LOOP DETECTION IN PROLOG BY SEARCHING FOR PRIMITIVE CYCLIC GOALS

Dimiter Skordev
Sofia University, Sofia, Bulgaria
Igor Đurđanović
Paderborn University, Paderborn, Germany

1 Introduction

A new method will be proposed for the detection of some loops during depth-first search in Prolog, and an implementation of this method will be described.[1] The method is a modification of another one presented in the paper [3]. As in [3], we shall consider only Prolog programs consisting of Horn clauses and goals consisting of atomic formulas. The new method has the following advantage: only the leftmost formulas of some goals appearing in the execution process have to be compared now instead of beginnings with a flexible length as it was previously. In this respect, the method has a certain common feature with one proposed by Van Gelder in [4].[2]

Throughout the paper, a Prolog program $\mathcal{P}$ will be supposed to be given. We adopt the leftmost selection rule, and then any given query uniquely determines (up to renamings of variables) a *depth-first search path.* This path is a sequence of goals, which is defined as follows. Let us call a goal G *terminal for the depth-first search* (or simply *terminal*, for short) if this goal is empty or has a leftmost formula unifiable with the head of none of the clauses in $\mathcal{P}$ (of course, unification possibly preceded by an appropriate renaming of program variables is considered). To each non-terminal goal G, a goal $\sigma(G)$, *the* SLD-*successor of* G, corresponds. Namely, $\sigma(G)$ is the result obtained from G by means of an SLD-resolution step on the basis of $\mathcal{P}$ (the resolution is done upon the leftmost formula of G, and the first clause in $\mathcal{P}$ with a head unifiable with G is used). In general, the SLD-successor of G could contain variables whose choice is not uniquely determined and even may depend on previous variable bindings, but we overcome these problems by identifying any two goals or atomic formulas which are identical up to some renaming of variables (such goals or formulas will be called to be *essentially the same*). By definition, the initial member of the depth-search path of $\mathcal{P}$ on a given query is the goal from the query, and, whenever a member G of the depth-first search path is present, then $\sigma(G)$ is the next member of the path in case G is non-terminal, and there is no next member of the path otherwise.

The depth-first search path of $\mathcal{P}$ on a given query may be finite or infinite, and in the second of these cases it is said that *the depth-first search of $\mathcal{P}$ on the query enters into a loop.* We study the problem of detection of some of these loops during the execution of the program. The new method which we propose detects the same kind of loops as the former one, i.e. all

[1]The method has been designed by the first author during a stay at the Paderborn University, and the implementation is done by the second author. The research of the first author has been supported by a DAAD grant and by the Bulgarian Ministry of Science and Higher Education - Contract No. MM 43, 1991.

[2]Unfortunately, the method from [4] is not correct, as shown in the note [2]. A revised version of that method is proposed in Van Gelder's response [5] to [2].

loops having the periodic feature described in [3]. The alternative description of such loops by means of the notion of a primitive cyclic goal plays a crucial role in the reasoning about the proposed method.

An atomic formula A will be called *cyclic* (with respect to the given program $\mathcal{P}$) if the depth-first search path of $\mathcal{P}$ on the query $?- A$, besides its initial member, contains some further one with a leftmost formula essentially the same as A. More precisely, we shall say that A is *cyclic with period* r if the above mentioned further member of the depth-first search path in question is obtained from the initial one by means of r consecutive applications of the operation σ. A goal will be called *primitive cyclic* (*with period* r) if this goal is non-empty, and its leftmost formula is cyclic (with period r). It will be proved that the execution of $\mathcal{P}$ on a given query enters into a loop of the kind studied in [3] if and only if the depth-first search path of $\mathcal{P}$ on this query has some primitive cyclic member.

2 Description of the loop detection method

For the method which we are going to describe, a computable strictly increasing infinite sequence $\tau_0 < \tau_1 < \tau_2 < \tau_3 < \ldots$ of natural numbers must be chosen such that the set of the differences $\tau_{i+1} - \tau_i$ is unbounded (in any practical application, only some finite initial part of this sequence will be used). The set of the members of this sequence will be further denoted by $\mathcal{T}$.

We shall first give a non-formal description of the proposed method. For short, let us call the number of the atomic formulas in a goal G *the length of* G (this number will be further denoted by $|G|$). The application of the method can be characterized as carrying out depth-first search steps accompanied by certain loop detection activities. More concretely, the length and the leftmost formula of the current goal have to be saved at appropriate moments and to be used further for the loop detection during a certain period of time, and this must be repeated until possibly the depth-first search terminates or a loop is detected. The moments for saving are the ones belonging to $\mathcal{T}$ and those ones, at which, so to say, the saved information becomes obsolete. The last happens when there are no more descendants of the saved formula in the current goal, and such a state of affairs can be noticed by seeing that the length of the current goal becomes less than the previously saved length. The saved information is used for the loop detection in the following way: the length and the leftmost formula of the current goal are compared with the saved ones, and if the length of the current goal is not less than the saved length, and the compared two atomic formulas turn out to be essentially the same, then a loop is detected.

To give a more precise description of the method, we introduce the notion of an *execution-detection state*. By definition, this is an arbitrary quadruple $<t, l, A, G>$, where t and l are natural numbers, A is some atomic formula or the empty string, and G is some goal. It will be said that *a loop is detected at* $<t, l, A, G>$ if $|G| \geq l$, A is an atomic formula, and G has a leftmost formula which is essentially the same as A. An execution-detection state will be said to be a *closing* one if its last component is a terminal goal or a loop is detected at this state. The detection method consists in constructing consecutively execution-detection states in a certain appropriate way, until possibly a closing execution-detection state is obtained. The process of constructing them is defined as follows. To examine the execution of $\mathcal{P}$ on a given query, we start with the execution-detection state $<0, 0, \epsilon, Q>$, where ϵ is the empty string, and Q is the goal from the query. Further, whenever an execution-detection state $<t, l, A, G>$ is constructed, and this state is not a closing one, we form a new execution-detection state $<t', l', A', G'>$ according to the following rules: (a) $t' = t+1$, and $G' = \sigma(G)$; (b) if $t \in \mathcal{T}$ or $|G| < l$, then $l' = |G|$, and A' is the leftmost formula of G, otherwise $l' = l$, $A' = A$ (it will be said that an execution-detection state $<t, l, A, G>$ *invokes saving* if this state is not a closing one, and the first case in rule (b) is present).

The sequence of execution-detection states defined in the above way will be called *the execution-detection path of $\mathcal{P}$ on the given query.* An example of such a path is shown on Figure 1, where it is assumed that $\tau_0 = 0$, $\tau_1 = 1$, $\tau_2 \geq 4$.

Program:

```
    p(X,a) :- p(X,f(Y)),p(Z,Y),p(Y,Z).
    p(X,f(a)).
```

Query:

```
    ?- p(U,U).
```

Path:

```
    <0, 0, ϵ,          :- p(U,U).>,                                  saving invoked
    <1, 1, p(U,U),     :- p(a,f(Y)),p(Z,Y),p(Y,Z).>,                 saving invoked
    <2, 3, p(a,f(Y)), :- p(Z,a),p(a,Z).>,                           saving invoked
    <3, 2, p(Z,a),     :- p(Z,f(Y1)),p(Z1,Y1),p(Y1,Z1),p(a,Z).>,
    <4, 2, p(Z,a),     :- p(Z1,a),p(a,Z1),p(a,Z).>.                  loop detected!
```

Figure 1: Execution-detection path

For the general case, we shall prove that a loop is detected at a member of the execution-detection path of $\mathcal{P}$ on a query if and only if there is a primitive cyclic member in the depth-first search path of $\mathcal{P}$ on this query. The implication from left to right will be established by seeing that if a loop is detected at a member of the execution-detection path then the last component of this member is a primitive cyclic goal. The proof of the converse implication will use, roughly speaking, the following fact: if at a moment of the depth-first search the current goal is not primitive cyclic, but some previous member of the depth-first search path is, then a primitive cyclic goal will be reached further again after one or more instances of decreasing of the minimal length of the goals from that moment on.

3 Proofs

A part of the results in this section can be found in [3] (where the denotation ρ is used instead of σ), but we present them again for the convenience of the reader. Those of the results whose proofs reduce to a straight-forward verification will be only formulated.

Adopting the identification of goals mentioned in the introduction, we shall denote the set of all goals by $\mathcal{G}$. Of course, σ is a partial operation in $\mathcal{G}$. Another partial unary operation in this set will be used a little further, namely the operation of deleting the rightmost formula of a non-empty goal; this operation will be denoted by π. We shall introduce also a partial ordering in $\mathcal{G}$. Let $G_1, G_2 \in \mathcal{G}$. We shall say that G_1 is a *restriction* of G_2, and we shall denote this by writing $G_1 \leq G_2$, if G_1 is essentially the same as some beginning of G_2. Obviously, the relation defined in this way is reflexive and transitive, and the conjunction of $G_1 \leq G_2$ and $G_2 \leq G_1$ implies that G_1 and G_2 are essentially the same.

Lemma 1 *If $G_1 \leq G$, $G_2 \leq G$, and $|G_1| \leq |G_2|$, then $G_1 \leq G_2$.*

Lemma 2 *If $G_1 \leq G_2$, $G_2 \in \mathrm{dom}(\sigma)$, and G_1 is non-empty, then $G_1 \in \mathrm{dom}(\sigma)$.*

Lemma 3 *Let i be a natural number. If $G_1 \leq G_2$, and $G_1 \in \mathrm{dom}(\sigma^i)$, then $G_2 \in \mathrm{dom}(\sigma^i)$, $\sigma^i(G_1) \leq \sigma^i(G_2)$, and $|\sigma^i(G_2)| - |\sigma^i(G_1)| = |G_2| - |G_1|$.*

Proof. Direct verification in the case of $i = 1$ and then proceeding by induction.

Let r be a positive integer. A goal G will be called *strongly cyclic with period r* if $G \in \mathrm{dom}(\sigma^r)$, and $G \leq \sigma^r(G)$. A goal will be called *cyclic with period r* if some of its beginnings is strongly cyclic with period r. Using Lemma 3 with $i = 1$, one proves consecutively the next two lemmas.

Lemma 4 *If r is a positive integer, and G is a goal, strongly cyclic with period r, then $G \in \mathrm{dom}(\sigma)$, and $\sigma(G)$ is again strongly cyclic with period r.*

Lemma 5 *If r is a positive integer, and G is a goal, cyclic with period r, then $G \in \mathrm{dom}(\sigma)$, and $\sigma(G)$ is again cyclic with period r.*

From Lemma 5 and the definition of the notion of depth-first search path, we get

Corollary 1 *If some member of the depth-first search path of $\mathcal{P}$ on a query is cyclic with some positive period then the execution of $\mathcal{P}$ on that query enters into a loop.*

If the assumption of Corollary 1 is satisfied for a given query, then we shall say that *the depth-first search of $\mathcal{P}$ on the query enters into a cyclic loop* (these are the loops studied in [3]). Clearly, each primitive cyclic goal with period r is cyclic with period r. Hence, if some member of the depth-first search path of $\mathcal{P}$ on a query is primitive cyclic, then the depth-first search of $\mathcal{P}$ on this query enters into a cyclic loop. The converse statement is less obvious, but it will be proved soon. For proving it, some lemmas more will be needed.

Lemma 6 *Let r be a positive integer, and G be a goal strongly cyclic with period r. Let $G_0 \leq G$ and $G_0 \in \mathrm{dom}(\sigma^r)$. Then the goal G_0 is also strongly cyclic with period r.*

Proof. We have $G \leq \sigma^r(G)$, and hence $|G| \leq |\sigma^r(G)|$, $G_0 \leq \sigma^r(G)$. By Lemma 3, $\sigma^r(G_0) \leq \sigma^r(G)$, and the equality $|\sigma^r(G)| - |\sigma^r(G_0)| = |G| - |G_0|$ holds. It is clear now that $|G_0| \leq |\sigma^r(G_0)|$, and thus, by Lemma 1, $G_0 \leq \sigma^r(G_0)$.

Lemma 7 *Let r be a positive integer, and $G \in \mathrm{dom}(\sigma^r)$. Let v be the minimal one among the lengths of the goals $\sigma^i(G)$, $i = 0, 1, \ldots, r-1$. Then $v > 0$, $G \in \mathrm{dom}(\pi^v)$, $\pi^{v-1}(G) \in \mathrm{dom}(\sigma^r)$, $\pi^v(G) \notin \mathrm{dom}(\sigma^r)$, and $|\sigma^i(\pi^{v-1}(G))| = 1$ for any natural number i which is less than r and satisfies the condition $|\sigma^i(G)| = v$.*

P r o o f. All $\sigma^i(G)$, $i = 0, 1, \ldots, r-1$, belong to $\mathrm{dom}(\sigma)$, hence they are non-empty, and therefore $v > 0$. Of course, $G \in \mathrm{dom}(\pi^v)$, since $|G| \geq v$. By induction on i, we shall show that $\pi^{v-1}(G) \in \mathrm{dom}(\sigma^i)$ for $i = 0, 1, \ldots, r$. For $i = 0$ the validity of this statement is trivial. Assume now that $i < r$ and $\pi^{v-1}(G) \in \mathrm{dom}(\sigma^i)$. By Lemma 3, $\sigma^i(\pi^{v-1}(G)) \leq \sigma^i(G)$, and $|\sigma^i(G)| - |\sigma^i(\pi^{v-1}(G))| = |G| - |\pi^{v-1}(G)| = v - 1$. Since $|\sigma^i(G)| \geq v$, we see that $\sigma^i(\pi^{v-1}(G))$ is a non-empty restriction of $\sigma^i(G)$. Then an application of Lemma 2 shows that $\sigma^i(\pi^{v-1}(G)) \in \mathrm{dom}(\sigma)$, i.e. $\pi^{v-1}(G) \in \mathrm{dom}(\sigma^{i+1})$, and thus the induction step is completed. Now let i be a natural number less than r and such that $|\sigma^i(G)| = v$. The result of the above application of Lemma 3 shows that $|\sigma^i(\pi^{v-1}(G))| = 1$ for this choice of i. We shall show now that, for the same i, $\pi^v(G) \notin \mathrm{dom}(\sigma^{i+1})$, and hence $\pi^v(G) \notin \mathrm{dom}(\sigma^r)$. Actually, if we suppose that $\pi^v(G) \in \mathrm{dom}(\sigma^{i+1})$, then we can conclude that $\pi^v(G) \in \mathrm{dom}(\sigma^i)$ and, making use of Lemma 3, see that $|\sigma^i(G)| - |\sigma^i(\pi^v(G))| = |G| - |\pi^v(G)| = v$. From here, it would follow that $\sigma^i(\pi^v(G))$ is empty, and this contradicts the assumption that $\pi^v(G) \in \mathrm{dom}(\sigma^{i+1})$.

Lemma 8 *Let r be a positive integer, and G be a goal cyclic with period r. Then those of the goals $\sigma^i(G)$, $i = 0, 1, \ldots, r-1$, which have a minimal length, are primitive cyclic with period r.*

P r o o f. Let v be the minimal one among the lengths of the mentioned goals, and let i be a natural number less than r such that $|\sigma^i(G)| = v$. By Lemmas 7 and 3, $\pi^{v-1}(G)$ is the shortest beginning of G belonging to $\mathrm{dom}(\sigma^r)$. Let G_1 be this beginning, and G_2 be a beginning of G, which is strongly cyclic with period r. Then G_1 is a beginning of G_2, and an application of Lemma 6 shows that G_1 is also strongly cyclic with period r. By Lemma 4, the goal $\sigma^i(G_1)$ will be also strongly cyclic with period r. By Lemma 7, $|\sigma^i(G_1)| = 1$. Hence the unique atomic formula of $\sigma^i(G_1)$ will be cyclic with period r. Since, by Lemma 3, $\sigma^i(G_1) \leq \sigma^i(G)$, we thus see that $\sigma^i(G)$ has a leftmost formula cyclic with period r, and hence this goal is primitive cyclic with period r.

Corollary 2 *The depth-first search of $\mathcal{P}$ on a query enters into a cyclic loop if and only if there is some primitive cyclic member of the depth-first search path of $\mathcal{P}$ on this query.*

The following lemma can be regarded as a partial conversion of Lemma 8.

Lemma 9 *If a goal G is primitive cyclic then $|\sigma^i(G)| \geq |G|$ for any natural number i.*

P r o o f. Let G be a primitive cyclic goal, and H be the beginning of G consisting only of its leftmost formula. Then H is strongly cyclic with some positive period, and therefore $H \in \mathrm{dom}(\sigma^i)$ for any natural number i. Consequently, $\sigma^i(H)$ is non-empty for any natural number i. By Lemma 3, we have the inequality $|\sigma^i(G)| - |\sigma^i(H)| = |G| - |H| = |G| - 1$, and this inequality together with the inequality $|\sigma^i(H)| \geq 1$ implies $|\sigma^i(G)| \geq |G|$.

Now we are going to give some lemmas concerning the execution-detection path of $\mathcal{P}$ on some fixed given query whose goal will be denoted by Q. For short, we shall usually omit the references to the program and to the query when mentioning this path or the depth-first search path of $\mathcal{P}$ on the same query.

Lemma 10 *For each natural number t, either there is a member with first component t in the execution-detection path (the member in question is unique in this case), or there is a closing member with first component less than t in the path. If a member of the path has first component t then the last component of this member is $\sigma^t(Q)$, and there is a member with first component t' in the path for each natural number t' less than t. In case a member of the path has first component t with $t \leq \tau_0$, then the second and the third components of this member are 0 and ϵ, respectively.*

Lemma 11 *Let $E = <t, l, A, G>$ be a member of the execution-detection path, r be a positive integer, and the following conditions be satisfied: $G \in \mathrm{dom}(\sigma^r)$, E invokes saving, no member of the execution-detection path with first component strictly between t and $t+r$ invokes saving, and at no such member a loop is detected. Let l' and A' be the length and the leftmost formula of G, respectively. Then $<t + r, l', A', \sigma^r(G)>$ is a member of the execution-detection path, and the goal whose unique formula is A' belongs to $\mathrm{dom}(\sigma^r)$.*

Proof. The statement that $<t + r, l', A', \sigma^r(G)>$ is a member of the execution-detection path can be proved by means of an induction showing that $<t + i, l', A', \sigma^i(G)>$ is a member of the execution-detection path for $i = 1, 2, \ldots, r$. The assumptions of the lemma show that l' is the minimal one among the lengths of the goals $\sigma^i(G)$, $i = 0, 1, \ldots, r - 1$. Hence, by Lemma 7, $\pi^{l'-1}(G) \in \mathrm{dom}(\sigma^r)$.

Lemma 12 *If a loop is detected at a member of the execution-detection path, then the last component of this member is a primitive cyclic member of the depth-first search path.*

Proof. Let a loop be detected at the member $E_1 = <t_1, l_1, A_1, G_1>$ of the execution-detection path. Hence, by the corresponding definition, $|G_1| \geq l_1$, A_1 is an atomic formula, and G_1 has a leftmost formula which is essentially the same as A_1. The last statement in Lemma 10 shows that $t_1 > \tau_0$. Let t_0 be the greatest one among the natural numbers t less that t_1 such that the member with first component t of the execution-detection path invokes saving (at least one such t exists, namely τ_0). Let $E_0 = <t_0, l_0, A_0, G_0>$ be the member of the execution-detection path with first component t_0. Then we can apply Lemma 11 with $E = E_0$, $r = t_1 - t_0$ and (taking into account the fact that $\sigma^r(G_0)$ is G_1) get the conclusion that l_1 and A_1 are the length and the leftmost formula of G_0, respectively, as well as the conclusion that $:\!-\ A_1. \in \mathrm{dom}(\sigma^r)$. By Lemma 3, $\sigma^r(\ :\!-\ A_1.\ ) \leq G_1$, and $|G_1| - |\sigma^r(\ :\!-\ A_1.)| = |G_0| - |\ :\!-\ A_1.| = l_1 - 1$. Since $|G_1| \geq l_1$, we see that the goal $\sigma^r(\ :\!-\ A_1.\ )$ is not empty and has essentially the same leftmost formula as G_1. Hence $\sigma^r(\ :\!-\ A_1.\ )$ has a leftmost formula essentially the same as A_1, and therefore A_1 is a cyclic formula. Of course, the leftmost formula of G_1 will be then cyclic too, and we see that G_1 is a primitive cyclic goal. By Lemma 10, this goal is a member of the depth-first search path on the given query.[3]

We are now ready to prove the main result about the proposed loop detection method.

Theorem 1 *For any given query, the following conditions are equivalent:*

[3]This will be not the first primitive cyclic member of the path, since G_0 will be also primitive cyclic.

(A) *The execution-detection path of $\mathcal{P}$ on the query contains a member at which a loop is detected.*

(B) *The execution of $\mathcal{P}$ on the query enters into a cyclic loop.*

(C) *The depth-first search path of $\mathcal{P}$ on the query contains a primitive cyclic member.*

P r o o f . The equivalence of conditions (B) and (C) has been already established in Corollary 2, and the implication from condition (A) to condition (C) is contained in Lemma 12. For completing the proof, we shall establish now the implication from condition (B) to condition (A). For that purpose, we assume now that condition (B) is satisfied for some given query. Then the depth-first search path of $\mathcal{P}$ on the query will contain some member which is cyclic with a positive period r. By Lemma 5, all further members of the path will be also cyclic with period r. The assumed properties of the sequence $\tau_0, \tau_1, \tau_2, \tau_3, \ldots$ make possible to find a subscript n such that the member $\sigma^{\tau_n}(Q)$ of the path is cyclic with period r, and the inequality $\tau_{n+1} - \tau_n \geq 2r - 1$ holds. We shall prove that a loop will be detected at some member of the execution-detection path of $\mathcal{P}$ on the query, and, more precisely, at some member with first component not greater than $\tau_n + 2r - 1$. By Lemma 10, either there is a member of the execution-detection path with first component $\tau_n + 1$, or there is a closing member of this path with first component not greater than τ_n. At a closing member of the path, surely a loop will be detected, since otherwise the last component of this member would be a terminal member of the depth-first search path, and such terminal member cannot exist in the considered situation. Therefore the second case of the above alternative leads immediately to the needed conclusion, and it remains to study the first case. Then there will be a non-closing member with first component τ_n in the execution-detection path, and this member will invoke saving. Let G be the goal $\sigma^{\tau_n}(Q)$, and v be the minimal one among the lengths of the goals $\sigma^i(G)$, $i = 0, 1, \ldots, r - 1$. It is easy to see that a finite sequence of natural numbers $i_0 < i_1 < \ldots < i_m$ can be found with the following properties: $i_0 = 0$, $i_m \leq r - 1$, $|\sigma^{i_m}(G)| = v$, and, for any natural number j less than m, i_{j+1} is the least one among the natural numbers i greater that i_j and satisfying the inequality $|\sigma^i(G)| < |\sigma^{i_j}(G)|$ (in case $|G| = v$, we have $m = 0$ and the last of these properties becomes trivial). By Lemma 9, there is no natural number i less than i_m such that the goal $\sigma^i(G)$ is primitive cyclic. This, together with Lemma 12, shows that it is not possible a loop to be detected at some execution-detection state with first component $\tau_n + i$, where $0 \leq i < i_m$. We shall show now that, for $j = 0, 1, \ldots, m$, a member with first component $\tau_n + i_j$ and last component $\sigma^{i_j}(G)$ exists in the execution-detection path, and this member invokes saving. We shall proceed by induction. For $j = 0$, the statement is true. Suppose now this statement is true for a certain natural number j less than m. Let the length and the first formula of $\sigma^{i_j}(G)$ be l_j and A_j, respectively. Then we can prove, again by induction, that $<\tau_n + i_j + h, l_j, A_j, \sigma^{i_j+h}(G)>$ is a member of the execution-detection path for $h = 1, \ldots, i_{j+1} - i_j$. Applying this for $h = i_{j+1} - i_j$, we conclude that $<\tau_n + i_{j+1}, l_j, A_j, \sigma^{i_{j+1}}(G)>$ is a member of the execution-detection path. The inequality $|\sigma^{i_{j+1}}(G)| < l_j$ shows that this member invokes saving. This completes the inductive step from j to $j + 1$. Now let us apply the proved statement for $j = m$. The conclusion is that a member with first component $\tau_n + i_m$ and last component $\sigma^{i_m}(G)$ exists in the execution-detection path, and this member invokes saving. Let G' be the goal $\sigma^{i_m}(G)$. By Lemma 8, G' is primitive cyclic with period r. Then $\sigma^r(G')$ will have essentially the same leftmost formula as G'. Let r_0 be the minimal one among the positive integers i such that $\sigma^i(G')$ has essentially the same leftmost formula as G'. Clearly, $r_0 \leq r$, hence $\tau_n + i_m + r_0 \leq \tau_n + 2r - 1 \leq \tau_{n+1}$. Making use of this inequality, of Lemma 9 and of minimality of r_0, we see that no member of the execution-detection path with first component strictly between $\tau_n + i_m$ and $\tau_n + i_m + r_0$ invokes saving, and at no such member a loop is detected. Thus we can apply Lemma 11 and conclude that the execution-detection path contains the member $<\tau_n + i_m + r_0, l', A', \sigma^{r_0}(G')>$, where l' and A' are the length and the leftmost formula of G', respectively. Since $\sigma^{r_0}(G')$ has essentially the same leftmost formula as G', it is sufficient to apply Lemma 9 once more for showing that a loop is detected at the above member of the execution-detection path.

4 Implementation

There are two ways of implementing the described algorithm:

- deep in the Prolog interpreter itself;
- as a interpreter in Prolog, executing Prolog program.

The first method is certainly much efficient, but requires more efforts and knowledge how Prolog is implemented (which differs from one implementation to another, and is usually hidden from end user), thus the the second method (used in our implementation) offers needed generality, but at the cost of execution speed.

The problem of global variables is solved by introducing operator `:=`, which *reads* or *writes* a value from / to *global variable* by using `asserta` and `retract`[4], and operator `<=`, which *pushes / pops* value in a *global stack*, realized in the similar manner (cf. Figure 2).

```
?-op( 10, xfx, <= ).
?-op( 10, xfx, := ).

X <= Y :- nonvar(X), !, P =.. [X, Y], asserta(P), !.
X <= Y :-  P =.. [Y, X], retract(P), !.

X := Y :- nonvar(X), !, P1 =.. [X, _], P2 =.. [X, Y], (retract(P1);true), asserta(P2), !.
X := Y :- P =.. [Y, X], P, !.
```

Figure 2: Global variables

To intercept the Prolog after resolution step, we are modifying the inspected program. All clauses which are not simple facts are modified, so that after each *head unification* our predicate `.inter`[5] is called, as shown in Figure 3.

[4]The Prolog implementations which have worked out problem of global variables can get advantage of it.

[5]The names of our predicates should not be the same as the names of inspected program predicates, this can be managed by using *strange* names for our predicates.

The predicate `.modify` is modifying each clause as described. For the cause of efficiency, `.inter` is given two arguments, the new list of formulas (as a result of SLD resolution), and the length of this list (found by predicate `.goal_len`), to avoid determining the length every time. The result of modification of example program is shown in Figure 4.

```
'.goal_len'( (F, R), L ) :- !, '.goal_len'( R, L1 ), L is L1 + 1, !.
'.goal_len'( B   , 1 ).

'.modify'( (H :- B), (H :- '.inter'(B, L)) ) :- !, '.goal_len'(B, L), !.
'.modify'( C   , C ).
```

Figure 3: Modifying clauses of inspected Prolog program

Original program:

```
   p(X) :- q(X,f(Y)).
   q(X,X).
   r(f(Z)) :- p(Z), r(Z), q(Z, f(Z)).
   s(f(f(Z))) :- p(Z), r(Z), s(Z).
```

Modified program:

```
   p(X) :- '.inter'((q(X,f(Y))),1).
   q(X,X).
   r(f(Z)) :- '.inter'((p(Z), r(Z), q(Z, f(Z))), 3).
   s(f(f(Z))) :- '.inter'((p(Z), r(Z), s(Z)), 3).
```

Figure 4: Example of modified clauses

Our predicate `.inter` is really a small interpreter consisting of two clauses (as shown in Figure 5) which distinguish two situation:

1. the number of new formulas to be executed is greater then one;
2. there is only one formula to be executed.

The difference is obvious, in second case there is no need to update the length of remaining formulas `.RestLen` (the number of all formulas remained to be solved so far) and to proceed with `Rest` if `First` succeeds. To prevent *backtracking*, predicate `.back` is used to stop execution when `First` failed. When all formulas are solved, we are simply going back to previous invocation of `.inter` solving the remained formulas (`Rest`) at this level.

```
'.inter'( (First, Rest), Len_c ) :- !,
    Len_o := '.RestLen', Len_n is Len_o + Len_c - 1, '.RestLen' := Len_n,
    '.loop_chk'( First ),
    ( First ; '.back' ),
    '.RestLen' := Len_o, Len_r is Len_c - 1,
    '.inter'(Rest, Len_r).

'.inter'( First, 1 ) :-
    '.loop_chk'( First ),
    ( First ; '.back' ).
```

Figure 5: Small interpreter of Prolog in Prolog

The `.loop_chk` predicate is a direct implementation of the described method (Figure 6). The predicate `.new_clock` is used for incrementing `.Clock` variable, the `.new_tau` is used to generate new τ as described in the footnote in Section 2.

```
'.loop_chk'( Formula ) :-
     Time      := '.Clock',
     Tau       := '.Tau',
     Len_s     := '.Len',
     Formula_s := '.Formula',
     Len_r     := '.RestLen',
     Len is 1 + Len_r,
     '.new_clock',
     write(Time), write(', '), write(Len), write(' :- '), write(Formula), nl,
     ( Time == Tau, Saved = 't == Tau', '.new_tau'
     ; Len < Len_s, Saved = '|G| < L '
     ; '.formula_eq'(Formula, Formula_s), write('LOOP'), nl, '.exit'
     ; Saved = 'no' ),
     ( Saved \== 'no',
       '.Len'  := Len,
       '.Formula' := Formula,
       write('+----- '), write(Saved),
       write(' ----- save -----> '), write((Len, Formula)), nl
     ; true ), !.
```

Figure 6: Loop check algorithm

The comparing of atomic formulas, which can be sometimes compound expressions, is done by flattening them to lists (Figure 7). This enables us to overcome Prolog lack of predicate for comparing variables. Problem of renaming variables is solved by parallel temporary bounding of compared variables to uniquely generated symbol by predicate `.gensym`[6]. The result of comparison is stored in the variable `.Equal` and the result of binding the variables is canceled by executing `fail`, and the second clause succeeds if the comparison has succeeded.

[6]Implementations which have built-in this predicate can take advantage of it; the *global variable* `.GenSym` is initialized to 0 at the very beginning of our algorithm.

```
'.gensym'('.e.n.u.m.'(V)) :- V := '.GenSym', Vn is V + 1, '.GenSym' := Vn, !.

'.term_eq'( X, Y ) :- var(X),  var(Y),  !, '.gensym'(X), Y = X, !.
'.term_eq'( X, Y ) :- nonvar(X), var(Y),  !, fail.
'.term_eq'( X, Y ) :- nonvar(Y), var(X),  !, fail.
'.term_eq'( X, Y ) :- atomic(X), atomic(Y), !, X = Y, !.
'.term_eq'( '.e.n.u.m.'(X), '.e.n.u.m.'(X) ) :- !.
'.term_eq'( [A|B], [C|D] ) :- !, '.term_eq'(A, C), !, '.term_eq'(B, D), !.
'.term_eq'( X, Y ) :- X =.. A, Y =.. B, !, '.term_eq'(A, B), !.

'.formula_eq'( X, Y ) :- '.term_eq'(X, Y), '.Equal' <= 1, fail.
'.formula_eq'( _, _ ) :- _ <= '.Equal'.
```

Figure 7: Formula comparison

The implementation is done in `C-Prolog` on Sun / Unix. The whole program is about 120 lines long. It includes a read / modify / assert part, which does reading, modifying and asserting clauses of the inspected program. Also a part for initializing, removing, reading and writing of global variables. The algorithm itself (predicates `.inter` and `.loop_chk`) took about 30 lines.

5 Acknowledgments

We are grateful to Prof. Dr. H. Kleine Büning and to Dr. Th. Lettmann for a useful discussion concerning the implementation problem. Thanks are due also to all participants in the seminar of the Institute of Practical Computer Science at the Paderborn University for their attention to our work.

References

[1] D. Skordev, An extremal problem concerning the detection of cyclic loops, *C. R. Acad. Bulgare Sci.*, **40**, No. 10, 1987, 5-8.

[2] D. Skordev, On Van Gelder's loop detection algorithm, *J. Logic Programming*, **14**, 1992, 181-183.

[3] D. Skordev, On the detection of some periodic loops during the execution of Prolog programs, *Banach Center Publications*, Warsaw (to appear).

[4] A. Van Gelder, Efficient loop detection in Prolog using the tortoise-and-hare technique, *J. Logic Programming*, **4**, 1987, 23-31.

[5] A. Van Gelder, Van Gelder's response, *J. Logic Programming*, **14**, 1992, 185.