarrow$ $\exists Dr = (G \xrightarrow{S,D} \bullet)$, with $nvar(Dr) \cap V = \varnothing$.

Proof
Let us consider part (a) of the lemma. By definition of p-variant it is $G' = G\gamma\underline{\tau}$, for a renaming γ and a shifting $\underline{\tau}$. By fact (p1) in Lemma 4.1, i.e. 'where σ is a renaming if γ is a renaming and $D/X = \varnothing$', the result appears as an immediate consequence of the proof of Lemma 4.1 itself. It is sufficient to note that, if X is empty and γ is a renaming, the fact that the used scheduling rule is specialisation independent becomes useless. Indeed, in reference to the proofs of Lemma 4.1, though the hypothesis of specialisation independence is dropped, the first steps of (5) and (4) are congruent lowerings of each other, because every scheduling rule is deterministic. Similar considerations are possible for part (b) of the lemma, in reference to the proof of Lemma 4.2. □

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Now, let us give a property that is valid for all scheduling rules and derives easily from Lemma 4.3. It asserts that two p-SLD derivations Dr_1 and Dr_2, via the same scheduling rule S, can be composed giving place to a longer derivation via S, if the last resolvent of Dr_1 coincides with the first of Dr_2.

Property 4.3 (*combination*)
Let S be any (state) scheduling rule. The following implication holds:

$$\exists Dr_1, Dr_2 \quad \text{with} \quad Dr_1 = (G \xrightarrow{S,E} F), \quad Dr_2 = (F \xrightarrow{S,H} Q)$$
$$\Longrightarrow \quad \exists Dr = (G \xrightarrow{E} F \xrightarrow{H} R), \quad \text{with} \quad Dr \in \Delta(S),$$
$$\text{where } R \text{ is a p-variant of } Q.$$

Proof
By Lemma 4.3-b) applied to Dr_2, a p-SLD derivation $Dr' = (F \xrightarrow{S,H} R)$ exists with $nvar(Dr') \cap (nvar(Dr_1) \cup var(G)) = \emptyset$. Thus, Dr' is standardised apart with respect to Dr_1. Since S is a state scheduling rule, Dr is obtained as the composition of Dr_1 and Dr'. The fact that R is a p-variant of Q follows from Lemma 4.3-a), applied to Dr_2 and Dr'. □

5 Stack-queue selection rules

Prolog interpreters adopt a leftmost scheduling policy such that the first atom in the goal is always selected for rewriting and is replaced in the resolvent by the body of the applied clause. In other words, the actual resolvent is maintained as a *stack*, the atom on the top of the stack is always selected for rewriting, while new atoms from the applied clause are pushed on the top of the stack. In analogy, a *queue scheduling policy* may be considered, which corresponds to a very simple case of *fair* selection rule (see Lloyd, 1987). As for the stack scheduling policy the first atom in the resolvent is always selected, but new atoms are positioned at the end of the old resolvent. Thus, the resolvent is treated as a queue of atoms and any queued atom is eventually selected in the case of infinite derivations

In this section, the class of *stack-queue* scheduling rules is defined, which is a generalisation of both stack and queue scheduling policies. According to stack-queue rules, for any clause $c = (ht \longleftarrow B)$, two p-goals M_s and M_q can be identified, with $B = M_s|M_q$, such that the atoms in M_s are always scheduled in *stack mode* while the atoms in M_q are scheduled in *queue mode*. More formally, we have the following definition. As shown in the sequel of this Section 5, the stack-queue class turns out to be an operational characterisation of the class of specialisation independent scheduling rules

Definition 5.1 (*stack-queue derivation steps*)
A set SQ of derivation steps is said to be of *stack-queue* type, if it verifies the following condition. Given any clause $c = (ht \longleftarrow B)$, two p-goals M_s and M_q exist with $M_s|M_q = B$, such that for any p-goal $(a|K)$:

$$a|K \xrightarrow{SQ,c\xi,\mu} R \quad \Longrightarrow \quad R = (M_s\xi\underline{\gamma}|K|M_q\xi\underline{\gamma})\mu.$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

The following property states that any set of stack-queue derivation steps is specialisation independent. Then, as stated in Theorem 5.1, any set of stack-queue derivation steps which satisfies the completeness property is a specialisation independent scheduling rule.

Property 5.1 (*stack-queue implies specialisation independence*)
Let SQ be a stack-queue set of derivation steps. Then SQ is specialisation independent.

Proof
Let us consider two derivation steps in SQ and suppose that derivation step (2) is a lowering of (1) by F. This means that (1) and (2) have the following form, where $c = (ht \longleftarrow M_s|M_q)$:

$$a|K \xrightarrow{SQ,c} (M_s\xi'\underline{\gamma}'|K|M_q\xi'\underline{\gamma}')\alpha' \tag{1}$$

$$a\lambda\underline{\sigma}|(K\lambda\underline{\sigma}+F) \xrightarrow{SQ,c} (M_s\xi''\underline{\gamma}''|(K\lambda\underline{\sigma}+F)|M_q\xi''\underline{\gamma}'')\alpha''. \tag{2}$$

To show that SQ is specialisation independent, we have to verify that derivation step (2) is a congruent lowering of (1) by F, i.e. a shifting $\underline{\rho}$ exists, such that:

$$M_s\underline{\gamma}''|M_q\underline{\gamma}'' = (M_s\underline{\gamma}'|M_q\underline{\gamma}')\underline{\rho}, \quad K\underline{\sigma} = K\underline{\rho}. \tag{3}$$

By Property 3.2, a shifting $\underline{\rho}$ exists such that:

$$M_s\underline{\gamma}''|K\underline{\sigma}|M_q\underline{\gamma}'' =^{(Ax-iii)} (M_s\underline{\gamma}'|K|M_q\underline{\gamma}')\underline{\rho} = M_s\underline{\gamma}'\underline{\rho}|K\underline{\rho}|M_q\underline{\gamma}'\underline{\rho}.$$

Since it is evident that $\#M_s\underline{\gamma}'\underline{\rho} = \#M_s\underline{\gamma}''$ and $\#K\underline{\rho} = \#K\underline{\sigma}$, by Property (3.1-i) we have:

$$M_s\underline{\gamma}'' = M_s\underline{\gamma}'\underline{\rho}, \quad M_q\underline{\gamma}'' = M_q\underline{\gamma}'\underline{\rho}, \quad K\underline{\sigma} = K\underline{\rho},$$

which immediately implies assertion (3). □

Theorem 5.1 (*stack-queue scheduling rules*)
Let SQ be a complete set of stack-queue derivation steps. Then SQ is a specialisation independent scheduling rule.

5.1 Specialisation independence implies stack-queue

Now, we prove (Theorem 5.2) that any specialisation independent scheduling rule is actually a stack-queue rule. Thus, combining this fact with Theorem 5.1, we have that Definition 4.2 and the operational characterisation of Definition 5.1 identify the same family of scheduling rules. To this aim, let us show the following lemma.

Lemma 5.1 (*not internal positioning*)
Let S be a specialisation independent scheduling rule. Given any clause $c = (ht \longleftarrow B)$, for every derivation step of the form:

$$a|K \xrightarrow{S,c\xi,\eta} R, \tag{1}$$

two subgoals M_s and M_q exist, with $B = M_s|M_q$, such that:

$$R = (M_s\xi\underline{\gamma}|K|M_q\xi\underline{\gamma})\eta.$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Proof
Let us consider a p-goal like:

$a|K\underline{\omega}_1|K\underline{\omega}_2|...|K\underline{\omega}_n$, with $n > \#B$.

On the basis of (1), by Property 3.3 a derivation step also exists of the following form:

$$a|K\underline{\omega}_1|...|K\underline{\omega}_n \xrightarrow{S,c\gamma,\mu} (Q = ((K\underline{\omega}_1|...|K\underline{\omega}_n) + B\gamma\underline{\tau})\mu). \tag{2}$$

Since $n > \#B$, an index j must exist such that no atom of B has been positioned inside $K\underline{\omega}_j$. A priori several j's might exist. Without loss of generality, we take any one of them. Thus, two p-goals M_s and M_q must exist, with $M_s|M_q = B$, such that:

$$Q = (M_s\underline{\tau}\gamma + (K\underline{\omega}_1|...|K\underline{\omega}_{j-1}))|K\underline{\omega}_j|(M_q\underline{\tau}\gamma + (K\underline{\omega}_{j+1}|...|K\underline{\omega}_n))\mu. \tag{3}$$

Now, by definition, derivation step (1) has the form:

$$a|K \xrightarrow{S,c} (R = (K + B\xi\underline{\sigma})\eta). \tag{1a}$$

Since S is a specialisation independent rule, step (2) is a congruent lowering of step (1a) by the subgoal $(K\underline{\omega}_1|...|K\underline{\omega}_{j-1}|K\underline{\omega}_{j+1}|...|K\underline{\omega}_n)$, so that a shifting $\underline{\rho}$ exists with $K\underline{\rho} = K\underline{\omega}_j$ and $B\underline{\sigma}\underline{\rho} = B\underline{\tau} = M_s\underline{\tau}|M_q\underline{\tau}$. Then, recalling that (3) implies $M_s\underline{\tau} \dashv K\underline{\omega}_j \dashv M_q\underline{\tau}$, we obtain:

$$(K + B\xi\underline{\sigma})\underline{\rho} = K\underline{\omega}_j + B\xi\underline{\tau} = (M_s\xi\underline{\tau}|K\underline{\omega}_j|M_q\xi\underline{\tau}) = (M_s\xi\underline{\tau}|K\underline{\rho}|M_q\xi\underline{\tau}).$$

Finally:

$$R = (K + B\xi\underline{\sigma})\eta = (K + B\xi\underline{\sigma})\eta\underline{\rho}\underline{\rho}^{-1} = (M_s\xi\underline{\tau}\underline{\rho}^{-1}|K|M_q\xi\underline{\tau}\underline{\rho}^{-1})\eta. \quad \square$$

The following Theorem 5.2 shows that, for any scheduling rule, specialisation independence implies that the rule is stack-queue. Together with Theorem 5.1, this result proofs that stack-queue is an operational characterisation of the set of specialisation independent scheduling rules.

Theorem 5.2 (*specialisation independence implies stack-queue*)
Let S be a specialisation independent scheduling rule. Given any clause $c = (ht \longleftarrow B)$, two p-goals M_s and M_q exist, with $M_s|M_q = B$, such that for every derivation step of the form:

$$a|K \xrightarrow{S,c\xi,\eta} R \tag{1}$$

it is:

$$R = (M_s\xi\underline{\pi}|K|M_q\xi\underline{\pi})\eta.$$

Proof
Let p be the predicate symbol of atom ht. Consider a p-atom b of the form $b = p(x_1, ..., x_k)[s]$, where $x_1, ..., x_k$ are distinct variables. Then, consider a ground p-atom r such that $b \dashv r$. By construction of b and completeness of S, a derivation step of the type $(b|r \xrightarrow{S,c} \bullet)$ exists, which necessarily has the following form because r is a single atom:

$$b|r \xrightarrow{S,c} (M_s\lambda\underline{\varepsilon}|r|M_q\lambda\underline{\varepsilon})\mu, \quad \text{with } B = M_s|M_q. \tag{2}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Now, let us prove that $M_s|M_q$ is the partition of B which is required by the thesis. Consider derivation step (1). Two cases are possible, either $K = \varnothing$ or $K \neq \varnothing$.

Case 1 ($K = \varnothing$).

In this case we have:

$$a \xrightarrow{S,c} (R = B\xi\underline{\pi}\eta = (M_s\xi\underline{\pi}|M_q\xi\underline{\pi})\eta).$$

Case 2 ($K \neq \varnothing$).

On the basis of (1), we have that also p-atom a has p as a predicate symbol, so that a substitution τ and a shifting $\underline{\sigma}$ exist with $a = b\tau\underline{\sigma}$. By (1) and Property 3.3, a derivation step exists like:

$$(b\tau\underline{\sigma}|(r\tau\underline{\sigma} + K) = a|(K + r\underline{\sigma})) \xrightarrow{S,c\xi',\eta'} Q, \tag{4}$$

where by Lemma 5.1 we have that:

$$Q = (N_s\xi'\underline{\gamma}|(r\underline{\sigma} + K)|N_q\xi'\underline{\gamma})\eta', \quad \text{with } B = N_s|N_q. \tag{5}$$

The proof can be now completed by exploiting derivation step (4) as a sort of 'bridge' between (1) and (2). In fact, since S is specialisation independent rule, derivation step (4) is a congruent lowering of step (2) by K, so that a shifting $\underline{\rho}'$ exists with $r\underline{\rho}' = r\underline{\sigma}$ and $(M_s\underline{\varepsilon}|M_q\underline{\varepsilon})\underline{\rho}' = N_s\underline{\gamma}|N_q\underline{\gamma}$. As a consequence (see (5) and (2)), we can write:

$$N_s\underline{\gamma}|r\underline{\sigma}|N_q\underline{\gamma} = N_s\underline{\gamma}|N_q\underline{\gamma} + r\underline{\sigma} = (M_s\underline{\varepsilon}|M_q\underline{\varepsilon})\underline{\rho}' + r\underline{\rho}' = (M_s\underline{\varepsilon}|r|M_q\underline{\varepsilon})\underline{\rho}',$$

with $r\underline{\sigma} = r\underline{\rho}'$.

Then, by Property (3.1-ii) we have that $N_s\underline{\gamma} = M_s\underline{\varepsilon}\underline{\rho}'$, which obviously implies:

$$\#N_s = \#M_s. \tag{6}$$

Now, let us note that by Lemma 5.1 it must be:

$$R = (A_s\xi\underline{\pi}|K|A_q\xi\underline{\pi})\eta, \quad \text{with } B = A_s|A_q. \tag{7}$$

Since S is a specialisation independent rule, derivation step (4) is a congruent lowering of step (1) by $r\underline{\sigma}$, so that a shifting $\underline{\rho}''$ exists with $K\underline{\rho}'' = K$ and $(A_s\underline{\pi}|A_q\underline{\pi})\underline{\rho}'' = N_s\underline{\gamma}|N_q\underline{\gamma}$. As a consequence (see (5) and (7)) we can write:

$$N_s\underline{\gamma}|K|N_q\underline{\gamma} = N_s\underline{\gamma}|N_q\underline{\gamma} + K = (A_s\underline{\pi}|A_q\underline{\pi})\underline{\rho}'' + K\underline{\rho}'' = (A_s\underline{\pi}|K|A_q\underline{\pi})\underline{\rho}'',$$

with $K = K\underline{\rho}''$.

Then, by Property (3.1-ii), we have that $N_s\underline{\gamma} = A_s\underline{\pi}\underline{\rho}''$, which obviously implies:

$$\#N_s = \#A_s. \tag{8}$$

By (2) and (7), it is $M_s|M_q = A_s|A_q = B$. By (6), (8) and Property (3.1-i), we have that:

$$A_s = M_s \text{ and } A_q = M_q.$$

Substituting in (7), the thesis is obtained. □

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

5.2 Notes on the structure of stack-queue derivations

Let us consider a stack-queue derivation like:

$$A|B \overset{SQ,M,\sigma}{\longrightarrow} \bullet, \quad \text{where } M = c_1, c_2, ...c_h \text{ and } M/B = \emptyset. \tag{1}$$

By definition of stack-queue scheduling rules, only atoms in A together with atoms deriving from A and allocated in stack mode can be rewritten in derivation (1). Thus, derivation (1) has the form:

$$A|B \overset{SQ,c_1\xi_1,\sigma_1}{\longrightarrow} X_1|A_1|B\sigma_1|Y_1 \overset{SQ,c_2\xi_2,\sigma_2}{\longrightarrow} ...$$
$$X_i|A_i|B\sigma_1...\sigma_i|Y_i \overset{SQ,c_{i+1}\xi_{i+1},\sigma_{i+1}}{\longrightarrow} ... \overset{SQ,c_h\xi_h,\sigma_h}{\longrightarrow} X_h|A_h|B\sigma_1...\sigma_h|Y_h, \tag{1a}$$

where:

- each X_i is formed by new atoms deriving from A which are allocated in stack mode,
- each A_i is formed by atoms of A which are not yet rewritten,
- each Y_i is formed by new atoms deriving from A which are allocated in queue mode.

The above structural considerations suggest the following formal definition.

Definition 5.2 (*A-preq type derivations*)
A p-SLD derivation, of the form $A|B \overset{SQ,M}{\longrightarrow} \bullet$, is of *pre-queued type w.r.t.* the subgoal A (simply written *A-preq type* in the following) if the only rewritten atoms are:

– atoms from the subgoal A,
– atoms deriving from A and allocated in stack mode.

Note that Definition 5.2 is significant even if $B = \emptyset$. It is evident that any A-preq derivation has the form (1a). In the sequel we use the following shortened notation to represent A-preq type derivations:

$$A|B \overset{SQ,M,\sigma}{\longrightarrow} A^s|B\sigma|A^q, \tag{Ap}$$

where, with reference to (1a), $A^s = X_h|A_h$ stands for 'stacked subgoal derived from A', and $A^q = Y_h$ means 'queued subgoal derived from A'. It is evident that in any preq type derivation we have $M/B = \emptyset$.

The following definition characterises an *A-queued* derivation as an A-preq derivation where all atoms of A are rewritten together with all atoms deriving from A and allocated in stack mode, i.e. $A^s = \emptyset$. Intuitively, an A-queued derivation is an A-preq derivation which cannot be extended without loosing its A-preq nature. Indeed, the acronym 'A-preq' stands for 'A-pre-queued' derivation.

Definition 5.3 (*A-queued derivations*)
Let SQ be a stack-queue scheduling rule. A derivation which is of A-preq type and has the form:

$$A|B \overset{SQ,K,\sigma}{\longrightarrow} B\sigma|A^q \tag{Aq}$$

is said to be *queued w.r.t. A* (simply written *A-queued* in the following).

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

In the following Section 5.3, we will exploit the notations introduced in (Ap) and (Aq) to represent A-preq type and A-queued derivations, respectively. It is worth noting that starting from a p-goal of the form $A|B$, when the A-queued derivation is reached, the last resolvent presents a situation where the roles of A and B are exchanged. In practice, restarting from $B\sigma|A^q$, the derivation can attempt to proceed towards a $(B\sigma)$-queued derivation. The proof of an important result in Section 5.3 (Duplication Theorem 5.3) is based on this cyclic behaviour of stack-queue derivations.

5.3 Duplication tolerance

In this section an important property is shown for stack-queue scheduling rules. Let us give an intuitive presentation of this result, which is stated in the *full duplication theorem* (Theorem 5.4). Suppose that a p-SLD derivation Dr of G in P can be developed via a stack-queue scheduling rule SQ. Then consider a p-goal G' which is equal to G apart from the duplication of some atoms. Furthermore, suppose that each copy is scheduled after the corresponding original atom. In this hypothesis, the full duplication theorem asserts that a p-SLD derivation of G' in P exists via the same scheduling rule SQ, where all derivation steps of Dr are redone in the order.

The full duplication theorem is basic for the proof of the final results of the paper, i.e. results about redundancy elimination tolerance which are given in Section 6. Indeed, let us consider the problem of preserving program termination. Intuitively, program termination is preserved if the introduction of redundancy elimination does not provoke any really different new derivations. Reversing the viewpoint, termination is retained if any derivation, developed in presence of redundancy elimination, can be traced again when redundancy is left in place. The full duplication theorem asserts this kind of fact in the simplest case, i.e. when redundancy has the form of a replica of atoms already present in the initial p-goal, provided that the scheduling rule is of stack-queue type.

First we show a duplication theorem (Theorem 5.3) which is valid when only one atom or group of adjacent atoms is duplicated. Then the result is easily extended to obtain the full theorem. Though intuitive in appearance, Theorem 5.3 has a relatively complex proof. In this section we give only a sketch of the argument. In the sketch, we will make reference to the particular case of completely ground derivations, i.e. derivations such that all resolvents are ground. This simplification will allow us to highlight the essence of the argument, without having to do with technical problems deriving from variable instantiations. Formal presentation of the proof of Theorem 5.3 is given in Appendix B. Note that the hypothesis of ground resolvents is verified in the case that no new variable is present in clause bodies and initial goals are ground.

Theorem 5.3 (*duplication theorem*)
Let P be a logic program and SQ a stack-queue scheduling rule. Given two p-goals of the form $A|B|C|D$ and $A|B|C|B\underline{\pi}|D$, the following implication holds:

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

$$A|B|C|D \overset{SQ,X.P}{\longrightarrow} Q \tag{1}$$
$$\Longrightarrow \exists Y \text{ such that } A|B|C|B\underline{\pi}|D \overset{SQ,Y.P}{\longrightarrow} R$$
$$\text{with } X \subseteq_L Y \quad \text{and} \quad \#Q \preceq \#R.$$

Proof (sketch)
Let $\Delta(SQ, n)$ denote the subset of $\Delta(SQ)$ such that, for any derivation Dr in $\Delta(SQ, n)$, it is $\#Dr \preceq n$, where $\#Dr$ denotes the length of Dr. We show the thesis by induction on n. In other words, we show that the thesis holds when derivation (1) belongs to $\Delta(SQ, n)$, for any $n \geq 0$. The fact is obvious for $\Delta(SQ, 0)$. In order to justify the inductive step from $\Delta(SQ, n-1)$ to $\Delta(SQ, n)$, for $n > 0$, let us consider a derivation like:

$$(A|B|C|D \overset{X.P}{\longrightarrow} Q) \in \Delta(SQ, n) \tag{1a}$$

and show that $(A|B|C|B\underline{\pi}|D \overset{SQ,Y.P}{\longrightarrow} R)$ exists with $X \subseteq_L Y$ and $\#Q \preceq \#R$. The following three possible situations must be taken into account. Then, we start with case 3, which is the most significant one.

1. derivation (1a) is of $(A|B|C)$-preq type,
2. derivation (1a) is of $(A|B|C|D)$-preq type, and not of $(A|B|C)$-preq type,
3. derivation (1a) is not of $(A|B|C|D)$-preq type.

Case 3.
As already said, the simplified argument, which we use in this sketch, works in the hypothesis that all resolvents are ground, so that derivation (1a) has the following form:

$$A|B|C|D \overset{H}{\longrightarrow} B|C|D|A^q \overset{K}{\longrightarrow} C|D|A^q|B^q \overset{M}{\longrightarrow}$$
$$D|A^q|B^q|C^q \overset{N}{\longrightarrow} A^q|B^q|C^q|D^q \overset{T}{\longrightarrow} Q, \quad \text{where } H|K|M|N|T = X. \tag{2}$$

Then, it is intuitive that a derivation can be constructed like the following, where $\underline{\phi}$ is a suitable shifting:

$$A|B|C|B\underline{\pi}|D \overset{SQ,H}{\longrightarrow} B|C|B\underline{\pi}|D|A^q \overset{SQ,K}{\longrightarrow}$$
$$C|B\underline{\pi}|D|A^q|B^q \overset{SQ,M}{\longrightarrow} B\underline{\pi}|D|A^q|B^q|C^q \overset{SQ,K}{\longrightarrow} \tag{3}$$
$$D|A^q|B^q|C^q|B^q\underline{\phi} \overset{SQ,N}{\longrightarrow} A^q|B^q|C^q|B^q\underline{\phi}|D^q.$$

By construction of (2), $A^q|B^q|C^q|D^q \overset{T}{\longrightarrow} Q$ is a derivation belonging to $\Delta(SQ, m)$, with $m < n$. By inductive hypothesis, a derivation exists such that:

$$A^q|B^q|C^q|B^q\underline{\phi}|D^q \overset{SQ,Y'.P}{\longrightarrow} R', \tag{5}$$
$$\text{with } T \subseteq_L Y' \text{ and } \#Q \preceq \#R'. \tag{5a}$$

By Property 4.3, derivations (3) and (5) can be combined to yield a derivation of the form:

$$A|B|C|B\underline{\pi}|D \overset{SQ,(H|K|M|K|N),P}{\longrightarrow} A^q|B^q|C^q|B^q\underline{\phi}|D^q \overset{SQ,Y',P}{\longrightarrow} R, \tag{6}$$

where R is a p-variant of R', which implies $\#R = \#R'$. Finally:

$$X = H|K|M|N|T \subseteq_L^{(5a)} (H|K|M|K|N|Y'), \quad \#Q \preceq^{(5a)} \#R' = \#R.$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Case 2.

Derivation (1a) has the form $A|B|C|D \overset{H|K|M|N}{\longrightarrow} D^s|A^q|B^q|C^q|D^q$, where $H|K|M|N = X$. Analogously to case 3), a derivation can be constructed like:

$$A|B|C|B\underline{\pi}|D \overset{SQ,(H|K|M|K|N),P}{\longrightarrow} D^s|A^q|B^q|C^q|B^q\underline{\pi}|D^q.$$

Case 1.

Derivation (1a) has the form $A|B|C|D \overset{X}{\longrightarrow} (A|B|C)^s|D|(A|B|C)^q$. A derivation exists like:

$$(A|B|C)|B\underline{\pi}|D \overset{SQ,X}{\longrightarrow} (A|B|C)^s|B\underline{\pi}|D|(A|B|C)^q. \quad \square$$

Now we can state and prove the full duplication theorem, which extends the previous Theorem 5.3 to the duplication of two or more not adjacent atoms in the initial goal of a p-SLD derivation.

Theorem 5.4 (*full duplication theorem*)
Let P be a logic program and SQ a stack-queue scheduling rule. Given a p-goal $N + F$ such that:

$$\forall b[s] \in F, \quad \exists b[s'] \in N \quad \text{with } s' < s,$$

the following implication holds:

$$N \overset{SQ,M,P}{\longrightarrow} Q$$
$$\Longrightarrow \exists Y \text{ such that } N + F \overset{SQ,Y,P}{\longrightarrow} R$$
$$\text{with } M \subseteq_L Y \text{ and } \#Q \preceq \#R.$$

Proof
By hypothesis, the subgoal F is made of duplicated atoms. Then, the proof is by induction on the length of F. Indeed, if F is empty the thesis is true. Now, suppose that the thesis is already proven for any F with $\#F = n \geq 0$. Then let us consider any p-goal $G = F|b[s]$ with $\#F = n$. By inductive hypothesis a derivation exists such that:

$$N + F \overset{SQ,Z,P}{\longrightarrow} S, \quad \text{with} \quad M \subseteq_L Z \quad \text{and} \quad \#Q \preceq \#S.$$

By hypothesis, three p-goals A, C and D exist together with a p-atom $b[s']$, such that:

$$N + (F|b[s]) = A|b[s']|C|b[s]|D \quad \text{and} \quad N + F = A|b[s']|C|D.$$

As a consequence, Theorem 5.3 can be applied to $N + F$ and $N + (F|b[s])$ yielding:

$$N + (F|b[s]) \overset{SQ,Y,P}{\longrightarrow} R, \quad \text{with } Z \subseteq_L Y \text{ and } \#S \preceq \#R.$$

Now the induction step is completed, because:

$$M \subseteq_L Z \subseteq_L Y \quad \text{and} \quad \#Q \preceq \#S \preceq \#R. \quad \square$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

6 Redundancy elimination tolerance

In this section, the tolerance of stack-queue scheduling rules to redundancy elimination is considered. The preservation of program termination in shown in Section 6.1. The preservation of the completeness of EVR_L loop check is shown in Section 6.2 for function free programs. First, the idea of goal reduction, which is originally given in Ferrucci, Pacini and Sessa (1995), and is recalled in Definition 2.1 of this paper, is restated. Indeed, in Section 2, little attention is paid to the positions of atoms which are removed from a resolvent. However, if the execution is based on atom priority values, it is intuitive that removing an atom without any convenient expedient may overthrow the essence of previous atom scheduling. Thus, a refined definition of goal reduction is given below (Definition 6.1) which fits the frame of priority SLD derivation mechanisms.

The inspiring idea of *priority reduction* is quite simple. According to Definition 2.1, for any removed atom b, an *eliminating* atom $a = b\tau$ exists which remains in the reduced resolvent. Several removed atoms may share the same eliminating one. In reference to Definition 6.1 below, for any eliminating atom $a_j[p_j]$, the corresponding subset A_j of eliminated atoms is pointed out. Then, except for the case $a_j[p_j] \dashv A_j$, any $a_j[p_j]$ is advanced to the least priority value in A_j. In other words, *each eliminating atom is advanced to replace the first scheduled atom among its eliminated ones*. Intuitively, the aim is to restore the essence of the previous atom priorities. The notation $\{+A_j, 1 \preceq j \preceq h\}$ will represent the merging $A_1 + A_2 + ... + A_h$, and the notation $prs(A_j)$ the set of priority values in A_j.

Definition 6.1 (*priority reduced goals*)
Let X be a set of variables, τ a substitution and G a p-goal. A p-goal N is a *reduced p-goal* of G by τ up to X, denoted by $G >>^{\tau} N$, if the following conditions hold:

(i) $G = F + \{+a_j[p_j],\ 1 \preceq j \preceq h\} + \{+A_j,\ 1 \preceq j \preceq h\}$,
where $\forall b[s] \in A_j, \quad b\tau = a_j, \quad 1 \preceq j \preceq h$,

(ii) $N = F + \{+a_j[r_j],\ 1 \preceq j \preceq h\}$,
where $r_j = min(\{p_j\} \cup prs(A_j)), \quad 1 \preceq j \preceq h$,

(iii) $\forall x \in (X \cup var(N)) \quad$ it is $\quad x\tau = x$.

Example 6.1
Given the p-goal

$G = p(z)[1], q(w)[2], p(a)[3], p(y)[4], q(v)[5]$,

the following N is a reduced p-goal of G by the substitution $\tau = \{z/a, y/a, v/w\}$:

$N = p(a)[1], q(w)[2]$.

Note that $p(a)[3]$ has been advanced to replace the first of the atoms it eliminates, that is $p(z)[1]$. □

Now, the idea of *priority reduced SLD derivation* can be defined as a generalisation of Definition 3.5. In essence a priority reduced SLD derivation is a p-SLD derivation

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

where, at any step, a priority reduction of the resolvent according to Definition 6.1 is allowed.

Definition 6.2 (*priority Reduced SLD derivation*)
Let P be a program and G_o a p-goal. A *priority reduced SLD derivation* of G_o in P (*p-RSLD derivation* for short) is a possibly infinite sequence of priority reductions and derivation steps

$$G_o >>^{\alpha_o} N_o \overset{c_o\xi_o,\theta_o}{\longrightarrow} G_1 \ldots G_k >>^{\alpha_k} N_k \overset{c_k\xi_k,\theta_k}{\longrightarrow} G_{k+1} >>^{\alpha_{k+1}} N_{k+1} \ldots$$

where, for any $j \geq 0$,

(i) c_j is a clause in P,

(ii) $var(c_j\xi_j) \cap (var(G_o) \cup var(c_o\xi_o) \cup \ldots \cup var(c_{j-1}\xi_{j-1})) = \emptyset$,

(iii) $G_j >>^{\alpha_j} N_j$ up to $var(G_o\theta_o \ldots \theta_{j-1})$.

The notation

$$G \overset{S,D}{\longrightarrow}>> N$$

will be used to represent a p-RSLD derivation which is developed in agreement with the scheduling rule S using the template D. The last resolvent N is intended to be a reduced resolvent.

6.1 Termination preserving

In this section, the redundancy elimination tolerance of stack-queue scheduling rules is shown, with reference to program termination (Theorem 6.1). The following lemma is fundamental for proving the preservation of termination, as well as the preservation of EVR_L loop check completeness.

Lemma 6.1
Let P be a program and SQ a stack-queue scheduling rule. The following implication holds:

$$G \overset{SQ,X,P}{\longrightarrow}>> Q \tag{1}$$

$$\Longrightarrow \quad \exists Z \text{ such that } G \overset{SQ,Z,P}{\longrightarrow} R, \quad \text{with } X \subseteq_L Z,\ \#Q \leq \#R.$$

Proof
The proof is by induction on the length of X. If $\#X = 0$, the thesis is trivially verified with $Z = \emptyset$. Then let us consider $X = c|H$. Derivation (1) may be rewritten as:

$$(G >>^{\tau} N) \overset{SQ,c}{\longrightarrow} F \overset{SQ,H}{\longrightarrow}>> Q. \tag{2}$$

Since $\#H < \#X$, by inductive hypothesis, a p-SLD derivation exists of the form:

$$F \overset{SQ,K,P}{\longrightarrow} T, \quad \text{with } H \subseteq_L K,\ \#Q \leq \#T. \tag{3}$$

By Property 4.3, the first derivation step of (2) and derivation (3) can be combined to yield a derivation of the following form:

$$N \overset{SQ,c}{\longrightarrow} F \overset{SQ,K}{\longrightarrow} S, \quad \text{where } S \text{ is a p-variant of } T. \tag{4}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Now, let us consider the p-goal $G\tau$. With reference to Definition 6.1, we have that:

$$G\tau = (F + \{+a_j[p_j],\ 1 \preceq j \preceq h\})\tau + \{+A_j\tau,\ 1 \preceq j \preceq h\} =^{(Def.\ 6.1-iii)}$$
$$F + \{+a_j[p_j],\ 1 \preceq j \preceq h\} + \{+A_j\tau,\ 1 \preceq j \preceq h\} =^{(Def.\ 6.1-i-ii)}$$
$$F + \{+a_j[r_j],\ 1 \preceq j \preceq h\} + \{+A_j\tau\{r_j/p_j\},\ 1 \preceq j \preceq h\}$$
$$= N + \{+A_j\tau\{r_j/p_j\},\ 1 \preceq j \preceq h\}^{[2]},$$

where $a_j[r_j] \dashv A_j\tau\{r_j/p_j\}$ and any atom in $A_j\tau\{r_j/p_j\}$ is a duplicate of a_j, $1 \preceq j \preceq h$. Then, N and $G\tau$ verify the hypothesis of Theorem 5.4. As a consequence, by (4) a derivation also exists such that:

$$G\tau \overset{SQ,Z,P}{\longrightarrow} V, \quad \text{with } c|K \subseteq_L Z \text{ and } \#S \preceq \#V \tag{5}$$

Now, let us apply lifting Lemma 4.2 to (5). We obtain that a p-SLD derivation exists like:

$$G \overset{SQ,Z,P}{\longrightarrow} R. \tag{6}$$

where, applying lowering Lemma 4.1 to (5) and (6), we have that $\#V = \#R$. Finally, we conclude:

$$X = c|H \subseteq_L^{(3)} c|K \subseteq_L^{(5)} Z,$$
$$\#Q \preceq^{(3)} \#T =^{(4)} \#S \preceq^{(5)} \#V = \#R. \quad \square$$

Theorem 6.1 (*Termination preserving*)
Let P be a program, G a p-goal and SQ a stack-queue scheduling rule. If every p-SLD derivation of G in P via SQ is finite, then any p-RSLD derivation via SQ is finite too.

Proof
Let T be the p-SLD tree of G in P via SQ. By hypothesis, every p-SLD derivation of G in P via SQ is finite. As a consequence, since T is a finitely branching tree, by Konig's lemma (see Theorem K, in Knuth, 1997) T is a finite tree. Let f be the depth of T. Given any p-RSLD derivation of the form $G \overset{SQ,X,P}{\longrightarrow} >> \bullet$, by Lemma 6.1 a p-SLD derivation of the form $G \overset{SQ,Z,P}{\longrightarrow} \bullet$ exists in T, with $X \subseteq_L Z$. However, $\#Z \preceq f$, so that we obtain $\#X \preceq \#Z \preceq f$. In conclusion, the length of all p-RSLD derivations of G in P via SQ is limited by f. $\square$

Let us close this section with two examples which show that both stack-queue scheduling and eliminating atom advancement are essential for redundancy elimination tolerance. The first example shows the necessity of advancement of eliminating atoms. The second one is an example of state scheduling rule which is not tolerant to redundancy elimination, though goal reduction is performed in agreement with Definition 6.1. Of course, the scheduling rule is not of stack-queue type. In the following sketches of p-SLD and p-RSLD derivations, explicit indication of priority values is omitted, for the sake of brevity.

[2] The notation $A_j\tau\{r_j/p_j\}$ means that the priority value r_j is replaced by p_j in the p-goal $A_j\tau$.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Resolvent	Reduced Resolvents
$p\|q(a) >>^{\varepsilon}$	$p\|q(a) \xrightarrow{S,c}$
$q(x_1)\|p\|q(a) >>^{\{x_1/a\}}$	$p\|q(a) \xrightarrow{S,c}$
$q(x_2)\|p\|q(a) >>^{\{x_2/a\}}$	$p\|q(a) \xrightarrow{S,c}$
.....	

Fig. 3

Resolvents	Reduced Resolvents
$q(x, x_1)\|t(x_1, x) >>^{\varepsilon}$	$q(x, x_1)\|t(x_1, x) \xrightarrow{S,c3}$
$r\|s(x_2, x_1)\|r\|q(x, x_2)\|t(x_1, x) >>^{\varepsilon}$	$r\|s(x_2, x_1)\|q(x, x_2)\|t(x_1, x) \xrightarrow{S,c1}$
$s(x_2, x_1)\|q(x, x_2)\|t(x_1, x) >>^{\varepsilon}$	$s(x_2, x_1)\|q(x, x_2)\|t(x_1, x) \xrightarrow{S,c2}$
$q(x, x_2)\|t(x_2, x_1)\|t(x_1, x) >>^{\varepsilon}$	$q(x, x_2)\|t(x_2, x_1)\|t(x_1, x) \xrightarrow{S,c3}$
.....	

Fig. 4

Example 6.2
Let us consider the stack scheduling rule (i.e. the usual leftmost rule) and the following single clause program P:

$$c = p \longleftarrow q(x)|p.$$

It is evident that all p-SLD derivations fail. However, if advancement of eliminating atoms is not performed, an infinite p-RSLD derivation of P exists, as shown in Figure 3. □

Example 6.3
Let S be a scheduling rule which behaves as a stack rule, with an exception when atoms having s as a predicate symbol are rewritten. In this case new atoms are positioned immediately after the first old atom, if one exists. Then, let us consider the logic program P consisting of the following clauses:

$$c1 = r \longleftarrow$$

$$c2 = s(x, y) \longleftarrow t(x, y)$$

$$c3 = q(x, y) \longleftarrow r|s(z, y)|r|q(x, z).$$

It is easy to verify that all p-SLD derivations of P terminate independently of the initial p-goal. In fact, given a p-SLD derivation of G in P, where G is any p-goal, two cases are possible: either an atom with predicate symbol q is rewritten or not. If no atom with predicate symbol q is rewritten, the derivation terminates evidently. Otherwise the derivation fails, as described below:

$$G \xrightarrow{S} q(..)|K \xrightarrow{S,c3} r|s(..)|r|q(..)|K \xrightarrow{S,c1}$$
$$s(..)|r|q(..)|K \xrightarrow{S,c2} r|t(..)|q(..)|K \xrightarrow{S,c1} t(..)|q(..)|K.$$

Now let us show that, if reduction of resolvents is allowed, an infinite p-RSLD derivation of P exists.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

It is easy to verify that the infinite RSLD derivation in Figure 4 cannot be pruned neither by EVR_L loop check nor by more powerful checks (like SIR_M) which are based on *subsumption* relationships between resultants (Bol, Apt and Klop, 1991). □

6.2 *Preserving the completeness of* EVR_L *loop check*

In this section we prove the preservation of EVR_L loop check completeness, passing from p-SLD to p-RSLD. The result holds for function free programs, provided that stack-queue scheduling rules are used in combination with priority reduction of resolvents, as introduced in Definition 6.1. The section starts with a characterisation of EVR_L loop check which exploits the concept of priority shifting and is equivalent to the one stated in Definition 2.3. In essence, passing from Definition 2.3 to Definition 6.3 below, only assertion (ii) is modified. On the other hand, the requirement $N_j = N_i\tau\underline{\tau}$ is plainly equivalent to $N_i\tau =_L N_j$, since any shifting $\underline{\tau}$ implies that the order of atoms is preserved.

Definition 6.3 (*Priority Equality Variant Check for Resultants*)
A p-RSLD derivation

$$G_o >>^{\alpha_o} N_o \overset{c_o\xi_o,\theta_o}{\longrightarrow} G_1 \ldots G_{h-1} >>^{\alpha_{h-1}} N_{h-1} \overset{c_{h-1}\xi_{h-1},\theta_{h-1}}{\longrightarrow} G_h >>^{\alpha_h} N_h \ldots$$

is *pruned* by *priority Equality Variant of Resultant* check (called *p-*EVR_L *check*, in the following), if for some i and j, with $0 \leq i < j$, a renaming τ and a shifting $\underline{\tau}$ exist such that:

(i) $G_o\theta_o \ldots \theta_{j-1} = G_o\theta_o \ldots \theta_{i-1}\tau$,

(ii) $N_j = N_i\tau\underline{\tau}$.

With reference to the above definition, any couple $Rs_h = [N_h, G_o\theta_o \ldots \theta_{h-1}]$ is a *reduced resultant*. Given two reduced resultants $Rs_j = [N_j, G_o\theta_o \ldots \theta_{j-1}]$ and $Rs_i = [N_i, G_o\theta_o \ldots \theta_{i-1}]$, for which requirements (i) and (ii) of Definition 6.3 hold, we will write $Rs_i \cong Rs_j$. In other words, Definition 6.3 expresses that p-EVR_L loop check is based on detecting that a reduced resultant is obtained which is connected by the relationship $\cong$ to a preceding one in the same derivation. It is worth noting that $\cong$ is an equivalence relationship.

Now let us prove Theorem 6.2, which states that the completeness of p-EVR_L loop check is preserved passing from p-SLD to p-RSLD, if stack-queue scheduling rules are used. To this aim we provide a necessary condition which holds whenever p-EVR_L prunes every infinite p-SLD derivation of a goal G in a program P via a scheduling rule S. Indeed, as shown in Lemma 6.2, in this hypothesis the length of resolvents of all possible derivations of G in P via S is limited. The structure and the proof of Lemma 6.2 are strictly analogous to the ones of Lemma 2.2. Note also that Lemma 6.2 holds for any scheduling rule. On the contrary, the stack-queue hypothesis is necessary in Theorem 6.2, which concludes the section.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Lemma 6.2
Let P be a program and G a p-goal. Suppose that all infinite p-SLD derivations of G in P via a scheduling rule S are pruned by p-EVR_L. Then, a finite bound l exists such that, for each resolvent R in any p-SLD derivation of G in P via S, it is $\#R \preceq l$.

Proof
The proof of this lemma can be obtained from the one of Lemma 2.2, by means of the following replacements:

'Let T be the p-SLD tree of G in P via S' for 'Let T be an S-tree of G in P',
'By Determinism Lemma 4.3' for 'Since T contains all SLD derivations of G in P',
'p-EVR_L' and 'p-variant' for 'EVR_L' and 'variant', respectively. □

Theorem 6.2 (*p-EVR_L loop check completeness preservation*)
Let P be a function free program, G_o a p-goal and SQ a stack-queue scheduling rule. Suppose that all infinite p-SLD derivations of G_o in P via SQ are pruned by p-EVR_L, then all infinite p-RSLD derivations of G_o in P via SQ are pruned by p-EVR_L.

Proof
Let D be an infinite p-RSLD derivation of G_o in P via SQ. Let $(G_o \overset{SQ,X}{\longrightarrow}>> Q)$ be any finite prefix of D. By Lemma 6.1, a p-SLD derivation $D' = (G_o \overset{SQ,Z}{\longrightarrow} R)$ exists with $\#Q \preceq \#R$. On the other hand, by Lemma 6.2 a bound l exists such that $\#Q \preceq \#R \preceq l$. However, Q is the generic reduced resolvent in D, so that the number of atoms in all reduced resolvents of D is bounded by l. As a consequence, the number of atoms in all reduced resultants of D is also limited. Since the program P has finite many predicate symbols and constants and no function symbol is allowed, the relationship $\cong$ between reduced resultants of D has only finitely many equivalence classes. Then, for some $0 \preceq i < k$ in D, we have that the k^{th} reduced resultant is related by $\cong$ to the ith one. This implies that D is pruned by p-EVR_L. □

7 Conclusions

In the paper, the problem of possible undesirable effects of redundancy elimination from resolvents is addressed. In particular we have shown that program termination and loop check completeness can be lost. Conditions are characterised which ensure the redundancy elimination tolerance, in the sense that program termination and completeness of equality loop check are preserved when redundancy is eliminated. However, difficulties in analysing interdependence of redundancy elimination effects from the used selection rule have arisen, and the necessity of a framework to formalise suitable features of selection rules has been highlighted. To this aim, a highly expressive execution model based on priority mechanism for atom selection is developed in the paper. The distinctive aspect is that primary importance is given to the event of arrival of new atoms from the body of the applied clause at rewriting time, when new atoms can be freely positioned with respect to old ones in the

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

resolvent. Then, at any derivation step, the atom with optimum priority is simply selected.

The results presented in the paper show that the new computational model is able to give remarkable insights into general properties of selection rules. As a matter of fact, the priority model allows us to formalise the delicate concepts on which the axiomatic definition of specialisation independent scheduling rules is based. As a quite unexpected result, the specialisation independence turns out to be equivalent to stack-queue scheduling technique, which has a very simple operational characterisation. In other words, the priority mechanism is necessary to formalise the real semantic features of specialisation independent scheduling rules. On the contrary, the full generality of the same mechanism can be abandoned if only operational aspects of specialisation independent rules are of interest, in the sense that all we need is a 'watershed' between the stacked and the queued atoms.

It is widely acknowledged that the study of selection rules is a difficult subject which deserves attention. We are confident that the computational model proposed in the paper can be usefully exploited in future work to get further insights into topics which are related to selection rule theory and application, such as loop check, termination and optimisation of derivation processes.

A Appendix

This Appendix contains the formal proofs of Properties 3.3 and 4.1. The very simple Property A1 is considered before proving Property 3.3.

Property A.1

Let S be a complete set of derivation steps. Given a p-goal G and a clause c, the following implication holds:

$$\exists Ds \text{ derivation step of the type } (G \xrightarrow{c\xi} \bullet), \tag{1}$$
$$\Longrightarrow \exists Ds' \text{ of the type } (G \xrightarrow{c\xi} \bullet), \quad \text{with } Ds' \in S.$$

Proof

Let $G = a|K$ and $c = (ht \longleftarrow B)$. By (1) and the completeness of S (part i), a derivation step exists of the form:

$$(a|K \xrightarrow{c} (K + B\xi'\underline{\theta})\mu') \in S \tag{2}$$

By definition, the derivation step in (1) has the form:

$$a|K \xrightarrow{c} (K + B\xi\underline{\tau})\mu.$$

Then, it is evident that a derivation also exists like:

$$Ds' = (a|K \xrightarrow{c} (K + B\xi\underline{\theta})\mu).$$

By construction, derivation steps (2) and Ds' are congruent lowerings of each other. Then, by completeness of S (part ii), derivation step Ds' belongs to S. □

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Property A.2 (*Property 3.3*)
Let S be a complete set of derivation steps. Given two p-goals $a\gamma\underline{\tau}|G$ and $a|F$, let us fix arbitrarily a finite set V of variables. The following implication holds:

$$\exists Ds \text{ derivation step of the form } a\gamma\underline{\tau}|G \xrightarrow{c} \bullet \tag{1}$$
$$\Longrightarrow \exists Ds' \text{ of the form } a|F \xrightarrow{c} \bullet, \text{ with } Ds' \in S \text{ and } nvar(Ds') \cap V = \varnothing.$$

Proof
Let $c = (ht \longleftarrow B)$. On the basis of (1), by definition of derivation step, a standardisation apart renaming ξ' for c and an mgu β exist, with $a\gamma\beta = (ht)\xi'\beta$. Then, let us consider a renaming ξ of $c\xi'$, such that the following assertions hold for the range of ξ:

$$var(a|F) \cap var(c\xi'\xi) = \varnothing, \tag{2a}$$
$$domain(\gamma) \cap var((ht)\xi'\xi) = \varnothing, \tag{2b}$$
$$domain(\xi^{-1}) \cap var(a\gamma) = \varnothing \tag{2c}$$
$$var(c\xi'\xi) \cap V = \varnothing. \tag{2d}$$

By facts (2b) and (2c), we have that:

$$a\gamma\xi^{-1}\beta =^{(2c)} a\gamma\beta = (ht)\xi'\beta = (ht)\xi'\xi\xi^{-1}\beta =^{(2b)} (ht)\xi'\xi\gamma\xi^{-1}\beta.$$

In other words, a and $(ht)\xi'\xi$ unify through the unifier $\gamma\xi^{-1}\beta$. On the other hand, the fact (2a) says that $\xi'\xi$ is a standardisation apart renaming for c with respect to $a|F$. Then, a derivation step exists of the form $a|F \xrightarrow{c\xi'\xi} \bullet$. By hypothesis the set S is complete, so that by Property A1 we have also a derivation step such that:

$$Ds' = (a|F \xrightarrow{c\xi'\xi} \bullet) \in S.$$

Since it is $nvar(Ds') = var(c\xi'\xi)$, by (2d) we have that $nvar(Ds') \cap V = \varnothing$. □

Property A.3 (*Property 4.1*)
Let $c = (ht \longleftarrow B)$ be a clause. Let us consider two derivation steps Ds_1 and Ds_2 such that the Ds_2 is a lowering of Ds_1 by X. The following implication holds:

$$Ds_1 = (a|K \xrightarrow{c} (K + B\xi'\underline{\theta'})\mu'), \tag{1}$$
$$Ds_2 = (a\tau\underline{\sigma}|(K\tau\underline{\sigma} + X) \xrightarrow{c} (K\tau\underline{\sigma} + B\xi''\underline{\theta''} + X)\mu'') \tag{2}$$
$$\Longrightarrow \exists\delta \text{ such that } K\tau\mu'' = K\mu'\delta, \quad B\xi''\mu'' = B\xi'\mu'\delta,$$
$$\text{where } \delta \text{ is a renaming, if } \tau \text{ is a renaming.}$$

Proof
By definition of derivation step, we have:

$$var(a|K) \cap var((ht \longleftarrow B)\xi') = \varnothing, \tag{3}$$
$$var((a|K)\tau) \cap var((ht \longleftarrow B)\xi'') = \varnothing, \tag{4}$$
$$\mu' = mgu(a, (ht)\xi'), \quad \mu'' = mgu(a\tau, (ht)\xi''). \tag{5}$$

Let $\pi = \tau/var(a|K)$[3] and $\phi = ((\xi')^{-1}\xi'')/var((ht \longleftarrow B)\xi')$. By (3) it is:

$$domain(\pi) \cap domain(\phi) = \varnothing, \tag{6a}$$
$$(ht \longleftarrow B)\xi'\pi = (ht \longleftarrow B)\xi' \text{ and } (a|K)\phi = (a|K). \tag{6b}$$

[3] The notation $\tau/var(a|K)$ represents τ restricted to the variables of $a|K$.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

As a consequence of (6a), the union $(\pi \cup \phi)$ is a well defined substitution. Then, we may write that:

$$a(\pi \cup \phi)\mu'' =^{(6b)} a\pi\mu'' = a\tau\mu'' =^{(5)} (ht)\xi''\mu'' = (ht)\xi'(\xi')^{-1}\xi''\mu'' \\ = (ht)\xi'\phi\mu'' =^{(6b)} (ht)\xi'(\pi \cup \phi)\mu'',$$

so that $(\pi \cup \phi)\mu''$ is an unifier of a and $(ht)\xi'$. Since μ' is an mgu of a and $(ht)\xi'$, a substitution δ exists with:

$$(\pi \cup \phi)\mu'' = \mu'\delta. \tag{7}$$

Then, we have:

$$K\tau\mu'' = K\pi\mu'' =^{(6b)} K(\pi \cup \phi)\mu'' =^{(7)} K\mu'\delta, \tag{8a}$$

$$B\xi''\mu'' = B\xi'(\xi')^{-1}\xi''\mu'' = B\xi'\phi\mu'' =^{(6b)} B\xi'(\pi \cup \phi)\mu'' =^{(7)} B\xi'\mu'\delta. \tag{8b}$$

Now let us suppose that τ is a renaming. In this case, facts (3) and (4) become symmetric at all. As a consequence, by symmetry with respect to (8a) and (8b), a substitution γ exists such that $K\mu' = K\tau\mu''\gamma$ and $B\xi'\mu' = B\xi''\mu''\gamma$. Then we have:

$$(K\mu' + B\xi'\mu')\delta\gamma = (K\tau\mu'' + B\xi''\mu'')\gamma = K\mu' + B\xi'\mu'.$$

It is evident that δ is a renaming for $K\mu' + B\xi'\mu'$, then the thesis is verified. □

B Appendix

In this Appendix we provide a formal proof of the duplication theorem (Theorem 5.3). Such a proof exploits two lemmas which are given below. Lemma B1 establishes a condition which allows us to repeat derivations via a specialisation independent scheduling rule, when we pass from a goal G to a suitable kind of instantiations of G. Lemma B1 is a correspondent, for p-SLD derivations, of part (ii) of Strong Lifting Lemma (Gabrielli, Levi and Meo, 1996). Indeed, both part (ii) of the Strong Lifting Lemma and Lemma B1 can be seen as results about sufficient conditions for derivation lowering from a goal G to instantiations of G itself. Here a direct proof of Lemma B1 is given which takes into account technical aspects concerning our priority value mechanism. Lemma B1 does not relate resolvents, because it is not important for the purposes of this Appendix.

Lemma B.1

Let S be a specialisation independent scheduling rule, G a p-goal and ϕ a substitution. The following implication holds:

$$G \xrightarrow{S,X,\theta} \bullet \quad \Longrightarrow \quad G\theta\phi \xrightarrow{S,X} \bullet. \tag{1}$$

Proof

The proof is by induction on the length of X. If $\#X = 0$, the thesis is trivially true. For $\#X > 0$, let $G = a|F$, $X = c|H$ with $c = (ht \longleftarrow B)$, and rewrite derivation (1) as follows:

$$a|F \xrightarrow{S,c\xi,\gamma} (Q = (F + B\xi\underline{\pi})\gamma) \xrightarrow{S,H,\mu} \bullet, \quad \text{where } \gamma\mu = \theta. \tag{2}$$

Then, let us consider the substitution $\phi_g = \phi\sigma_g$, where σ_g is such that $(a|F)\theta\phi_g = G\theta\phi_g$ is ground. Since γ is an mgu of a and $(ht)\xi$, we have $a\gamma = (ht)\xi\gamma$, which means $a\theta\phi_g = a\gamma\mu\phi_g = (ht)\xi\gamma\mu\phi_g = (ht)\xi\theta\phi_g$. But $a\theta\phi_g$ is ground, so that we obtain

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

the equality $(a\theta\phi_g)\theta\phi_g = a\theta\phi_g = ((ht)\xi)\theta\phi_g$. In other words, $a\theta\phi_g$ and $(ht)\xi$ unify through the unifier $\theta\phi_g$. Moreover, the renamed clause $c\xi$ is obviously standardised apart with respect to the ground p-goal $(a|F)\theta\phi_g$, so that a derivation step like $(a|F)\theta\phi_g \xrightarrow{c\xi} \bullet$ exists. Thus, by completeness of S and Property A1, a derivation step also exists of the form:

$$(G\theta\phi_g = (a|F)\theta\phi_g) \xrightarrow{S,c\xi,\eta} (R = (F\theta\phi_g + B\xi\underline{\pi}')\eta). \tag{3}$$

Now, the substitution η is an mgu of $a\theta\phi_g$ and $(ht)\xi$, so that a substitution π exists with:

$$\theta\phi_g = \eta\pi. \tag{4}$$

On the other hand, since S is specialisation independent, step (3) is a congruent lowering of the first step of (2) by $\emptyset$, i.e.

$$\exists\underline{\rho} \text{ such that } F\underline{\rho} = F,\ B\underline{\pi}\underline{\rho} = B\underline{\pi}', \tag{5}$$

which implies:

$$Q\mu\phi_g = (F + B\xi\underline{\pi})\gamma\mu\phi_g\underline{\rho}\underline{\rho}^{-1} = (F\underline{\rho} + B\xi\underline{\pi}\underline{\rho})\theta\phi_g\underline{\rho}^{-1} =^{(5)} (F + B\xi\underline{\pi}')\theta\phi_g\underline{\rho}^{-1}.$$

But $F\theta\phi_g$ is ground, so that $(F\theta\phi_g)\theta\phi_g = F\theta\phi_g$. As a consequence:

$$Q\mu\phi_g = (F\theta\phi_g + B\xi\underline{\pi}')\theta\phi_g\underline{\rho}^{-1} =^{(4)} (F\theta\phi_g + B\xi\underline{\pi}')\eta\pi\underline{\rho}^{-1} = R\pi\underline{\rho}^{-1}.$$

By inductive hypothesis applied to the tail of (2), we have that $(Q\mu\phi_g = R\pi\underline{\rho}^{-1}) \xrightarrow{S,H} \bullet$, which by Lifting Lemma 4.2 implies that $R \xrightarrow{S,H} \bullet$. Now, by Property 4.3, the last obtained derivation can be combined with (3) yielding:

$$(G\theta\phi_g = G\theta\phi\sigma_g) \xrightarrow{S,X} \bullet.$$

By Lifting Lemma 4.2, we conclude $(G\theta\phi \xrightarrow{S,X} \bullet)$, so that the inductive step is completed. □

The following Lemma B2 is a special form of determinism lemma which holds for preq type stack-queue derivations. Roughly speaking, the lemma states that an A-preq type derivation, starting from a p-goal of the form $A|X$, can be replicated from a p-goal like $A\lambda\underline{\lambda}|Y$, where λ is a renaming. Note that no hypothesis is made on X and Y which can be completely unrelated. The intuitive explication is that only atoms deriving from A are rewritten so that neither X nor Y have any active role in the derivations. The formal statement and the proof of Lemma B2 are preceded by the quite simple Property B1.

Property B.1

Let SQ be a stack-queue scheduling rule. The following implication holds:

$$A|X \xrightarrow{SQ,D} Q, \text{ of } A\text{-preq type} \tag{1}$$

$$A\gamma\underline{\lambda}|Y \xrightarrow{SQ,D} R, \text{ of } (A\gamma\underline{\lambda})\text{-preq type, where } \gamma \text{ is a renaming} \tag{2}$$

$$\Longrightarrow \exists\delta,\underline{\delta} \text{ such that } R/(A\gamma\underline{\lambda}) = (Q/A)\delta\underline{\delta}, \text{ where } \delta \text{ is a renaming.}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Proof
By hypothesis, derivations (1) and (2) are of A-preq and $(A\gamma\underline{\lambda})$-preq type respectively, so that $D/A = D/(A\gamma\underline{\lambda}) = D$. Then, by lifting Lemma 4.2, a derivation exists like:

$$A \xrightarrow{S,D} T. \tag{3}$$

By lowering Lemma 4.1, applied to (3) and (1), a renaming α and a shifting $\underline{\alpha}$ exist with $Q/A = T\alpha\underline{\alpha}$. By lowering Lemma 4.1 applied to (3) and (2), a renaming β and a shifting $\underline{\beta}$ exist with $R/(A\gamma\underline{\lambda}) = T\beta\underline{\beta}$. Finally, we derive that:

$$R/(A\gamma\underline{\lambda}) = T\alpha\underline{\alpha}\alpha^{-1}\underline{\alpha}^{-1}\beta\underline{\beta} = (Q/A)\alpha^{-1}\underline{\alpha}^{-1}\beta\underline{\beta}. \quad \square$$

Lemma B.2 (*preq type determinism*)
Let SQ be a stack-queue scheduling rule and V any finite set of variables. Let $A|X$ and $A\lambda\underline{\lambda}|Y$ two p-goals, where λ is a renaming. The following implication holds:

$$A|X \xrightarrow{SQ,K,\psi} A^s|X\psi|A^q, \text{ of } A\text{-preq type}, \tag{1}$$
$$\Longrightarrow \exists\delta, \underline{\delta} \text{ and } D = (A\lambda\underline{\lambda}|Y \xrightarrow{SQ,K,\theta} A^s\delta\underline{\delta}|Y\theta|A^q\delta\underline{\delta}), \text{ of } (A\lambda\underline{\lambda})\text{-preq type},$$
$$\text{where } \delta \text{ is a renaming and } nvar(D) \cap V = \varnothing.$$

Proof
Let $A = a|F$. We show Lemma B2 by induction on the length of the template K. If $\#K = 0$, the assert is evident. If $\#K > 0$, let $K = c|H$ with $c = (ht \longleftarrow M_s|M_q)$. Derivation (1) can be rewritten as follows:

$$a|F|X \xrightarrow{SQ,c\xi,\mu} (Q = a^s|F\mu|X\mu|a^q) \xrightarrow{SQ,H,\sigma} A^s|X\psi|A^q \tag{1a}$$
$$\text{with } a^s = M_s\xi\underline{\alpha}\mu \text{ and } A^s = (a^s|F\mu)^s. \tag{1b}$$

Then, by Property 3.3 with reference to the first step of (1a), a derivation step Ds exists such that:

$$Ds = ((a|F)\lambda\underline{\lambda}|Y \xrightarrow{SQ,c\tau,\eta} R), \text{ with } nvar(Ds) \cap V = \varnothing. \tag{2}$$

Since the selection rule SQ is stack queue, it must be:

$$R = M_s\tau\underline{\gamma}\eta|F\lambda\underline{\lambda}\eta|Y\eta|M_q\tau\underline{\gamma}\eta.$$

By Property B1 applied to derivation step (2) and the first step of (1a), a renaming β and a shifting $\underline{\beta}$ exist with:

$$M_s\tau\underline{\gamma}\eta|F\lambda\underline{\lambda}\eta|M_q\tau\underline{\gamma}\eta = R/^2/(A\lambda\underline{\lambda}) = (Q/^{1a}/A)\beta\underline{\beta} = (a^s|F\mu|a^q)\beta\underline{\beta}.$$

Now, by (1b) it is $\#a^s = \#M_s\tau\underline{\gamma}\eta$, so that the equality $M_s\tau\underline{\gamma}\eta|F\lambda\underline{\lambda}\eta = (a^s|F\mu)\beta\underline{\beta}$ holds, by Property (3.1-i). Thus, the inductive hypothesis can be applied to the tail of (1a). As a consequence, a p-SLD derivation D' exists, which is of $(M_s\tau\underline{\gamma}\eta|F\lambda\underline{\lambda}\eta)$-preq type and has the following form:

$$R \xrightarrow{SQ,H,\pi} ((a^s|F\mu)^s\delta'\underline{\delta}'|Y\eta\pi|Z =^{(1b)} A^s\delta'\underline{\delta}'|Y\eta\pi|Z), \tag{3}$$
$$\text{with } nvar(D') \cap (var((a|F)\lambda\underline{\lambda}|Y) \cup nvar(Ds) \cup V) = \varnothing. \tag{3a}$$

On the basis of (3a) above, derivation step (2) and derivation (3) can be combined to yield the derivation D:

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

$$D = (A\lambda\underline{\lambda}|Y \overset{SQ,K}{\longrightarrow} A^s\delta'\underline{\delta}'|Y\eta\pi|Z), \tag{4}$$

where D is of $(A\lambda\underline{\lambda})$-preq type. The thesis is now proven. Indeed, by Property B1 applied to derivations (1) and (4), a renaming δ and a shifting $\underline{\delta}$ exist with $A^s\delta'\underline{\delta}'|Z = (A^s|A^q)\delta\underline{\delta}$, so that by Property (3.1-i) we have $A^s\delta'\underline{\delta}' = A^s\delta\underline{\delta}$ and $Z = A^q\delta\underline{\delta}$. The fact that $nvar(D) \cap V = \varnothing$ follows from (2) and (3a). □

Theorem B.1 (*Theorem 5.3 – duplication theorem*)
Let SQ be a stack-queue scheduling rule. Given two p-goals of the form $A|B|C|D$ and $A|B|C|B\underline{\pi}|D$, the following implication holds:

$$(A|B|C|D \overset{SQ,X,P}{\longrightarrow} Q) \tag{1}$$
$$\Longrightarrow \exists Y \text{ such that } (A|B|C|B\underline{\pi}|D \overset{SQ,Y,P}{\longrightarrow} R)$$
$$\text{with } X \subseteq_L Y \text{ and } \#Q \preceq \#R.$$

Proof
Let $\Delta(SQ, n)$ denote the subset of $\Delta(SQ)$, such that for any derivation Dr in $\Delta(SQ, n)$ it is $\#Dr \preceq n$, where $\#Dr$ denotes the length of Dr. We show the thesis by induction on n, i.e. we show that the thesis holds when derivation (1) belongs to $\Delta(SQ, n)$, for any $n \succeq 0$. The fact is obvious for $\Delta(SQ, 0)$. In order to prove the inductive step from $\Delta(SQ, n-1)$ to $\Delta(SQ, n)$, for $n > 0$, let us consider a derivation like:

$$(A|B|C|D \overset{X,P,\theta}{\longrightarrow} Q) \in \Delta(SQ, n), \tag{1a}$$

and show that $(A|B|C|B\underline{\pi}|D \overset{SQ,Y,P}{\longrightarrow} R)$ exists with $X \subseteq_L Y$ and $\#Q \preceq \#R$. Actually, the proof of the inductive step will be organised in two phases:

- first, the inductive step is shown in the case that the initial p-goal $A|B|C|D$ is ground,
- then, the validity of the inductive step is extended to generic initial p-goals.

Let us recall that the sketch of Section 5.3 was given in the simplifying hypothesis that: every clause body introduces no new variable and initial p-goals are ground. In this sense, we may say that the first phase removes the first restriction, while the second one is retained. In the second phase, also the restriction on the groundness of initial goals is overcome.

First phase (*the initial p-goal $A|B|C|D$ is ground*).

With reference to (1a), the following three possible situations must be taken into account. Then, we start with Case 3, which is the most significant one.

1. derivation (1a) is of $(A|B|C)$-preq type,
2. derivation (1a) is of $(A|B|C|D)$-preq type, and not of $(A|B|C)$-preq type,
3. derivation (1a) is not of $(A|B|C|D)$-preq type.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

Case 3.

Derivation (1a) has the following form:

$$A|B|C|D \xrightarrow{H} B|C|D|A^q \xrightarrow{K} C|D|A^q|B^q \xrightarrow{M}$$
$$D|A^q|B^q|C^q \xrightarrow{N} A^q|B^q|C^q|D^q \xrightarrow{T} Q, \tag{2}$$

where $H|K|M|N|T = X$.

In fact, since $A|B|C|D$ is ground, in each of the four initial segments of (2) only standardisation apart variables, introduced in the same segment, can be instantiated. In particular, $var(A^q)$ are not instantiated in the second segment, $var(A^q|B^q)$ are not in the third segment, and $var(A^q|B^q|C^q)$ are not in the fourth one. It is evident, as a consequence, that the p-goal $A^q|B^q|C^q|D^q$ consists of four subgoals without common variables. Now, since also $B\underline{\pi}$ is ground, a derivation can be constructed through five successive applications of Lemma B2, as depicted below:

$$A|B|C|B\underline{\pi}|D \xrightarrow{SQ,H} B|C|B\underline{\pi}|D|A^q\alpha\underline{\alpha} \xrightarrow{SQ,K}$$
$$C|B\underline{\pi}|D|A^q\alpha\underline{\alpha}|B^q\beta\underline{\beta} \xrightarrow{SQ,M} B\underline{\pi}|D|A^q\alpha\underline{\alpha}|B^q\beta\underline{\beta}|C^q\gamma\underline{\gamma} \xrightarrow{SQ,K} \tag{3}$$
$$D|A^q\alpha\underline{\alpha}|B^q\beta\underline{\beta}|C^q\gamma\underline{\gamma}|B^q\phi\underline{\phi} \xrightarrow{SQ,N} (Z = A^q\alpha\underline{\alpha}|B^q\beta\underline{\beta}|C^q\gamma\underline{\gamma}|B^q\phi\underline{\phi}|D^q\delta\underline{\delta})$$

where α, β, γ, ϕ and δ are renamings.

At each application of Lemma B2, a segment of derivation (3) is obtained on the basis of a corresponding segment of derivation (2). Moreover, Lemma B2 assures that each new segment can be freely standardised apart, so that each segment can be readily added to the sequence of its predecessors in (3). Note that the second segment of (2) is considered twice, in order to generate both the second and the fourth segment of (3). In analogy with derivation (2), the final p-goal Z of derivation (3) consists of five subgoals without common variables. As a consequence, the five renamings α^{-1}, β^{-1}, γ^{-1}, ϕ^{-1} and δ^{-1} have disjoint domains, so that they can be joined in order to form a unique substitution

$$\xi = (\alpha^{-1} \cup \beta^{-1} \cup \gamma^{-1} \cup \phi^{-1} \cup \delta^{-1}). \tag{4a}$$

Then, let us consider the p-goal $A^q|B^q|C^q|B^q\underline{\pi}'|D^q$, where $\underline{\pi}'$ is a suitable shifting. By (4a) and Property 3.2, we have that:

$$A^q|B^q|C^q|B^q\underline{\pi}'|D^q =^{(Ax-iii)} (A^q\underline{\alpha}|B^q\underline{\beta}|C^q\underline{\gamma}|B\underline{\phi}|D^q\underline{\delta})\underline{\sigma} =^{(4a)}$$
$$(A^q\alpha\underline{\alpha}|B^q\beta\underline{\beta}|C^q\gamma\underline{\gamma}|B^q\phi\underline{\phi}|D^q\delta\underline{\delta})\xi\underline{\sigma} = Z\,\xi\underline{\sigma}. \tag{4b}$$

By construction of (2) the derivation $(A^q|B^q|C^q|D^q \xrightarrow{T} Q)$ belongs to $\Delta(SQ, m)$, with $m < n$. By inductive hypothesis, a derivation exists of the form:

$$(Z\,\xi\underline{\sigma} = A^q|B^q|C^q|B^q\underline{\pi}'|D^q) \xrightarrow{SQ,Y',P} R', \tag{5}$$

$$\text{with} \quad T \subseteq_L Y' \quad \text{and} \quad \#Q \preceq \#R'. \tag{5a}$$

By (5) and Lifting Lemma 4.2 a derivation exists like:

$$Z \xrightarrow{SQ,Y'} \bullet, \tag{6}$$

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

By Property 4.3, derivations (3) and (6) can be combined to yield a derivation of the form:

$$A|B|C|B\underline{\pi}|D \overset{SQ,(H|K|M|K|N).P}{\longrightarrow} Z \overset{SQ,Y'.P}{\longrightarrow} R, \tag{7}$$

where, by lowering Lemma 4.1 applied to (5) and the tail of (7), it is $\#R = \#R'$. Finally:

$$X = H|K|M|N|T \subseteq_L^{(5a)} (H|K|M|K|N|Y') \text{ and } \#Q \preceq^{(5a)} \#R' = \#R.$$

Case 2.

Derivation (1a) has the following form:

$$A|B|C|D \overset{H|K|M|N}{\longrightarrow} D^s|A^q|B^q|C^q|D^q, \text{ with } H|K|M|N = X.$$

Analogously to preceding case (3), through Lemma B2 a derivation can be constructed like:

$$A|B|C|B\underline{\pi}|D \overset{SQ,(H|K|M|K|N).P}{\longrightarrow} D^s\delta\underline{\delta}|A^q\alpha\underline{\alpha}|B^q\beta\underline{\beta}|C^q\gamma\underline{\gamma}|B^q\phi\underline{\phi}|D^q\delta\underline{\delta}.$$

Case 1.

Derivation (1a) has the form:

$$A|B|C|D \overset{X}{\longrightarrow} (A|B|C)^s|D|(A|B|C)^q.$$

Through Lemma B2, a derivation can be constructed like:

$$(A|B|C)|B\underline{\pi}|D \overset{SQ,X}{\longrightarrow} (A|B|C)^s\gamma\underline{\gamma}|B\underline{\pi}|D|(A|B|C)^q\gamma\underline{\gamma}.$$

Second phase (*the initial p-goal $A|B|C|D$ is generic*).

In the preceding first phase of this proof, the inductive step is verified in the hypothesis that the initial p-goal $A|B|C|D$ is ground. Now consider a generic p-goal of the form $A|B|C|D$. With reference to (1a), let ϕ_g be a grounding substitution for $(A|B|C|D)\theta$. By Lemma B1, a derivation exists such that:

$$((A|B|C|D)\theta\phi_g \overset{X}{\longrightarrow} Q') \in \Delta(SQ, n), \tag{8}$$

where, by lowering Lemma 4.1 applied to (1a) and (8), we have:

$$\#Q' = \#Q. \tag{8a}$$

Since the inductive hypothesis is already proven for ground initial goals, by (8) a derivation exists:

$$(A|B|C|B\underline{\pi}|D)\theta\phi_g \overset{SQ,Y.P}{\longrightarrow} R', \tag{9}$$

$$\text{with } X \subseteq_L Y \text{ and } \#Q' \preceq \#R'. \tag{9a}$$

Then, by lifting Lemma 4.2 a derivation exists:

$$A|B|C|B\underline{\pi}|D \overset{SQ,Y.P}{\longrightarrow} R, \tag{10}$$

where, by lowering Lemma 4.1 applied to (9) and (10), we have

$$X \subseteq_L^{(9a)} Y \text{ and } \#Q =^{(8a)} \#Q' \preceq^{(9a)} \#R' =^{(Lem.4.1)} \#R.$$

As a consequence, the induction step is completely verified. □

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press

References

Apt, K. R. (1990) Logic Programming. In: van Leeuwen, J. (ed.), *Handbook of Theoretical Computer Science*, Vol. B, pp. 493–574. Elsevier.

Apt, K. R. (1998) *From Logic Programming to Prolog*. Prentice-Hall.

Apt, K. R., Bol, R. N. and Klop, J. W. (1989) On the Safe Termination of Prolog Programs. In: G. Levi and M. Martelli (eds.), *Proc. Sixth Int. Conf. on Logic Programming*, pp. 353–368. MIT Press.

Apt, K. R. and Pedreschi, D. (1993) Reasoning about Termination of Pure Prolog Programs *Information and Computation*, **106**, 109–157.

Bol, R. N. (1992) Generalising Completeness Results for Loop Checks in Logic Programming *Theoretical Computer Science*, **104**, 3–28.

Bol, R. N, Apt, K. R. and Klop, J. W. (1991) An Analysis of Loop Checking Mechanisms for Logic Programs *Theoretical Computer Science*, **86**, 35–79 .

Bol, R. N. and Degersted, L. (1998) Tabulated Resolution for the well-founded Semantics *Journal of Logic Programming*, **34**, 67–109 .

De Schreye, D. and S. Decorte, S. (1994) Termination of Logic Programs: the never-ending story *Journal of Logic Programming*, **19/20**, 199–260.

Dietrich, S. W. (1987) Extension Tables: Memo Relations in Logic Programming. *Proc. of Symp. on Logic Programming*, pp. 264–273. IEEE Press.

Ferrucci, F., Pacini, G. and Sessa, M. I. (1995) Redundancy Elimination and Loop Checks for Logic Programs *Information and Computation*, **119**, 137–153.

Gabbrielli, M., Levi, G. and Meo, M. C. (1996) Resultants Semantics for Prolog *Journal of Logic Computation*, **6**, 491–521.

Joyner, W. H. Jr. (1976) Resolution Strategies as Decision Procedures *Journal of the Association for Computing Machinery*, **23**, 398–417.

Knuth, D. E. (1997) *The Art of Computer Programming* Vol. 1. Addison-Wesley.

Kowalski, R. (1979) Algorithm=Logic+Control *Comm. ACM*, **22**, 424–435.

Lloyd, J. W. (1987) *Foundations of Logic Programming*. Springer Verlag.

Pacini, G. & Sessa, M. I. (2000) Loop Checking in SLD-derivations by Well-Quasi-Ordering of Goals, *Theoretical Computer Science*, **238**, 221-246.

Ramakrishnan, I. V., Rao, P., Sagonas, K., Swift T. and Warren, D. S. (1999) Efficient Access Mechanisms for Tabled Logic Programs, *Journal of Logic Programming*, **38**, 31–54.

Smith, D. E., Genesereth, M. R. and Ginsberg, M. L. (1986) Controlling Recursive Inference *Artificial Intelligence*, **30**, 343–389.

Tamaki, T. and T. Sato (1986) OLD Resolution with Tabulation. In: Shapiro, E. (ed.), *Proc. Third Int. Conf. on Logic Programming: Lecture Notes in Computer Science 225*, pp. 84–98. Springer-Verlag.

Van Gelder, A. (1987) Efficient Loop Detection in PROLOG using the Tortoise-and-Hare Technique *Journal of Logic Programming*, **4**, 23–31.

Vieille, L. (1989) Recursive Query Processing: The Power of Logic, *Theoretical Computer Science*, **69**, 1–53.

https://doi.org/10.1017/S1471068401001235 Published online by Cambridge University Press