A NOTE ON COMPUTATIONAL COMPLEXITY OF LOGIC PROGRAMS

(Preliminary Draft)

Andrzej Lingas
Software Systems Research Center
Linköping University
S-581 83 Linköping, Sweden

Abstract

Shapiro defined three complexity measures over logic programs - goal-size, length and depth - and showed their relation to complexity measures for alternating Turing machines. We introduce the fourth complexity measure - conjunctive goal-size - and employing the known ideas of Turing-machine complexity theory we analyze the relation among the complexity measures over logic programs. In particular, for any deterministic logic program of conjunctive goal-size $S(n)$ and length $L(n)$ we can construct an equivalent deterministic logic program of depth $O(log(L(n))$ and length $O(L(n))$, and if the program is strongly deterministic then we can construct another equivalent strongly deterministic logic program of goal-size $O(log(S(n)) + log(L(n)))$ and length $O(S(n)L(n))$.

Introduction

The idea of procedural interpretation to Horn-clause logic begun a new era in logic programming. Today, the programming language Prolog, based on this idea, is a viewed as a start point to the basic programming language of the fifth generation computer systems [FGCS81].

The standard method of executing a program in Prolog is by so called backtracking, consuming a large amount of time and space. In order to achieve the planned speed up in time performance, Japaneses have to improve backtracking by mixing with other methods, for instance bottom-up, and work out an efficient parallel implementation of Prolog. A solid analysis of the computational complexity of logic programs should precede the speed up efforts.

In a large part our goal is to use the similarity between logic programs and alternating Turing machines in order to derive relationships among various complexity measures over logic programs. Efficient implementing of logic programs in various computational models may benefit from these results. As these results rephrase in part known facts from Turing machine theory in the language of logic programming, they seem to be of smaller importance for abstract complexity theory. The other our goal is to comment informally on the possibility of a fast parallel implementation of logic programs, and, on complexity of bottom-up computations of logic programs that are neglected in the logic programming society.

Basic Notions

We totally adopt Shapiro's definitions of *definite clauses, goals, conjunctive goals,* clause's *head* and *body, logic program, goal reduction, substitution, unifier, derivation* and *refutation* of a goal from a logic program, the phrase "a program P *solves* a goal", *refutation tree, length, depth, goal-size* of refutation (see [Sh82a]).

The author came to the conclusion that it is natural and convenient to allow also variable free *initial axioms* as input data.
Initial axioms are inserted in the list of axioms of a logic program before starting its computation.
A pair (G, A) consisting of an initial goal G (possible a conjunctive goal) and a set of initial axioms is called an initial *goal-axiom pair.*
A goal-axiom pair (G, A) has a refutation from a logic program P if G has a refutation from $P \cup A$ in the Shapiro's sense.
The interpretation of a logic program P, $I(P)$, is the set of all variable-free goal-axiom pairs that are constructable from predicates, constants and functors appearing in the language in which P is written, and have a refutation from P.
Following our modification of logic program semantics in comparison with Shapiro, we redefine complexity measures over logic programs as follows:

A logic program P is respectively of *goal-size, depth, length complexity* $C(n)$ if for any goal-axiom pair in $I(P)$ of size n there respectively exists a refutation of goal-size, depth, length $C(n)$.

For the definitions of *non-deterministic* and *deterministic Turing machine* the reader is referred to [CKSh82].

Moreover we use the following definitions:

(1) An *axiom* is a clause with the empty body.
(2) Given a computation of a logic program, C, a *reduction step* of C is the reduction of a chosen goal to a sequence of new goals by a single application of a clause in the program.
(3) Let P, R be a logic program and a refutation, respectively.
The *conjunctive goal-size* of R is the maximum size of the current list of goals at any reduction step of R (respectively, goal size is the maximum size of any unit goal at any step of R). P is of *conjunctive goal-size complexity* $U(n)$ if for any goal-axiom pair G in $I(P)$ of size n there is a refutation of G from P of conjunctive goal-size $\leq U(n)$.
The non-deterministic length of R is the number of nodes in the refutation tree

such that there are at least two clauses whose heads match the goal chosen to reduce. P is of non-deterministic length complexity $N(n)$ if for any goal-axiom pair G in $I(P)$ of size n there is a refutation of G from P of non-deterministic length $\leq N(n)$.
If $N(n) \equiv 0$ then P is *strongly deterministic*.
If every goal-axiom pair in $I(P)$ admits only one refutation then P is *deterministic*, see [H81].
Obviously, if P is strongly deterministic then it is deterministic. The notion of strong determinism for logic programs corresponds to that of determinism for Turing machines.
(4) We assume a standard list representation. The term [] denotes the empty list, and the term $[X|Y]$ stands for a list whose head is X and tail is Y. A string $\alpha_1\alpha_2...\alpha_n$ is represented by the list $[\alpha_1|[\alpha_2|[...|\alpha_n]]]$. With the exception of Theorem 3 integers n are represented as n-fold composition of the functor s applied to the constant 0. Writing a logic program, we skip the clauses and axioms defining the arithmetic predicates of $=, <, \leq, >, \geq$. Finally, we assume that we can test equality between an atom and term by applying a standard equality and inequality predicates built in the formalism of logic programs.
(5) According to the assumed string representation (see 4), Turing machine M is equivalent to a logic program P if after erasing the square brackets and the symbol "|" in the words of $L(M)$, we obtain $I(P)$.

Relationships among Complexity Measures over Logic Programs

In the following remark, we can find a couple of obvious observations on complexity measures over logic programs.

Remark 1. Let P be a logic program of depth complexity $D(n)$, conjunctive goal-size complexity $G(n)$ and length complexity $L(n)$. The following inequalities hold:

$D(n) \leq L(n)$
$L(n) \leq d^{G(n)}$ where d is a constant uniform in P.
Moreover, if we restrict initial goals to single goals then, we have $L(n) \leq c^{D(n)+1} - 1$ where c is the maximum number of goals in a clause of P.

In several computational models, the depth complexity is a natural lower bound on the time taken by parallel evaluation. In the computational model of logic programming it is hard to approximate the lower bound with efficient parallel computations. Simply, solving a conjunctive goal with shared variables cannot be spawned directly.

By virtue of the following theorem, for any logic program there exists an equivalent logic program of fairly small depth. The proof is by applying Savitch's trick, originally applied to simulate non-deterministic space bounded Turing machines by deterministic ones (see [Sa70]), and then, by time bounded alternating Turing machines [CKS80].

Theorem 1. Any logic program P of length complexity $L(n)$ and non-deterministic length complexity $N(n)$ can be transformed into a logic program Q such that a goal-axiom pair $((G_1, G_2, ..., G_l), (A_1, A_2, ..., A_k))$ of size n is in $I(P)$ if and only if there exists a refutation of the corresponding goal-axiom pair $(p([G_1|[G_2|[...|G_l]]], [],$ $\lceil log(L(n)) \rceil$, $(p([A_1|X], [X], 0), p([A_2|X], [X], 0), ..., p([A_k|X], [X], 0)))$ from Q of depth $\lceil log(L(n)) \rceil$, length $4L(n)$ and non-deterministic length $N(n)$. If the program P is deterministic (respectively, strongly deterministic) then Q is also deterministic (respectively, strongly deterministic).
Proof. To form the clauses of Q, we use only the predicate $p(X, Y, i)$. It reads:

If X and Y are lists representing goals and i is a natural number then the goals from X can be reduced to those from Y in 2^i reduction steps.

For each clause $A \leftarrow B_1, ..., B_k$. of P, the program Q contains the axiom $p([A|X], [B_1|[B_2|...[B_k|X]]], 0)$. Note that the predicates from P become functors here. Next, Q contains the axiom $p([], [], 0)$. saying that we can reduce the empty list of goals to itself in one reduction step. The only clause with non-empty body in Q is as follows:

$$p(X, Y, s(j)) \leftarrow p(X, Z, j), p(Z, Y, j).$$

Given a refutation of G from P, of length $L(n)$, there exists a refutation of G from P, say R, such that at each reduction step in R the first clause on the current list of goals is chosen to reduce and R is of length $L(n)$. Having R, we form a refutation of the corresponding initial goal from Q by applying the only clause of Q in depth-first manner, and then, the axioms of Q. As a result, we obtain a refutation whose tree has leaves labelled by instantiated predicate $p(X, Y, 0)$ corresponding to single reduction steps of R. The length of the refutation does not exceed $2^{\lceil log(L(n))+1 \rceil} - 1$. Its non-deterministic length is the same as that of R. If R is the only refutation of the initial goal-axiom pair from P then it is the only refutation of the corresponding initial goal-axiom pair from Q.
Conversely, given a refutation of the corresponding goal from Q, we can easily find out a refutation of G from P. ∎

In Savitch's simulation of non-deterministic space bounded Turing machines with deterministic space bounded Turing machines, the intermediate tape configuration (corresponding to the intermediate list of goals Z in the above proof) is determined by exhaustive search (see [Sa70]). In the proof of Theorem 1, the intermediate list of goals substituted for Z is the outcome of calling $p(X, Z, j)$ (in Concurrent Prolog [Sh82b], the basic clause in Q would be rather written as $p(X, Y, s(j)) \leftarrow p(X, Z, j), p(Z?, Y, j)$). From the point of deterministic simulation, our method of finding the intermediate state is more efficient than Savitch's one if the non-deterministic length complexity of P is small, and worse otherwise.
The first who showed how to simulate Turing machines with logic programs was Tarlund [T67]. Shapiro proved a close relationship between complexity of alternating Turing machines and complexity of logic programs [Sh82a]. The following theorem reveals relationships between complexity of non-deterministic Turing machines and complexity of logic programs (In this theorem, as well as in Theorem 3 and Corollary 1 and 2 we informally use the notion of simulation whose meaning can be deduced from the proof of Theorem 3).

Theorem 2. Any multi tape (deterministic) Turing machine operating in time $T(n)$, and space $S(n)$ can be simulated by a (strongly deterministic, respectively) logic program of length complexity $O(T(n))$, and conjunctive goal-size complexity $O(S(n))$. Conversely, any (strongly deterministic) logic program of length complexity $L(n)$, and conjunctive goal-size complexity $S(n)$ can be transformed into an equivalent (deterministic, respectively) Turing machine operating in time $O(L(n) \times S(n)^2)$, and space $O(S(n))$.
Hint. Note that a single reduction step can be simulated by a deterministic Turing machine in time $O(S(n)^2)$ (see [R65]) and read the proof of Theorem 4.4 and 5.4 in [Sh82a]. ▮

By Theorem 1 and 2 we obtain the following corollary:

Corollary 1. Any (deterministic) Turing machine operating in time $T(n)$, and space $S(n)$ can be simulated by a (strongly deterministic, respectively) logic program of depth complexity $O(log(T(n))$, length complexity $O(T(n))$, and conjunctive goal-size complexity $O(S(n))$.

Probably, several important problems solvable by deterministic Turing machines in polynomial time are not solvable in parallel time $O(log^k n)$, i.e. by parallel machines with polynomial number of processors with fixed fan-in and fan-out, running in time $O(log^k n)$ (see [B77],[CKS81]). As by Corollary 1, 46c

deterministic Turing machines operating in polynomial time can be simulated by deterministic logic programs of logarithmic depth complexity, probably a small depth complexity of a logic program does not ensure the existence of a fast parallel implementation of the program, in the general case. It seems that the requirements that a logic program should satisfy to admit an essential parallel speed up are more complex. In the next section, we shall briefly discuss this problem from the point of view of bottom-up computations. Here, we informally propose the following requirements, coherent with the top-down nature of derivations from logic programs.

Let P be a logic program of length complexity $L(n)$. For i, j, let $R_{i,j}(n)$ be the equivalence relation between conjunctive goals such that $G_1 R_{i,j}(n) G_2$ if and only if for any goal-axiom pair of size n, G, any refutation of G from P with the the i —th element G_1 performs the same i-th through j-th reduction steps as any refutation of G from P with the i-th element G_2. In other words, to determine the i-th through j-th reduction steps of a refutation of G from P whose i-th element is G_1 it is sufficient to know a representative of the equivalence class of $R_{i,j}(n)$ for G_1. Suppose that for $n \in N$ there exists a tree T_n of fixed degree with leaves consecutively labeled by 1 through $L(n)$, and a number m_n such that for any subtree of T_n with the leftmost leaf labelled by i and the rightmost leaf labelled by j, the number of equivalence classes of $R_{i.j}(n)$ is at most m_n. In the simplest case, the tree T_n may correspond to the refutation tree of P. Given an goal-axiom pair of size n, G, we can recursively find a refutation of G from P (if it exists) by applying divide and conquer strategy induced by T_n and trying all representatives of the equivalence classes of $R_{i,j}(n)$ in parallel. Provided that T_n and the representatives are given, the refutation can be determined in time $O(log(m_n) \times height(T_n))$ with the use of $O(2^{log(m_n) \times height(T_n)})$ processors. In particular, if m_n is a constant uniform in n and $height(T_n) = O(logn)$, P can be implemented in parallel time $O(logn)$. The reader can find more details about this approach, expressed rather in terms of Turing machines, in [L83]. Here, we offer only the following simple example.

Example 1

Let us consider the following logic program, $delmem(Z, X, Z', H)$, where Z is an input linear list of a constant length over a finite alphabet Σ, X is an input list over Σ organized as a complete binary tree of height H, Z' is the output list composed of all the elements of Z that are not in X, $member(A, Z)$, $notmember(A, Z)$, $delete(A, Z, Z')$ stand for the standard predicates testing membership of A in Z and deleting A from Z (i.e. $Z' = Z - A$) respectively (see

[CM81]), we may assume without loss of generality these standard predicates to be available primitives since they are applied to sublists of the input list Z which is of fixed length in this example.

$$delmem(Z,X,Z',H) \leftarrow delmem(Z,X,Z',0,H).$$

$$delmem(Z,[X \mid Y],Z',K,H) \leftarrow$$

$$K < H, delmem(Z,X,Z'',s(K),H), delmem(Z'',Y,Z',s(K),H).$$

$$delmem(Z,A,Z',H,H) \leftarrow member(A,Z), del(A,Z,Z').$$

$$delmem(Z,A,Z,H,H) \leftarrow notmember(A,Z).$$

Note that if we neglect labels, the form of a refutation tree of P for any goal-axiom pair with the input list X of length n is totally determined by n. Let T_n be a tree of such a form, with leaves consecutively labelled by 1 through n. Clearly, if i and j are the labels of the rightmost and the leftmost leaf in a subtree of T_n, then the i-th through j-th reduction steps in any refutation of an goal-axiom pair with the input list X of length n from our logic program is a refutation of the goal $delmem(U,Y,U',k,h)$ corresponding to the root of the subtree. Thus, the i-th through j-th steps are totally determined by the instantiation of Z, U. Therefore, the equivalence classes of the relation $R_{i,j}(n)$ can be identified with the possible instantiations of Z. As the input list Z is of a fixed length, the number of possible instantiations of Z is a constant uniform in n. Hence, the number m_n is a constant uniform in n here. It is not difficult to see that the language specified by $delmem(Z,X,Z',K,H)$ is regular but in the general case, the language specified by a logic program for which $m_n = O(1)$ is not necessarily regular [L83]. The tree T_n induces the same divide and conquer strategy as the recursive definition of $delmem(Z,Y,Z',k,h)$, therefore, we do not need to transform P in this respect. To try all of the representatives of equivalence classes of the relations $R_{i,j}(n)$, equivalently all possible instantiations of U, it is sufficient to add the following clauses with B ranging over all possible instantiations of Z:

$$delmen(Z,[X \mid Y],Z',K,H) \leftarrow$$

$$K < H, delmem(Z,X,B,s(K),H), delmem(B,Y,Z',s(K),H).$$

It is easy to see that by fully using the OR-parallelism introduced by the above, additional clauses, $delmem(Z,Y,Z',k,h)$ can be implemented in parallel time $O(logn)$.▮

The following theorem relates time and space complexity of Turing machines to goal-size complexity of logic programs. The proof is analogous to the proof of Chandra *et al.* showing $\bigcup_{c>0} DTIME(c^{S(n)}) \subseteq ASPACE(S(n))$ [CKSh82].

Theorem 3. Let M be a deterministic Turing machine operating in time $T(n)$ and space $S(n)$. M can be simulated by a strongly deterministic logic program Q of goal-size complexity $O(logT(n) + logS(n))$.
Outline of Proof. We assume several restrictions on M following the proof of Theorem 3.4 in [CKS80]. In particular, M has only one tape, on the tape the input word is written, M accepts an input by entering its unique accepting state q_A with the head scanning the $T(n) + 1st$ tape square, etc. (see [CKS80] for details). A computation of M on the input word is described as a sequence of configurations, each in the form $\alpha q \beta$ where $\alpha\beta$ describes the contents of squares 0 through $4T(n)$, q is the current state of M and the head of M points the rightmost symbol of α.

For any four symbols from the tape alphabet and the set of states of M, $\delta_{-1}, \delta_0, \delta_1, \delta_2$, among which at most one represents state of M, there is unique symbol δ such that for any j if $\delta_{-1}, \delta_0, \delta_1, \delta_2$ occupy positions $j-1, j, j+1, j+2$ in a configuration of M, then δ occupies the position j in the next configuration of M. For each such quintuple $\delta_{-1}, \delta_0, \delta_1, \delta_2$, δ, the program Q contains the axiom $next(\delta_{-1}, \delta_0, \delta_1, \delta_2, \delta)$.. The basic predicate in Q is $accept(j, t, a)$. It says that in the $t - th$ configuration of the computation of M, the $j - th$ square contains the symbol a.
To prove the theorem we cannot represent the integers j, t using the unary notation defined in the previous section. Here the integer of binary representation $b_1, ..., b_l$ is written as $[b_l|[...|b_1]]$. The successor predicate is defined as follows:

$$suc([1|X],[0|Y]) \leftarrow suc(X,Y).$$

$$suc([0|X],[1|X]).$$

$$suc([],1).$$

The definitions of the predicates of $<$ and $\leq$ for this specific representation of integers are left to the reader.
The main clause in Q is as follows:

$$accept(j,u,X) \leftarrow suc(t,u),\ suc(i,j),\ accept(i,t,Y),$$

$$accept(j,t,Z),$$

$$suc(j,k),\ accept(k,t,W),$$

$$suc(k,l),\ accept(l,t,T),$$

$$next(Y,Z,W,T).$$

To verify the initial contents of the tape we use the clauses $accept(j,0,X) \leftarrow input(j,X)$. The integers occurring in any derivation of $accept(T(n)+1, T(n), q_A)$ from P have binary representation of the length not exceeding $\lceil min\{log(T(n)), log(S(n))\}\rceil$. To prove the theorem, we show by induction that $accept(j,t,a)$ can be proved in $O(L(n))$ reduction steps if and only if the given interpretation of $accept(j,t,a)$ is right. ▮

By Theorem 2 and 3 we obtain the following corollary:

Corollary 2. Any strongly deterministic logic program of length complexity $L(n)$ and conjunctive goal-size complexity $S(n)$ can be simulated by a strongly deterministic logic program of length complexity $O(L(n) \times S(n)^2)$, and goal-size complexity $O(log(L(n)) + log(S(n)))$.

Bottom — up Computations of Logic Programs and their Complexity

That what we mean by a computation of a logic program P might be specified as a *top-down computation* of P. A *bottom-up computation* of P is a reversed (top-down) computation of P, and can be briefly described as follows.

The computation starts from a set of instantiated axioms. At each step we non-deterministically pick a clause of P, $A \leftarrow B_1', B_2', ..., B_k'$ (it might be an axiom, i.e. $k = 0$). Then we non-deterministically choose a sequence $B_1, B_2, ..., B_k$ from the list of current axioms in order to unify it with the body of the previously chosen clause. The unification is via a substitution θ and the axiom $A\theta$. is added to the current list of axioms. The computation terminates when there exists a substitution θ unifying each initial goal with a member of the current list of axioms. The definitions of of *derivation, refutation* of an initial goal-axiom pair from P etc. as well as the definitions of *depth, length* and *conjunctive goal-size* complexity for bottom-up computations are similar to those for top-down computations, and are left to the reader.

Remark 2. If a logic program is of bottom-up depth complexity $D(n)$, bottom-up length complexity $L(n)$, bottom-up goal-size complexity $U(n)$, then it is of depth complexity $D(n)$, length complexity $L(n)$ and goal-size complexity $U(n)$.

Choosing a clause at a step of a bottom-up computation of P and a sequence of some current axioms in order to unify with the body of the clause, we do not know whether it leads to a proof of the initial goals. Moreover, the number of possible choices of the sequence of some current axioms may be of order n^k where k is the number of goals in the body of the chosen clause. That is why programs in Prolog are executed in a top-down manner. We may argue that if the program P is non-deterministic then choosing a clause in a top-down computation in order to unify its head with the selected goal, we neither know whether it will solve the goal. However, if we do not loop then we may backtrack in case of failure like the running Prolog interpreters whereas the definition of failure for a bottom-up computation is not clear. Neverthless, it is author's feeling that for an important class of logic programs bottom-up computations are essentially more efficient than (top-down) computations. This class may include so called *dynamic programming* procedures which recursively generate a lot of symmetric subgoals in order to solve the original goal.

An example of a logic program for the dynamic programming procedure of Cocke, Kasami and Young, accepting words from the language $L(G)$ where $G = (N, \Sigma, P, S)$ is a context-free grammar in Chomsky normal form (see [AHU74]), is shown as Program 1.

Program 1

$$\begin{array}{lll}
p_A(i,j) & \leftarrow & q_A(i,i,j).\ \text{for } A \in N, \\
q_A(i,k,j) & \leftarrow & s(s(k)) < j, q_A(i,s(k),j).\ \text{for } A \in N, \\
q_A(i,k,j) & \leftarrow & i < k < j, p_B(i,k), p_C(k,j). \quad \text{for} \quad A \to BC \in P, \\
p_A(i,s(i)) & \leftarrow & i < n, input(a,i). \quad \text{for} \quad A \to a \in P.
\end{array}$$

The program is design to succeed on the goal-axiom pair consisting of the goal $p_S(0,n)$ and the axioms $input(w_i, i)$. where $w_1, \ldots w_n$ is the input word if and only if the input word belongs to $L(G)$. In the worst case we may have to backtrack an exponential in n number of times in order to find a (top-down) computation accepting w whereas a bottom-up computation yields an answer in $O(n^3)$ deduction steps if it proves a new goal at each deduction step. Why are bottom-up computations successful here ? Simply, there are only $O(n^3)$ variable free goals that can be solved by Program 1 starting from the initial axioms.

Definition 1. Let R be a refutation. The *goal number* of R is the total number of distinct goals in the nodes of the refutation tree. A logic program P is of *goal number complexity* $G(n)$ if for any goal A in $I(P)$ of size n there exists a refutation of A from P of goal number $\leq G(n)$.

Our observation about bottom-up computations of Program 1 can be generalized as follows:

Remark 3. Let P be a logic program. P is of goal-number complexity $G(n)$ if and only if it is of bottom-up length complexity $G(n)$. Moreover, if P is of goal-size complexity $U(n)$ then it is of goal number complexity $d^{U(n)}$ where d is a constant uniform in P.

The analogous remark for (top-down) computations would not be true. Simply, it might happen that to solve a given goal, we have to solve the same goal several times. In implementing (top-down) computations we can get rid of the above inefficiency by dynamically extending the original set of axioms of P by the solved intermediate goals.

By Theorem 2 and Remark 2 and 3, we can observe that any Turing machine operating in polynomial time can be simulated by a logic program of polynomial goal number complexity.

In a simple parallel implementation of bottom-up computations, we do not encounter the problem of variable sharing for subgoals of equal rank. Therefore, the depth complexity and the time taken by a single deduction step seem to decide about the time performance of a bottom-up computation of a logic program of polynomial goal number complexity, in a parallel computational model. The recent paper of Lewis and Statman [LS8?] has shown the problem of unification between first order terms to be complete in co-NLog Space. Therefore, the existence of a parallel algorithm for the unification problem operating in time $O(logn^k)$ and using a polynomial in n number of processors would imply the existence of such algorithms for any problem from NLog Space or co-NLog Space, which seems unlikely. Hence, we cannot count on a parallel implementation of the single deduction or reduction step in time $O(logn^k)$. If the logic program P in Theorem 1 is not patological then the bottom-up depth complexity of the resulting program Q is equal to the depth complexity of Q. Therefore, by Remark 2, we can usually apply Theorem 1 in order to compress the bottom-up depth complexity. The following theorem, analogous to Theorem 1, shows how to achieve this for any logic program.

Theorem 4. Let P be a logic program of bottom-up length complexity $L(n)$. Let $B_1, ..., B_m$ be the list of all axioms in P. P can be transformed into a logic program Q such that a goal-axiom pair $((G_1, ..., G_k), (A_1, ..., A_l))$ is in $I(P)$ if and only if the corresponding goal-axiom pair $(p([B_1|[...|B_m]], [G_1|[...[G_k|-]]], \lceil log(L(n))\rceil), (p([X],[A_1|X],0), ..., p([X],[A_l|X],0)))$ has a refutation from Q of depth $\lceil log(L(n))\rceil + \lceil log(n + L(n))\rceil$.

Outline of Proof. The proof is again by applying Savitch's trick, analogously as in the proof of Theorem 1. Here the predicate $p(X, Y, i)$ says:

If X and Y are lists of axioms and i is natural number then the axioms in Y can be derived from those in X in 2^i (bottom-up) deduction steps.

The axioms chosen from the current list of axioms in order to unify with the body of chosen clause may occupy various positions on the list. Therefore, we include in Q the following clauses to pull the chosen axioms to the front of the list (because the list of axioms may be of length $n + L(n)$ we again apply Savitch's trick).

$$p(X,Y,0) \leftarrow q(X,Y,\lceil log(n+L(n))\rceil).$$

$$q(X,Y,s(j)) \leftarrow q(X,Z,j), q(Z,Y,j).$$

$$q([X|[Y|Z]],[Y|[X|Z]],0) \leftarrow X{\neq}Y.$$

$$q(X,X,0).$$

Finally, for each clause $A{\leftarrow}B_1, ..., B_n$ in P we have the corresponding axiom $q([B_1|[...[B_n|X]...],[A|[B_1|[...|[B_n|X]...],0)$. ∎

In the above theorem, the program Q is non-deterministic even if the program P is strongly deterministic (compare with Theorem 1).

Possible Extensions

(1) The goal-size complexity of Q in Theorem 1 might be as large as $L(n) \times U(n)$ if P is of goal-size complexity $U(n)$. It seems possible to generalize Theorem 1 by showing a trade off between the depth complexity and the goal-size complexity of Q.
(2) It is possible to formalize the notion of simulation or introduce a more general concept of equivalence among logic programs and Turing machines.
(3) It would be interesting to design a parallel algorithm for the unification problem operating in time $O(n^\alpha)$ and using (n^β) processors where $\alpha < 1$ and $\beta < 1$.

Acknowledgements

I would like to express my appreciation to Jan Komorowski, and Jan Maluszynski for their encouragment and remarks.

References

[AHU74] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[B77] Borodin, A.B., *On relating time and space to size and depth*, SIAM J. Compt., vol. 6(4), 1977.

[CKS81] Chandra,A.K., D.C. Kozen, L.J. Stockmeyer, *Alternation*, Journal of the ACM 28(1), 1981.

[CM81] Clocksin, W.F. and C.S. Mellish, *Programming in Prolog*, Springer-Verlag, 1981.

[H81] Hogger, C.J., Derivation of Logic Programs, Journal of the ACM 28(1), 1981.

[FGC81] Proceedings of International Conference on Fifth Generation Computer Systems, Tokyo, 1981.

[L83] Lingas, A., *Languages with Sparse Computations*, in preparation.

[LS8?] Lewis, H.R. and R. Statman, *Unifiability is Complete for co-NLog Space*, unpublished manuscript, 198?.

[R65] Robinson, J.A., *A machine oriented logic based on the reduction principle*, Journal of the ACM 12, January, 1965.

[Sa70] Savitch, W.J., *Relationships Between Non-deterministic and Deterministic Tape Complexities*, JCSS 4(2), 1982.

[Sh82a] Shapiro,E.Y., *Alternation and the Computational Complexity of Logic Programs*, International Conference on Logic Programming, Marseil, 1982. [Sh82b] Shapiro,E.Y., *A Subset of Concurrent Prolog and Its Interpreter*, unpublished manuscript, 1982.

[T77] Tarlund, S.A., *Horn Clause Computability*, BIT 17, 1977.