Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 212 (2008) 71–85
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

A Novel Derivation Framework For Definite Logic Program[⋆]

MengJun Li[a,1] ZhouJun Li[b,2] HuoWang Chen[a,3] Ti Zhou[a,4]

[a] *School of Computer Science, National University of Defence Technology, Changsha, China*

[b] *School of Computer Science and Engineering, BeiHang University, Beijing, China*

Abstract

Is a closed atom derivable from a definite logic progam? This derivation problem is undecidable. Focused on this problem there exist two categories approaches: the accurate approach that does not guarantee termination, and the terminated abstract approaches. Both approaches have its advantages and disadvantages. We present a novel derivation framework for the definite logic program. A dynamic approach to characterizing termination of fixpoint is presented, then which is used to approximately predict termination of fixpoint in advance.If the fixpoint is predicted termination, we use the non-terminational approach to the derivation problem, otherwise,the terminated abstract approach is used. With this termination predicting approach, we combine the non-termination accurate approaches and the termination abstract approaches together for solving the derivation problem more efficiently. And the experiment results demonstrates the effectiveness of our approach.

Keywords: abstract and refinement,termination prediction,definite logic program.

1 Introduction

Is a closed atom derivable from a definite logic program? This derivation problem is undecidable. Many research problems in the area of computer science are reduced directly or indirectly to this problem, for instance, the verification of security protocols on secrecy property and authentication property can be reduced directly to this problem.

Abstract interpretation is a systematic methodology to design approximation algorithms for complex and undecidable problems in the area of computer science[1],

⋆ Supported by the National Natural Science Foundation of China under Grant No. 60473057, 90604007, 60703075, 90718017, the National High Technology Research and Development Program of China under Grant No.2007AA010301 and the Research Fund for the Doctoral Program of Higher Education of China under Grant No.20070006055.

1 Email: mengjun_li1975@yahoo.com.cn

2 Email: zhoujun.li@buaa.edu.cn

3 Email: hwchen@nudt.edu.cn

4 Email: tizhou@gmail.com

doi:10.1016/j.entcs.2008.04.054

which has been studied extensively in the logic programming community. Many efficient algorithms have been proposed to approximate the computation of the fixpoint of logic program, such as [5][6][7] [8][9].

There exist two categories approaches: the accurate approach that does not guarantee termination, and the terminated abstract approaches. Both approaches have its advantages and disadvantages. This paper presents a novel derivation framework for definite logic problem. The derivation framework consists of a derivation algorithm, an abstract and refinement approach for the fixpoint, and an algorithm for predicating termination of fixpoint.

(1) The derivation algorithm originated from the verification algorithm on secrecy property of security protocol presented in [2], the algorithm is very efficient by utilizing optimization techniques, but it does not terminate because the fixpoint of definite logic program does not terminate in general.

(2) The presented abstract and refinement approach computes a safe approximation of the fixpoint based on a variant of the $depth(k)$ abstract domain[5], and refine the computed abstract fixpoint by increasing the threshold k of the $depth(k)$ abstract domain[3]. The constructed abstract fixpoint terminates, and using which instead of the fixpoint the derivation algorithm will terminate. If the result of the derivation algorithm using the abstracted fixpoint shows the closed atom is not derivable, then it is actually not derivable since the abstracted fixpoint is a safe approximation; If the result of the derivation algorithm with the abstracted fixpoint shows the closed atom is derivable, and it is derivable also from the rules in abstract fixpoint which is not abstracted, then the derivation witness(a derivation tree) of the closed atom can be constructed using the same approach in [4]. Otherwise, the abstract fixpoint is refined by increasing the threshold k, and the derivation algorithm will use the refined abstract fixpoint to solve the derivation problem again.

(3) The prediction algorithm is utilized to predict termination of fixpoint in advance, if the fixpoint is predicted termination, then the derivation algorithm will use the fixpoint to solve the derivation problem, otherwise the derivation algorithm will use the abstract fixpoint. By the termination prediction algorithm, the non-termination accurate approach and the termination abstract approach are combined together to solve the derivation problem.

Related Work.

The question of how to define safe approximations of definite logic program has been discussed before [5][6][7] [8][9]. Most of this work considered the definition and precision of various different approximations. Compared with their work, our approach has the following characteristics:

(1) Not all definite logic program are abstracted with the variant of the $depth(k)$ abstract domain. In our approach, termination of fixpoint is predicted in advance, and fixpoint is abstracted by the variant of the $depth(k)$ abstract domain only when it is predicted not termination.

(2) Our approach supports the abstract refinement iteration analysis framework. There exists no explicit refinement ways for the above approximation algorithms, whereas the variant of the depth(k) abstract domain is prone to be refined by only increasing the threshold k. And the derivation, constructing derivation witness and refinement all can be implemented in a mechanized way.

The paper is organized as follows: in section 2, the syntax of definite logic program is presented; And in section 3, the derivation algorithm is presented; In section 4, the termination characterization and prediction approach of fixpoint is presented; In section 5, the abstraction and refinement approach of fixpoint is presented; In section 6, we present the experimental results to demonstrate the effectiveness of our approach. And finally we conclude the paper in section 7.

2 Definite Logic Program

The syntax of the definite logic program is given in Table 1.

$m, s, t ::=$	$Terms$	$F, C, A ::=$	$Atom, Fact$
x, y, z	$Variables$	$p(t), q(t)$	$Predicates$
a, b, c	$Constants$	R, R'	$Rules$
$f(M_1, \cdots, M_n)$	$functions$	$F_1 \wedge \cdots \wedge F_n \rightarrow F$	$Logic\ Rules$

Table 1
The Syntax of Definite Logic Program

Definition 2.1 Let $R_1 = H_1^1 \wedge \cdots \wedge H_m^1 \rightarrow C_1$ and $R_2 = H_1^2 \wedge \cdots \wedge H_n^2 \rightarrow C_2$ be two logic rules, if $C_1 = p(t_1)$, $C_2 = p(t_2)$, define rule implication $R_1 \Rightarrow R_2$ if and only if there exists a substitution θ such that: $t_1\theta = t_2$, and for each $H_i^1 (1 \leq i \leq m)$, $H_i^1\theta \in \{H_1^2, \cdots, H_n^2\}$.

Definition 2.2 Let F be a closed atom, and P be a definite logic program, F is derivable from P if and only if there exists a finite tree defined as follows:

(1) Its nodes (except the root node) are labelled by rules $R \in P$, and its edges are labelled by closed atoms.

(2) If the tree contains a node labelled by R with an incoming edge labelled by F_0 and n outgoing edges labelled by $F_1, \cdots, F_n$, then $R \Rightarrow F_1 \wedge \cdots \wedge F_n \rightarrow F_0$.

(3) The root node has only one outgoing edge labelled by F. such a tree is called a derivation tree of F from P. The above derivation tree is also called a derivation witness.

3 Derivation algorithm

Definition 3.1 Let $R = H \rightarrow F$ and $R' = H' \rightarrow F'$ be two logic rules, $F = p(t)$, let $F_0 = p(t')$ be an atom in H' such that t can be unified with t', then the resolution

$R^{'} \bullet R$ between R and $R^{'}$ is $(H \wedge (H^{'} - F_0))\theta \rightarrow F^{'}\theta$, $\theta = mgu(t, t^{'})$ is the most general unifier of t and $t^{'}$.

Definition 3.2 Atoms of the form $p(x)(x$ is an arbitrary variable) in the body of a logic rule are called false goals, atoms of the form $p(t)(t$ is not a variable) are called goals.

Definition 3.3 Let $H \rightarrow C$ be a logic rule, if the atoms in H are all false goals, then we say $H \rightarrow C$ is a solved form logic rule.

Let $SolvedForm$ denote the set of solved form logic rules, and $UnSolvedForm$ denote the complement of $SolvedForm$.

Definition 3.4 Let $R = H \rightarrow F$ and $R^{'} = H^{'} \rightarrow F^{'}$ be two logic rules, $R \in SolvedForm$, $R' \in UnSolvedForm$, $F = p(t)$, let $F_0 = p(t^{'})$ be a goal in $H^{'}$ such that $t^{'}$ can be unified with t, then the X-resolution $R' \circ R$ between R and $R^{'}$ is $(H \wedge (H^{'} - F_0))\theta \rightarrow F^{'}\theta, \theta = mgu(t, t^{'})$.

Let R be a logic rule and B be a logic rule set, define $addRule(R, B)$ as:
if $\exists R^{'} \in B, R^{'} \Rightarrow R$, then $addRule(R, B) = B$
else $addRule(R, B) = \{R\} \cup \{R^{'} | R^{'} \in B, R \not\Rightarrow R^{'}\} \cup \{marked(R^{''}) | R^{''} \in B, R \Rightarrow R^{''}\}$
where $marked(R'')$ denotes that $R^{''}$ will not be used to compute X-resolutions. And define:
$addRule(\{R_1, \cdots, R_m\}, B) = addRule(\{R_2, \cdots, R_m\}, addRule(R_1, B))$.
Let $Marked$ denote the set of logic rules those will not be used to compute X-resolutions, and $UnMarked$ denote the complement of $Marked$. Let $R = F_1 \wedge \cdots \wedge F_n \rightarrow C$ be a logic rule, the unary function $elimdup(R)$ returns a rule $R^{'}$ such that:

(1) In $\{F_1, \cdots, F_n\}$, only those atoms that satisfies the following conditions will occur in the body of $R^{'}$: if $j < i$,then $F_i \neq F_j$;
(2) C is the head of the rule $R^{'}$; Let P be a definite logic program, define:
$Rule^0(P) = \{elimdup(R) | R \in P\}$;
$T^0(P) = Rule^0(P) \cap SolvedForm$; $C^0(P) = Rule^0(P) \cap UnSolvedForm$;
$X\_Resolution^{n+1}(P) = \{elimdup(R) | R = R^{'} \circ R^{''}, R^{'} \in T^n(P), R^{''} \in C^n(P)\}$;
$Rule^{n+1}(P) = addRule(X\_Resolution^{n+1}(P), Rule^n(P))$;
$T^{n+1}(P) = Rule^{n+1}(P) \cap SolvedForm$;
$C^{n+1}(P) = Rule^{n+1}(P) \cap UnSolvedForm$;

Definition 3.5 Let P be a definite logic program, define $fixpoint(P) = \{T^n(P) | n \geq 0\} \cap UnMarked$, $fixpoint(P)$ is called the solved-form fixpoint of P.

Let R be a logic rule and B be a logic rule set, define $derivablerec(R, B, P)$ as:
if $\exists R^{'} \in B, R^{'} \Rightarrow R$, then $derivablerec(R, B, P) = \emptyset$
else if $R = \rightarrow C$, then $derivablerec(R, B, P) = \rightarrow C$
else $derivablerec(R, B, P) = \{derivablerec(elimdup(R^{'} \bullet R), \{R\} \cup B, P) |$

$R^{'} \in fixpoint(P)\}$

And define $derivable(F, P) = derivablerec(F \rightarrow F, \emptyset, P)$.

Theorem 3.6 *If $R \bullet R^{'}$ is defined, $R_1 \Rightarrow R$ and $R_1^{'} \Rightarrow R^{'}$, then either $R_1 \bullet R_1^{'}$ is defined and $R_1 \bullet R_1^{'} \Rightarrow R \bullet R^{'}$, or $R_1^{'} \Rightarrow R \bullet R^{'}$.*

Theorem 3.7 *Let P be a definite logic program and F be a closed atom, then $derivable(F, P)$ terminates.*

Theorem 3.8 *Let P be a definite logic program and F be a closed atom, then F is derivable from P if and only if F is derivable from $fixpoint(P)$.*

Theorem 3.9 *Let P be a definite logic program and F be a closed atom, then F is derivable from $fixpoint(P)$ if and only if $\rightarrow F \in derivable(F, P)$.*

The above four theorems are variants of the corresponding theorems in [2].

4 Termination Characterization and Prediction

Based on the dynamic approach presented in [11], a corresponding dynamic approach to characterize termination of solved-form fixpoint is presented, and which is used to predict termination of solved-form fixpoint in advance.

4.1 Termination Characterization

Definition 4.1 Let P be a definite logic program, and A be the head of a rule R, then $tag(A)$ is defined inductively as follows:
(1)if $R^{'}$ is the i^{th} rule of P and $R = elimdup(R^{'})$, then $tag(A) = i$;
(2)if $R = elimdup(R^{'} \circ R^{''})$ and $A^{''}$ is the head of $R^{''}$, then $tag(A) = tag(A^{''})$.

Observe that since a definite logic program has only a finite number of rules, infinite logic rules in solved-form fixpoint result from repeatedly applying the same set of rules, which leads to infinite repetition of selected variant goals or selected goals with recursive increase in term size[11]. By recursive increase of term size of a goal A from a goal B means that A is B with a few function/name/variable symbols added and possibly with some variables changed to different variables. Termination can be characterized by checking whether there exists infinite repetition of selected variant goals or selected goals with recursive increase in term size.

All the rules in solved-form fixpoint are solved form logic rules, then termination is characterized by checking whether there exists infinite repetition of selected variant goals or selected goals with recursive increase in term size among all solved-form logic rules. Since all atoms occurring in the body of a solved form logic rule are of the simple form $p(x)$, we only need to consider the head of solved form logic rule for termination characterization.

Combining all above ideas together, termination of solved-form fixpoint can be characterized by checking whether there exists at least a logic rule in the definite logic program which is applied repeatedly, and for a logic rule $R = H \rightarrow F$, R is applied repeatedly can be checked by infinite repetition of selected variant goals or selected goals with recursive increase in term size among those heads of logic rules in solved-form fixpoint whose value of tag is equal to $tag(F)$.

Definition 4.2 Let T be an atom and S be a string that consists of all function symbols, names and variables in T, which is obtained by reading these symbols sequentially from left to right. The symbolic string of T, denoted S_T, is the string S with every variable replaced by the new fresh symbol χ.

For instance, let $T = f(x, g(x, f(a, y)))$, then $S_T = f\chi g\chi fa\chi$. The projection relation defined as following precisely characterizes the repetition of selected variant goals or repetition of selected goals with recursive increase in term size.

Definition 4.3 Let S_{T_1} and S_{T_2} be two symbolic strings, S_{T_1} is a projection of S_{T_2}, denoted $S_{T_1} \subseteq_{proj} S_{T_2}$, if S_{T_1} is obtained from S_{T_2} by removing one or more elements.

For example, $a\chi\chi bc \subseteq_{proj} fa\chi a\chi\chi b\chi c$. For each solved form logic rule, its construction process can be described with the resolution chain, which is constructed from X-resolutions between logic rules in $SolvedForm$ and $UnSolvedForm$ and depictured as the reverse binary-tree in Fig1, where $R = elimdup(R_2 \circ R_1)$.

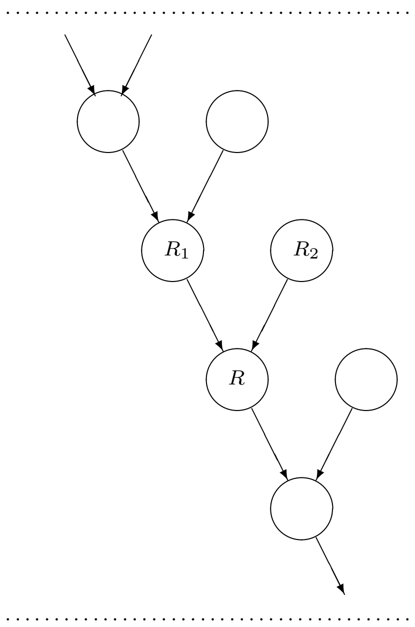

Fig 1: The Resolution Chain

The nodes in reverse binary-trees are labelled by logic rules in $SolvedForm$ or $UnSolvedForm$. The depth of nodes induce the ancestor-descendant relation $\prec_{anc}$ between heads of logic rules labelling these nodes. For example, in the reverse binary-tree in Fig1, the depth of the node labelled by $R = H \to F$ is equal to one adding the depth of the node labelled by $R_1 = H_1 \to F_1$, or the node labelled by $R_2 = H_2 \to F_2$, then the ancestor-descendant relation $\prec_{anc}$ among F, F_1, F_2 is: $F_1 \prec_{anc} F, F_2 \prec_{anc} F$. For convenience, $N_i :: A_i$ is used to denote a node N_i and the atom A_i which is the head of the logic rule labelling N_i.

Definition 4.4 Let $A_1 = p(t_1)$, $A_2 = p(t_2)$ be two atoms, A_1 is said to loop into A_2, denoted by $A_1 \leadsto_{loop} A_2$, if $S_{A_1} \subseteq_{proj} S_{A_2}$ and $tag(A_1) = tag(A_2)$.

Compared with the definition in [11], the loop into relation is defined restrictively over those heads whose values of *tag* is equal.

Definition 4.5 Let T be a reverse binary-tree and N_i, N_j be two nodes in T, if $A_i \prec_{anc} A_j$ and $A_i \leadsto_{loop} A_j$, then A_j is called a loop goal of A_i.

Definition 4.6 Let T be a reverse binary-tree, the sequence constructed inductively with the following rules is called the selection sequence of T:

(1) if the depth of node N_1 is 1 and it is labelled by a logic rule in $SolvedForm$, then $N_1 :: A_1$ is added into the selection sequence;

(2) if all the nodes whose depth is less than l and labelled by a logic rule in $SolvedForm$ are added into the selection sequence, if the depth of node N_l is l and it is labelled by a logic rule in $SolvedForm$, then $N_l :: A_l$ is added into the selection sequence.

Lemma 4.7 *[11] Let $\{A_i\}_{i=1}^{\infty}$ be an infinite sequence of strings over a finite alphabet Σ, then there is an infinite increasing integer sequence $\{n_i\}_{i=1}^{\infty}$ such that for all $i, A_{n_i} \subseteq_{proj} A_{n_{i+1}}$.*

Termination of solved-form fixpoint is characterized by checking whether there exists no infinite repetition of selected variant goals or of selected goals with recursive increase in term size, such crucial dynamic characteristics of infinite solved form logic rules are captured by loop goals, as shown by the following theorem.

Theorem 4.8 *Let P be a definite logic program and T be a reverse binary-tree constructed from the computation of solved-form fixpoint of P, then T is infinite if and only if there exists an infinite selection sequence $N_1 :: A_1, \cdots, N_{g_1} :: A_{g_1}, \cdots, N_{g_2} :: A_{g_2}, \cdots, N_{g_i} :: A_{g_i}, \cdots, N_{g_{i+1}} :: A_{g_{i+1}}, \cdots$, of T such that for all i, $A_{g_{i+1}}$ is a loop goal of A_{g_i}.*

Proof. ($\Leftarrow$) Straightforward.
($\Rightarrow$)T is an infinite reverse binary-tree, by the construction rules of selection sequence, there exists an infinite selection sequence $\{N_i :: A_i\}_{i=1}^{\infty}$ and for all i, $A_i \prec_{anc} A_{i+1}$. Since $\{N_i :: A_i\}_{i=1}^{\infty}$ is an infinite selection sequence and the number of the values of all $tag(A_i)$ is finite, from the infinite selection sequence $\{N_i :: A_i\}_{i=1}^{\infty}$, an infinite sub-sequence $\{N_{f_i} :: A_{f_i}\}_{i=1}^{\infty}$ can be constructed from $\{N_i :: A_i\}_{i=1}^{\infty}$ such that all $tag(A_{f_i})$ are equal. For convenience, we denotes $\{N_{f_i} :: A_{f_i}\}_{i=1}^{\infty}$ with $\{N_i :: A_i\}_{i=1}^{\infty}$ also. By the definition of S_{A_i}, X-resolution and the algorithm for computing the most general unifier, S_{A_i} is a string over the alphabet Σ_P consisting of all the function symbols and names in P and the new fresh symbol χ, since Σ_P is finite, by lemma 4.7, for the infinite selection sequence $\{S_{A_i}\}_{i=1}^{\infty}$ over Σ_P, there exists an infinite increasing integer sequence $\{g_i\}_{i=1}^{\infty}$ such that for all $i, A_{g_i} \subseteq_{proj} A_{g_{i+1}}$. Since

$\prec_{anc}$ is transitive, we also have $A_{g_i} \prec_{anc} A_{g_{i+1}}$, thus, for any i, $A_{g_{i+1}}$ is a loop goal of A_{g_i}. □

Theorem 4.9 *Let P be a definite logic program, then the solved-form fixpoint of P terminates if and only if: for each reverse binary-tree T , there exists no infinite selection sequence $N_1 :: A_1, \cdots, N_{g_1} :: A_{g_1}, \cdots, N_{g_2} :: A_{g_2}, \cdots, N_{g_i} :: A_{g_i}, \cdots, N_{g_{i+1}} :: A_{g_{i+1}}, \cdots$, in T such that for all i, $A_{g_{i+1}}$ is a loop goal of A_{g_i}.*

Our termination characterization is equivalent to the termination characterization in [11]. But the loop into relation in our approach is defined restrictively over those heads with identical values of *tag* of solved form logic rules, this will improve the precision and efficiency of our termination prediction algorithm.

4.2 Termination Prediction

Checking the above termination characterization condition is impossible. Instead, an approximation method can be used: an integer k(for example, $k = 3, 4, 5$) is selected as a threshold, if there exists an finite selection sequence $N_1 :: A_1, \cdots, N_{g_1} :: A_{g_1}, \cdots, N_{g_2} :: A_{g_2}, \cdots, N_{g_i} :: A_{g_i}, \cdots, N_{g_{i+1}} :: A_{g_{i+1}}, \cdots$, such that for all $i(k > i \geq 1)$, $A_{g_{i+1}}$ is a loop goal of A_{g_i}, then it is believed that the solved-form fixpoint does not terminate.

To predict termination of solved-form fixpoint, for each computed solved-form logic rule, we check the corresponding reverse binary tree , if there exists an finite selection sequence $N_1 :: A_1, \cdots, N_{g_1} :: A_{g_1}, \cdots, N_{g_2} :: A_{g_2}, \cdots, N_{g_i} :: A_{g_i}, \cdots, N_{g_{i+1}} :: A_{g_{i+1}}, \cdots$, such that for all $i(k > i \geq 1)$, $A_{g_{i+1}}$ is a loop goal of A_{g_i}, then we predict that the solved-form fixpoint does not terminate, otherwise the solved-form fixpoint terminates. The approximation method is used in [11]. The experiment results in section 6 validate the effectiveness of the algorithm for predicting termination of the solved-form fixpoint.

Our predicting algorithm is more precise and more efficient. Firstly, if the loop into relation is not defined restrictively over those heads of offspring of the same logic rule, there maybe some logic rules in a definite logic program, their several offspring are solved form logic rules and the heads of these rules satisfies the loop goal relations, then the algorithm will predict the solved-form fixpoint of the logic program model does not terminate, even though it actually terminates. Thus our predicting algorithm is more precise; Secondly, instead of checking all heads of solved form logic rules to predict termination, we only need to check those heads of offspring of the same logic rule in a definite logic program, so the efficiency of the algorithm will be improved.

5 Fixpoint Abstraction and Refinement

By theorem 3.7, the derivation algorithm terminates if solved-form fixpoint terminates. The variant *depth*(k) abstract domain limits the unbounded increase of terms' depths, which would guarantee termination of the abstraction solved-form fixpoint.

5.1 Fixpoint Abstraction

The abstraction of solved-form fixpoint is based on two abstraction functions: the function β_k defined over terms and the function α_k defined over solved form logic rules. The function β_k is defined inductively as follows:

if $k = 0$, define $\beta_k(t) = z$ for each term t, where z is a fresh variable;

if $k > 0$, define:

$\beta_k(a) = a$, if a is a name;

$\beta_k(x) = x$, if x is a variable;

$\beta_k(f(t_1, \cdots, t_n)) = f(\beta_{k-1}(t_1), \cdots, \beta_{k-1}(t_n))$, if f is a function symbol.

Using fresh variables, the function β_k abstracts terms into terms whose depth is less than or equal to $k + 1$, and limits the unbounded increase of depths of terms. In this paper,we assume that the selected value of term depth bound k is larger or equal to the largest depth of the terms in the definite logic program. The abstraction function α_k is defined using β_k, let $R = H \rightarrow p(M')$ be a solved form logic rule, α_k is defined as follows:

if $\beta_k(M') = M'$, then $\alpha_k(R) = R$;

if $\beta_k(M') \neq M'$, then $\alpha_k(R) = \rightarrow p(M')$.

The function α_k abstracts away the body of R if $\beta_k(M') \neq M'$, otherwise R is maintained in solved-form fixpoint, which is the key distinction of the variant *depth*(k) abstract domain and the *depth*(k) abstract domain. By the definition of rule implication, for each solved form logic rule R, $\alpha_k(R) \Rightarrow R$ holds.

Let P be an definite logic program, define:

$$\alpha^k T^0(P) = \{\alpha_k(elimdup(R)) | R \in P \cap SolvedForm\};$$
$$\alpha^k C^0(P) = \{elimdup(R) | R \in P \cap UnSolvedForm\}$$
$$\alpha^k Rule^0(P) = \alpha^k T^0(P) \cup \alpha^k C^0(P)$$
$$\alpha^k X\_Resolution^{n+1}(P) = \{elimdup(R) | R = R' \circ R'', R' \in \alpha^k T^n(P), R'' \in \alpha^k C^n(P)\}$$
$$\alpha^k T^{n+1}(P) = \{\alpha_k(R') | R' \in addRule(\alpha^k X\_Resolution^{n+1}(P), \alpha^k Rule^n(P)) \cap SolvedForm\}$$
$$\alpha^k C^{n+1}(P) = \{R' | R' \in addRule(\alpha^k X\_Resolution^{n+1}(P), \alpha^k Rule^n(P)) \cap UnSolvedForm\}$$
$$\alpha^k Rule^{n+1}(P) = \alpha^k T^{n+1}(P) \cup \alpha^k C^{n+1}(P)$$

Definition 5.1 Let P be a definite logic program, define $\alpha^k fixpoint(P) = \{\alpha^k T^n(P) | n \geq 0\} \cap UnMarked$, then $\alpha^k fixpoint(P)$ is called the abstract solved-form fixpoint of P.

By the definition of α_k, all rules $R = H \rightarrow p(M')$ in $fixpoint(P)$ are still reserved in $\alpha^k fixpoint(P)$ if the depth of M' is less than or equal to k, which are very fit for constructing derivation witness.

In the following theorem, we prove $\alpha^k fixpoint(P)$ terminates, the main idea is that: for each solved form logic rule $R = H \rightarrow p(M')$ in $\alpha^k fixpoint(P)$, for each false goal $q(x)$ occurs in H, x must occurs in M', otherwise $q(x)$ can be deleted from H, and the depth of M' is less or equal to $k + 1$, so the number of solved form logic rules in $\alpha^k fixpoint(P)$ is finite.

Theorem 5.2 *Let P be a definite logic program, then $\alpha^k fixpoint(P)$ terminates.*

Proof. The function symbols and the names occurring in P are finite, if those terms with variable renaming are considered identical, then the terms constructed from the function symbols, the names occurring in P and variables whose depth is less than or equal to $k + 1$ are finite. Let M be a term whose depth is less than or equal to $k + 1$, let $var(M)$ denote the set of variables occurring in M, $\alpha^k fixpoint(P) \subseteq \cup_{depth(M)\leq k+1} \cup_{i=1}^{|var(M)|} \{p_1(x_1)) \wedge \cdots \wedge p_i(x_i) \rightarrow p(M))\} \cup \{\rightarrow p(M)\}$, where $x_i \in var(M)$. Since the terms whose depth is less than or equal to $k+1$ are finite, the variables occur in these terms are finite,and the number of the predicative symbols in the definite logic program is finite, then $\alpha^k fixpoint(P)$ is a set whose elements are finite, thus $\alpha^k fixpoint(P)$ terminates. □

Lemma 5.3 *Let P be a definite logic program, for each $R \in Rule^n(P)$, there exists $R' \in \alpha^k Rule^n(P)$ such that $R' \Rightarrow R$.*

Proof.

(1) if $n = 0, \alpha^k Rule^0(P) = \alpha^k T^0(P) \cup \alpha^k C^0(P)$, since $\Rightarrow$ is reflexive and $\alpha_k(R) \Rightarrow R$, the conclusion holds;

(2) Assume that the conclusion holds when $n = m \geq 0$, in the case of $n = m + 1$, for each $R \in Rule^{m+1}(P)$, since $Rule^{m+1}(P) = addRule(X\_Resolution^{m+1}(P), Rule^m\ (P))$, if $R \in Rule^m(P)$, then the conclusion holds by the induction assumption; if $R \in X\_Resolution^{m+1}(P)$, then there exists $R_1 \in T^m(P) \subseteq Rule^m(P)$ and $R_2 \in C^m(P) \subseteq Rule^m(P)$ such that $R = elimdup(R_1 \circ R_2)$, by the induction assumption, there exist $R_1', R_2' \in \alpha^k Rule^m(P)$ such that $R_1' \Rightarrow R_1$, $R_2' \Rightarrow R_2$, since $R_1' \Rightarrow R_1$ and $R_1 \in SolvedForm$, then $R_1' \in \alpha^k Rule^m(P) \cap SolvedForm$, by theorem 1, then $R_1' \Rightarrow R_1 \bullet R_2$ or $R_1' \bullet R_2' \Rightarrow R_1 \bullet R_2$, further,$R_1' \Rightarrow elimdup(R_1 \bullet R_2)$ or $R_1' \bullet R_2' \Rightarrow elimdup(R_1 \bullet R_2)$.

case1: $R_1' \Rightarrow elimdup(R_1 \bullet R_2)$.

(1) If there exists $R'' \in \alpha^k X\_Resolution^{m+1}(P)$ such that $R'' \Rightarrow R_1'$, then $\alpha_k(R'') \Rightarrow R_1' \Rightarrow elimdup(R_1 \bullet R_2)$ and $\alpha_k(R'') \in \alpha^k T^{m+1}(P) \subseteq \alpha^k Rule^{m+1}(P)$;

(2) If there exists no $R'' \in \alpha^k X\_Resolution^{m+1}(P)$ such that $R'' \Rightarrow R_1'$, then $\alpha_k(R_1') \Rightarrow elimdup(R_1 \bullet R_2)$ and $\alpha_k(R_1') \in \alpha^k T^{m+1}(P) \subseteq \alpha^k Rule^{m+1}(P)$.

case2: $R_1' \bullet R_2' \Rightarrow elimdup(R_1 \bullet R_2)$.

(1) Since $elimdup(\alpha_k(R_1') \circ R_2') \in \alpha^k X\_Resolution^{m+1}(P)$, if there exists no $R'' \in \alpha^k Rule^m(P)$ such that $R'' \Rightarrow elimdup(\alpha_k(R_1') \circ R_2')$, if $elimdup(\alpha_k(R_1') \circ R_2') \in SolvedForm$,then $\alpha_k(elimdup(\alpha_k(R_1') \circ R_2')) \Rightarrow elimdup(R_1 \bullet R_2)$ and $\alpha_k(elimdup(\alpha_k(R_1') \circ R_2')) \in \alpha^k T^{n+1}(P) \subseteq \alpha^k Rule^{m+1}(P)$; if $elimdup(\alpha_k(R_1') \circ R_2') \in UnSolvedForm$, then $elimdup(\alpha_k(R_1') \circ R_2') \Rightarrow elimdup(R_1 \bullet R_2)$ and $elimdup(\alpha_k(R_1') \circ R_2') \in \alpha^k C^{n+1}(P) \subseteq \alpha^k Rule^{m+1}(P)$;

(2) if there exists $R'' \in \alpha^k Rule^m(P)$ such that $R'' \Rightarrow elimdup(\alpha_k(R_1') \bullet R_2')$, then $R'' \Rightarrow elimdup(\alpha_k(R_1') \bullet R_2') \Rightarrow elimdup(R_1 \bullet R_2)$, if $R'' \in SolvedForm$, then $\alpha_k(R'') \Rightarrow elimdup(R_1 \bullet R_2)$ and $\alpha_k(R'') \in \alpha^k T^{m+1}(P) \subseteq \alpha^k Rule^{m+1}(P)$,if $R'' \in UnSolvedForm$, then $R'' \Rightarrow elimdup(R_1 \bullet R_2)$ and $R'' \in \alpha^k C^{m+1}(P) \subseteq$

$\alpha^k Rule^{m+1}(P)$.

Thus the conclusion holds for $n = m + 1$. □

With lemma 5.3, the following theorem proves $\alpha^k fixpoint(P)$ is a safe approximation of $fixpoint(P)$, the main idea is that: for each closed atom F, if there exists a derivation tree of F from $fixpoint(P)$, then there exists a derivation tree of F from $\alpha^k fixpoint(P)$.

Theorem 5.4 *Let P be a definite logic program and F be a closed atom, if F is derivable from $fixpoint(P)$, then F is also derivable from $\alpha^k fixpoint(P)$.*

Proof. F is derivable from $fixpoint(P)$, then there exists a derivable tree T of F from $fixpoint(P)$. For each node m in T, assume the node m is labelled by $R \in fixpoint(P)$ with an incoming edge labelled by F_0 and n outgoing edges labelled by $F_1, \cdots, F_n$, then $R \Rightarrow F_1 \wedge \cdots \wedge F_n \rightarrow F_0$, since $R \in fixpoint(P) = \{T^n(P) | n \geq 0\} \cap UnMarked$, by lemma 5.3, there exists $R' \in \{\alpha^k Rule^n(P) | n \geq 0\}$ such that $R' \Rightarrow R \Rightarrow F_1 \wedge \cdots \wedge F_n \rightarrow F_0$, since $R' \Rightarrow R$ and $R \in SolvedForm$, then $R' \in SolvedForm$ and $R' \in \{\alpha^k T^n(P) | n \geq 0\}$, if $R' \in \alpha^k fixpoint(P)$, then replace R by R' in T, if $R' \notin \alpha^k fixpoint(P)$, by the definition $\alpha^k fixpoint(P) = \{\alpha^k T^n(P) | n \geq 0\} \cap UnMarked$, then there exists $R'' \in \alpha^k fixpoint(P)$ such that $R'' \Rightarrow R'$, replace R by R'' in T. Repeat this procedure until all the rules in $fixpoint(P)$ are replaced by rules in $\alpha^k fixpoint(P)$, then the derivation tree of F from $\alpha^k fixpoint(P)$ is constructed, thus F is derivable from $\alpha^k fixpoint(P)$. □

Theorem 5.4 shows that if F is not derivable from $\alpha^k fixpoint(P)$, then it is not derivable from $fixpoint(P)$ also; If F is derivable from $\alpha^k fixpoint(P)$, then it may or may not derivable from $fixpoint(P)$. If $fixpoint(P)$ does not terminate, it can be replaced by $\alpha^k fixpoint(P)$ in the derivation algorithm as follows:

```
if $\exists R' \in B, R' \Rightarrow R$, then  $derivablerec(R, B, P) = \emptyset$
else if $R = \rightarrow C$, then  $derivablerec(R, B, P) = \{\rightarrow C\}$
    else $derivablerec(R, B, P) = \{derivablerec(elimdup(R' \bullet R), \{R\} \cup B, P) | R' \in$
        $\alpha^k fixpoint(P)\}$
```

By theorem 5.2, $\alpha^k fixpoint(P)$ terminates, then by lemma 5.3, the derivation algorithm which uses $\alpha^k fixpoint(P)$ terminates.

5.2 Fixpoint refinement

Let P be a definite logic program and $\alpha^k fixpoint(P)$ be the abstract solved-form fixpoint, the set of logic rules in $\alpha^k fixpoint(P)$ which are not abstracted by α_k, denoted by $UnAbstract$, is defined inductively as follows:

(1) Let $R = H \rightarrow q(M') \in P \cap SolvedForm$, if $\beta_k(M') = M'$, then $\alpha_k(elimdup(R)) \in UnAbstract$;

(2) If $R \in \alpha^k C^0(P)$, then $R \in UnAbstract$;

(3) If there exists $R' \in \alpha^k T^n(P) \cap UnAbstract$ and $R'' \in \alpha_k C^n(P) \cap UnAbstract$ such that $R = elimdup(R' \circ R'')$, then $R \in UnAbstract$,

Definition 5.5 Let P be a definite logic program, define $\alpha^k partialfixpoint(P) = \{\alpha^k T^n(P) | n \geq 0\} \cap UnAbstract$, $\alpha^k partialfixpoint(P)$ is called the partial solved-form fixpoint of P.

The partial solved-form fixpoint $\alpha^k partialfixpoint(P)$ of P consists of all the solved form logic rules whose derivation is not abstracted by α_k.

If the derivation algorithm with $\alpha^k fixpoint(P)$ shows the closed atom F is derivable from the definite logic program, we run the derivation algorithm with $\alpha^k partialfixpoint(P)$ as follows:

```
  if $\exists R' \in B, R' \Rightarrow R$, then $derivablerec(R, B, P) = \emptyset$
 else if $R = \rightarrow C$, then $derivablerec(R, B, P) = \rightarrow C$
    else $derivablerec(R, B, P) = \{derivablerec(elimdup(R' \circ R), R \cup B, P) | R' \in$
          $\alpha^k partialfixpoint(P)\}$
```

By theorem 3.9, if F is derivable, the derivation witness can be constructed from $\alpha^k partialfixpoint(P)$ by the approach presented in [4]. If F is derivable from $\alpha^k fixpoint(P)$, but F is not derivable from $\alpha^k partialfixpoint(P)$, we increase the threshold of the term depth bound k, compute $\alpha^{k+1} fixpoint(P)$, and run the derivation algorithm with $\alpha^{k+1} fixpoint(P)$ again.

The following theorem shows $\alpha^{k+s} partialfixpoint(P)(s \geq 0)$ is a refinement of $\alpha^k partialfixpoint(P)$.

Theorem 5.6 *Let P be a definite logic program, then for each $s \geq 0$, $\alpha^k partialfixpoint(P) \subseteq \alpha^{k+s} partialfixpoint(P)$.*

Proof. For each $n \geq 0$, we prove that $\alpha^k T^n(P) \cap UnAbstract \subseteq \alpha^{k+s} T^n(P) \cap UnAbstract$ and$\alpha^k C^n(P) \cap UnAbstract \subseteq \alpha^{k+s} C^n(P) \cap UnAbstract$.

If $n = 0$, by the definition of α_k, $\alpha^k T^0(P) \cap UnAbstract = \alpha^{k+s} T^0(P) \cap UnAbstract$, $\alpha^k C^0(P) \cap UnAbstract = \alpha^{k+s} C^0(P) \cap UnAbstract$, the conclusion holds.

Assume that the conclusion holds when $n = m \geq 0$, in the case of $n = m + 1$, let $R \in \alpha^k T^{m+1}(P) \cap UnAbstract$, then $R \in \alpha^k T^m(P) \cap UnAbstract$ or $R \in \alpha^k X\_Resolution^{m+1}(P) \cap UnAbstract$. If $R \in \alpha^k T^m(P) \cap UnAbstract$, by the induction assumption, $R \in \alpha^{k+s} T^m(P) \cap UnAbstract$. If $R \in \alpha^k X\_Resolution^{m+1}(P) \cap UnAbstract$, then $R = elimdup(R' \circ R'')$, where $R' \in \alpha^k T^m(P) \cap UnAbstract$, $R'' \in \alpha^k C^m(P) \cap UnAbstract$, by the induction assumption, $R' \in \alpha^{k+s} T^m(P) \cap UnAbstract$, $R'' \in \alpha^{k+s} C^m(P) \cap UnAbstract$, thus $R \in \alpha^{k+s} X\_Resolution^{m+1}(P) \cap UnAbstract$. By the fact $R \in \alpha^{k+s} T^m(P) \cap UnAbstract$ or $R \in \alpha^{k+s} X\_Resolution^{m+1}(P) \cap UnAbstract$, then $R \in \alpha^{k+s} T^{m+1}(P) \cap UnAbstract$.

The fact that $\alpha^k C^n(P) \cap UnAbstract \subseteq \alpha^{k+s} C^n(P) \cap UnAbstract$ can be proved in the similar way. □

Since $fixpoint(P) \subseteq \cup_{k \geq 0} \{\alpha^k T^n(P)\}$ and $fixpoint(P) \subseteq UnAbstract$, it is easy to see that $fixpoint(P) \subseteq \cup_{k \geq 0} \alpha^k partialfixpoint(P)$, which means that the derivation witness can be constructed from $\alpha^k partialfixpoint(P)$ if the value of k is large enough.

Compared with the counterexample-driven abstraction refinement iteration analysis framework, our framework needn't decide whether the constructed derivation witness is false or not, all of the derivation, constructing derivation witness and refinement can be implemented in a mechanized way.

6 Experiments

To demonstrate the effectiveness of our approach,we have implemented these algorithms in our verifier prototype SPVT[12] for security protocols, and the security protocols in [10] are used to validate the effectiveness.

Table 2 shows the experiment results of termination prediction for solved-form fixpoint, where $k = 3$ is selected as the threshold. The experiment results in Table 2 shows that: for many protocols, their solved-form fixpoints terminate almost if and only if the prediction algorithm predicts it terminates.

The solved-form fixpoints of the Needham-Schroeder shared-key protocol and the Woo-Lam shared-key one-way authentication protocol Π_3 do not terminate, the time for running termination prediction algorithm is 0.078s and 0.109s respectively.

Security Protocols	Termination	Prediction Result
Simplified NS Public-key Authentication Protocol	true	true
NSL Public-key Authentication Protocol	true	true
NS Shared-key protocol	false	false
Yahalom Protocol	true	true
Otway-Rees Protocol	true	true
Woo-Lam Authentication Protocol Π	true	true
Woo-Lam Authentication Protocol Π_1	true	true
Woo-Lam Authentication Protocol Π_2	true	true
Woo-Lam Authentication Protocol Π_3	false	false
Woo-Lam Authentication Protocol Π_f	true	true

Table 2
The Experiment Results of Termination Prediction

Table 3 lists the run time of the abstract fixpoint of Π_3 when term depth bound $k = 3, 4, 5$. And when $k = 5$, the abstract-refinement iterative verification approach terminates since SPVT have constructed a counterexamples described as follows:

Term Depth Bound	Time
3	0.031
4	0.032
5	0.281

Table 3
The Experiment Results of the Woo-Lam shared-key one-way authentication protocol Π_3

$$\begin{aligned}
&host(k_{IS})(host(k_{AS})) \to host(k_{BS}): && host(k_{AS});\\
&host(k_{BS}) \to host(k_{IS})(host(k_{AS})): && N[i_B^1, host(k_{AS})];\\
&host(k_{IS})(host(k_{AS})) \to host(k_{BS}): && N[i_B^1, host(k_{AS})];\\
&host(k_{BS}) \to host(k_{IS})(host(k_{SS})): && encrypt(2tuple(host(k_{AS}), N[i_B^1, host(k_{AS})]),\\
& && k_{BS});\\
&host(k_{IS})(host(k_{SS})) \to host(k_{BS}): && encrypt(2tuple(host(k_{AS}), N[i_B^1, host(k_{AS})]),\\
& && k_{BS});
\end{aligned}$$

The above counterexample is the attack of Π_3 described in [10].

7 Conclusions

In this paper, we firstly present a termination prediction algorithm of solved-form fixpoint of definite logic program. Based on the prediction algorithm, the non-terminational accurate approach and the terminated abstract approach are combined together to solve the derivation problem more efficiently. The experimental results show the termination prediction algorithm is practical, and validate the effectiveness of the novel derivation framework for definite logic program.

References

[1] Cousot,P. and Cousot,R., *Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in: *Symposium on Principles of Programming Languages*, 1977, pp. 238–252.

[2] Blanchet, B., *An efficient cryptographic protocol verifier based on prolog rules*, in: *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, 2001, pp. 82–96.

[3] Mengjun Li, Ti Zhou, Zhoujun Li, HuoWang Chen, *An Abstraction and Refinement Framework for Verifying Security Protocols Based on Logic Programming*, in: *ASIAN*, 2007, pp. 166–180.

[4] Allamigeon,X. and Blanchet, B., *Reconstruction of Attacks against Cryptography Protocols*, in: *18th IEEE Computer Security Foundations Workshop (CSFW-18)*, 2005, pp. 140–154.

[5] Roberta Gori, Ernesto Lastres, Ren Moreno, and Fausto Spoto., *Approximation of the Well-Founded Semantics for Normal Logic Programs using Abstract Interpretation*, in: *APPIA-GULP-PRODE '98 Conference*,1998, pp. 433–441.

[6] John P. Gallagher and D. Andre de Waal., *Fast and precise Regular Approximation of Logic Program*, in: *ICLP*, 1994, pp. 599–613.

[7] Cousot,P. and Cousot,R., *Abstract Interpretation and Application to Logic Programs*, Journal of Logic Programming,**13** (1992), pp. 103–179.

[8] Baudouin Charlier,Sabina Rossi, and Pascal van Hentenryck., *Sequence-based abstract interpretation of prolog*,Theory and Practice of Logic Programming,**2**,2002,pp.25–84.

[9] Agostino Cortesi, Baudouin Le Charlier, Pascal van Hentenryck., *Combinations of abstract domains for logic programming: open product and generic pattern construction*,Science of Computer Programming ,**38**,2000,pp.27–71.

[10] Clark, J. A. and J. L. Jacob., *A survey of authentication protocol literature*, Technical Report 1.0 (1997).

[11] Yi-Dong Shen, Jia-Huai You, Li-Yan Yuan, Samuel S. P. Shen, Qiang Yang, *A dynamic approach to characterizing termination of general logic programs*, ACM Trans. Comput. Log.,**4**,2003,pp.417-430 .

[12] Mengjun Li, Zhoujun Li, HuoWang Chen, *SPVT: An efficient verification tool for security protocol*, Chinese Journal of Software **17** (2006), pp. 898–906.