Electronic Notes in Theoretical Computer Science 212 (2008) 71–85 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

A Novel Derivation Framework For Definite Logic Program

MengJun Li[a](#_bookmark0)*,*[1](#_bookmark1) ZhouJun Li[b](#_bookmark2)*,*[2](#_bookmark1) HuoWang Chen[a](#_bookmark0)*,*[3](#_bookmark1) Ti Zhou[a](#_bookmark0)*,*[4](#_bookmark1)

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 termina- tion, 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.

# 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 pro- tocols 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](#_bookmark3)],

*٨* 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](mailto:mengjun_li1975@yahoo.com.cn)

2 Email: [zhoujun.li@buaa.edu.cn](mailto:zhoujun.li@buaa.edu.cn)

3 Email: [hwchen@nudt.edu.cn](mailto:hwchen@nudt.edu.cn)

4 Email: [tizhou@gmail.com](mailto:tizhou@gmail.com)

1571-0661 © 2008 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

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](#_bookmark6)][[6](#_bookmark8)][[7](#_bookmark9)] [[8](#_bookmark10)][[9](#_bookmark11)].

There exist two categories approaches: the accurate approach that does not guar- antee termination, and the terminated abstract approaches. Both approaches have its advantages and disadvantages. This paper presents a novel derivation frame- work 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](#_bookmark4)], 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 approxima- tion of the fixpoint based on a variant of the *depth*(*k*) abstract domain[[5](#_bookmark6)], and refine the computed abstract fixpoint by increasing the threshold *k* of the *depth*(*k*) abstract domain[[3](#_bookmark5)]. The constructed abstract fixpoint terminates, and using which instead of the fixpoint the derivation algorithm will termi- nate. 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 con- structed using the same approach in [[4](#_bookmark7)]. 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 ad- vance, 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 algo- rithm, 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](#_bookmark6)][[6](#_bookmark8)][[7](#_bookmark9)] [[8](#_bookmark10)][[9](#_bookmark11)]. 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 ad- vance, 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 algo- rithms, 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 pro- gram 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.

# Definite Logic Program

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

|  |  |  |  |
| --- | --- | --- | --- |
| *m, s, t* ::= | *T erms* | *F, C, A* ::= | *Atom, Fact* |
| *x, y, z* | *V ariables* | *p*(*t*)*, q*(*t*) | *Predicates* |
| *a, b, c* | *Constants* | '  *R, R* | *Rules* |
| *f* (*M*1*,* ··· *, Mn*) | *f unctions* | *F*1 ∧ ··· ∧ *Fn* → *F* | *Logic Rules* |

Table 1

The Syntax of Definite Logic Program

Definition 2.1 Let *R*1 = *H*1 ∧···∧ *H*1 → *C*1 and *R*2 = *H*2 ∧···∧ *H*2 → *C*2 be two

1 *m* 1 *n*

logic rules, if *C*1 = *p*(*t*1), *C*2 = *p*(*t*2), define rule implication *R*1 ⇒ *R*2 if and only

if there exists a substitution *θ* such that: *t*1*θ* = *t*2, and for each *H*1(1 ≤ *i* ≤ *m*),

*i*

*H*1*θ* ∈ {*H*2*,* ··· *,H*2}.

*i* 1 *n*

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* ∈ *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*,* ··· *, Fn*, then *R* ⇒ *F*1 ∧ ··· ∧ *Fn* → *F*0.

1. 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.

# Derivation algorithm

Definition 3.1 Let *R* = *H* → *F* and *R*

'

' '

= *H* → *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* • *R* between *R* and *R*

'

is (*H* ∧ (*H*

— *F*0))*θ* → *F θ*, *θ* = *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* → *C* be a logic rule, if the atoms in *H* are all false goals, then we say *H* → *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* → *F* and *R* = *H*

'

'

'

→ *F* be two logic rules, *R* ∈

*SolvedForm*, *R*' ∈

'

' '

*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*

* *R* between *R* and *R* is

' ' '

(*H* ∧ (*H* − *F*0))*θ* → *F θ, θ* = *mgu*(*t, t* ).

Let *R* be a logic rule and *B* be a logic rule set, define *addRule*(*R, B*) as:

'

if ∃*R*

'

∈ *B, R*

⇒ *R*, then *addRule*(*R, B*)= *B*

' '

else *addRule*(*R, B*)= {*R*}∪ {*R* |*R*

''

' ''

∈ *B, R* /⇒ *R* }∪ {*marked*(*R*

''

)|*R*

∈ *B,*

*R* ⇒ *R* }

'' ''

where *marked*(*R*

define:

) denotes that *R* will not be used to compute X-resolutions. And

*addRule*({*R*1*,* ··· *, Rm*}*, B*)= *addRule*({*R*2*,* ··· *, Rm*}*, 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 ∧ ··· ∧

'

*Fn* → *C* be a logic rule, the unary function *elimdup*(*R*) returns a rule *R* such that:

1. In {*F*1*,* ··· *, Fn*}, only those atoms that satisfies the following conditions will

'

occur in the body of *R* : if *j < i*,then *Fi* /= *Fj* ;

'

1. *C* is the head of the rule *R* ; Let *P* be a definite logic program, define:

*Rule*0(*P* )= {*elimdup*(*R*)|*R* ∈ *P* };

*T* 0(*P* )= *Rule*0(*P* ) ∩ *SolvedForm*; *C*0(*P* )= *Rule*0(*P* ) ∩ *UnSolvedForm*;

*X Resolutionn*+1(*P* ) = {*elimdup*(*R*)|*R* = *R*' ◦

*Cn*(*P* )};

'' '

*R ,R* ∈

''

*T* (*P* )*,R* ∈

*n*

*Rulen*+1(*P* )= *addRule*(*X Resolutionn*+1(*P* )*, Rulen*(*P* ));

*T n*+1(*P* )= *Rulen*+1(*P* ) ∩ *SolvedForm*; *Cn*+1(*P* )= *Rulen*+1(*P* ) ∩ *UnSolvedForm*;

Definition 3.5 Let *P* be a definite logic program, define *f ixpoint*(*P* ) =

{*T n*(*P* )|*n* ≥ 0} ∩ *UnMarked*, *f ixpoint*(*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 ∃*R*

'

∈ *B, R*

⇒ *R*, then *derivablerec*(*R, B, P* )= ∅

else if *R* =→ *C*, then *derivablerec*(*R, B, P* )= → *C*

'

else *derivablerec*(*R, B, P* )= {*derivablerec*(*elimdup*(*R* • *R*)*,* {*R*}∪ *B, P* )|

'

*R* ∈ *f ixpoint*(*P* )}

And define *derivable*(*F, P* )= *derivablerec*(*F* → *F,* ∅*,P* ).

' ' ' '

Theorem 3.6 *If R* • *R is deﬁned, R*1 ⇒ *R and R*1 ⇒ *R , then either R*1 • *R*1 *is*

' ' ' '

*deﬁned and R*1 • *R*1 ⇒ *R* • *R , or R*1 ⇒ *R* • *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 f ixpoint*(*P* )*.*

Theorem 3.9 *Let P be a deﬁnite logic program and F be a closed atom, then F is derivable from f ixpoint*(*P* ) *if and only if* → *F* ∈ *derivable*(*F, P* )*.*

The above four theorems are variants of the corresponding theorems in [[2](#_bookmark4)].

# Termination Characterization and Prediction

Based on the dynamic approach presented in [[11](#_bookmark12)], a corresponding dynamic ap- proach to characterize termination of solved-form fixpoint is presented, and which is used to predict termination of solved-form fixpoint in advance.

* 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 *ith*

'

rule of *P* and *R* = *elimdup*(*R* ), then *tag*(*A*)= *i*;

'

1. if *R* = *elimdup*(*R*

'' ''

* *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](#_bookmark12)]. 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 vari- ant 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* → *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 *ST* , 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 *ST* = *fχgχfaχ*. 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 *ST*1 and *ST*2 be two symbolic strings, *ST*1 is a projection of *ST*2 , denoted *ST*1 ⊆*proj ST*2 , if *ST*1 is obtained from *ST*2 by removing one or more elements.

For example, *aχχbc* ⊆*proj faχaχχbχc*. 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 ◦ *R*1).

· ··· ·· ··· ··· ··· ·· ··· ··· ··· ·· ··· ··· ··· ·· ···

\

*,*\\

/

/r/*, ,,*

\ *J* / *J*

/

\

*, ,,*

*,*

\ /r

*R*1 *R*2

\ *J* / *J*

\

/

\ /r *, ,,*

*R*

*,*

\ *J* / *J*

*,*\ /r *,*

\ *J*

\

/

\

· ··· ·· ··· ··· ··· ·· ··· ··· ··· ·· ··· ··· ··· ·· ···

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 ≺*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* → *F* is equal to one adding the depth of the node labelled by *R*1 = *H*1 → *F*1, or the node labelled by *R*2 = *H*2 → *F*2, then the ancestor-descendant relation ≺*anc* among *F, F*1*, F*2 is: *F*1 ≺*anc F, F*2 ≺*anc F* . For convenience, *Ni* :: *Ai* is used to denote a node *Ni* and

the atom *Ai* which is the head of the logic rule labelling *Ni*.

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 ~*loop A*2, if *SA*1 ⊆*proj SA*2 and *tag*(*A*1)= *tag*(*A*2).

Compared with the definition in [[11](#_bookmark12)], 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 *Ni, Nj* be two nodes in *T* , if

*Ai* ≺*anc Aj* and *Ai* ~*loop Aj*, then *Aj* is called a loop goal of *Ai*.

Definition 4.6 Let *T* be a reverse binary-tree, the sequence constructed induc- tively 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 *Nl* is *l* and it is labelled by a logic rule in *SolvedForm*, then *Nl* :: *Al* is added into the selection sequence.

Lemma 4.7 *[*[*11*](#_bookmark12)*] Let* {*Ai*}∞

*i*=1

*be an inﬁnite sequence of strings over a ﬁnite alphabet*

Σ*, then there is an inﬁnite increasing integer sequence* {*ni*}∞

*i*=1

*such that for all*

*i, Ani* ⊆*proj Ani*+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 con- structed 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*,* ··· *, Ng*1 :: *Ag*1 *,* ··· *, Ng*2 :: *Ag*2 *,* ··· *, Ngi* :: *Agi ,* ··· *, Ngi*+1 :: *Agi*+1 *,* ·· ·*, of T such that for all i, Agi*+1 *is a loop goal of Agi.*

Proof. (⇐) Straightforward.

(⇒)*T* is an infinite reverse binary-tree, by the construction rules of selection se-

quence, there exists an infinite selection sequence {*Ni* :: *Ai*}∞

*i*=1

and for all *i*,

*Ai* ≺*anc Ai*+1. Since {*Ni* :: *Ai*}∞

*i*=1

is an infinite selection sequence and the number of

the values of all *tag*(*Ai*) is finite, from the infinite selection sequence {*Ni* :: *Ai*}∞ ,

*i*=1

an infinite sub-sequence {*Nfi* :: *Afi* }∞ can be constructed from {*Ni* :: *Ai*}∞

*i*=1

*i*=1

such that all *tag*(*Afi* ) are equal. For convenience, we denotes {*Nfi* :: *Afi* }∞

*i*=1

with

{*Ni* :: *Ai*}∞

*i*=1

also. By the definition of *SAi* , X-resolution and the algorithm for com-

puting the most general unifier, *SAi* 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 {*SAi* }∞

*i*=1

over Σ*P* , there exists an

infinite increasing integer sequence {*gi*}∞ such that for all *i, Agi* ⊆*proj Agi*+1 . Since

*i*=1

≺*anc* is transitive, we also have *Agi* ≺*anc Agi*+1 , thus, for any *i*, *Agi*+1 is a loop goal of *Agi* .

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*,* ··· *, Ng*1 :: *Ag*1 *,* ··· *, Ng*2 :: *Ag*2 *,* ··· *, Ngi* :: *Agi,* ··· *, Ngi*+1 :: *Agi*+1 *,* ·· ·*, in T such that for all i, Agi*+1 *is a loop goal of Agi .*

Our termination characterization is equivalent to the termination characteriza- tion in [[11](#_bookmark12)]. 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.

* 1. *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*,* ··· *, Ng*1 :: *Ag*1 *,* ··· *, Ng*2 :: *Ag*2 *,* ··· *, Ngi* :: *Agi,* ··· *, Ngi*+1 :: *Agi*+1 *,* ··· *,* such that for all *i*(*k > i* ≥ 1), *Agi*+1 is a loop goal of *Agi* , 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 se- lection sequence *N*1 :: *A*1*,* ··· *, Ng*1 :: *Ag*1 *,* ··· *, Ng*2 :: *Ag*2 *,* ··· *, Ngi* :: *Agi ,* ··· *, Ngi*+1 :: *Agi*+1 *,* ··· *,* such that for all *i*(*k > i* ≥ 1), *Agi*+1 is a loop goal of *Agi* , then we predict that the solved-form fixpoint does not terminate, otherwise the solved-form fixpoint terminates. The approximation method is used in [[11](#_bookmark12)]. 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.

# Fixpoint Abstraction and Refinement

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

* 1. *Fixpoint Abstraction*

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

if *k* = 0, define *βk*(*t*)= *z* for each term *t*, where *z* is a fresh variable; if *k >* 0, define:

*βk*(*a*)= *a*, if *a* is a name;

*βk*(*x*)= *x*, if *x* is a variable;

*βk*(*f* (*t*1*,* ··· *, tn*)) = *f* (*βk*−1(*t*1)*,* ··· *, βk*−1(*tn*)), if *f* is a function symbol.

Using fresh variables, the function *βk* 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 *αk* is defined using *βk*, let *R* = *H* → *p*(*M* ) be a solved form logic rule, *αk*

is defined as follows:

' '

if *βk*(*M* )= *M* , then *αk*(*R*)= *R*;

' ' '

if *βk*(*M* ) /= *M* , then *αk*(*R*) =→ *p*(*M* ).

' '

The function *αk* abstracts away the body of *R* if *βk*(*M* ) /= *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*, *αk*(*R*) ⇒ *R* holds.

Let *P* be an definite logic program, define:

*αkT* 0(*P* )= {*αk*(*elimdup*(*R*))|*R* ∈ *P* ∩ *SolvedForm*}; *αkC*0(*P* )= {*elimdup*(*R*)|*R* ∈ *P* ∩ *UnSolvedForm*} *αkRule*0(*P* )= *αkT* 0(*P* ) ∪ *αkC*0(*P* )

*αkX Resolutionn*+1(*P* )= {*elimdup*(*R*)|*R* = *R*' ◦ *R*'' ' ∈ *α T* (*P* )*,*

*k n*

*,R*

*R*'' ∈ *αkCn*(*P* )}

' '

*αkT n*+1(*P* )= { |*R* ∈ *addRule*(*αkX Resolutionn*+1(*P* )*, αkRulen*(*P* ))∩

*αk*(*R* )

*SolvedForm*}

*αkCn*+1(*P* )= {*R*' |*R*' ∈ *addRule*(*αkX Resolutionn*+1(*P* )*, αkRulen*(*P* ))∩

*UnSolvedForm*}

*αkRulen*+1(*P* )= *αkT n*+1(*P* ) ∪ *αkCn*+1(*P* )

Definition 5.1 Let *P* be a definite logic program, define *αkf ixpoint*(*P* ) =

{*αkT n*(*P* )|*n* ≥ 0}∩ *UnMarked*, then *αkf ixpoint*(*P* ) is called the abstract solved- form fixpoint of *P* .

'

By the definition of *αk*, all rules *R* = *H* → *p*(*M* ) in *f ixpoint*(*P* ) are still

reserved in *αk*

'

*f ixpoint*(*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 *αkf ixpoint*(*P* ) terminates, the main idea is

'

that: for each solved form logic rule *R* = *H* → *p*(*M* ) in *αk*

'

*f ixpoint*(*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 *αkf ixpoint*(*P* ) is finite.

Theorem 5.2 *Let P be a deﬁnite logic program, then αkf ixpoint*(*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* , *αkf ixpoint*(*P* ) ⊆ ∪*depth*(*M*)≤*k*+1 ∪|*var*(*M*)| {*p*1(*x*1)) ∧ ··· ∧ *pi*(*xi*) → *p*(*M* ))} ∪ {→ *p*(*M* )} , where *xi* ∈ *var*(*M* ). Since the terms whose depth is less than or equal to

*i*=1

*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 *αkf ixpoint*(*P* ) isa set whose elements are finite, thus *αkf ixpoint*(*P* ) terminates.

Lemma 5.3 *Let P be a deﬁnite logic program, for each R* ∈ *Rulen*(*P* )*, there exists*

*R*' ∈ *αkRulen*(*P* ) *such that R*' ⇒ *R.*

Proof.

1. if *n* = 0*, αkRule*0(*P* )= *αkT* 0(*P* ) ∪ *αkC*0(*P* ), since ⇒ is reflexive and *αk*(*R*) ⇒ *R*, the conclusion holds;
2. Assume that the conclusion holds when *n* = *m* ≥ 0, in the case of *n* = *m* + 1, for each *R* ∈ *Rulem*+1(*P* ) , since *Rulem*+1(*P* ) = *addRule*(*X Resolutionm*+1(*P* )*, Rulem* (*P* )), if *R* ∈ *Rulem*(*P* ), then the conclu- sion holds by the induction assumption; if *R* ∈ *X Resolutionm*+1(*P* ), then there exists *R*1 ∈ *T m*(*P* ) ⊆ *Rulem*(*P* ) and *R*2 ∈ *Cm*(*P* ) ⊆ *Rulem*(*P* ) such that *R* =

' '

*elimdup*(*R* ◦

∈ *αkRulem*(*P* )

1 *R*2), by the induction assumption, there exist *R*1*, R*2

' ' '

such that *R*1 ⇒ *R*1, *R*2 ⇒ *R*2, since *R*1 ⇒ *R*1 and *R*1 ∈ *SolvedForm*,

' '

then *R* ∈ *αkRulem*(*P* ) ∩ *SolvedForm*, by theorem 1, then *R* ⇒ *R* • *R* or

' ' 1 '

' ' 1 1 2

*R*1 •*R*2 ⇒ *R*1•*R*2, further,*R*1 ⇒ *elimdup*(*R*1 •*R*2) or *R*1 •*R*2 ⇒ *elimdup*(*R*1 •*R*2).

'

case1: *R*1 ⇒ *elimdup*(*R*1 • *R*2).

* 1. If there exists *R*'' ∈ *αkX Resolutionm*+1(*P* ) such that *R*'' ⇒ *R*' , then

1

''

*αk*(*R*

' ''

) ⇒ *R*1 ⇒ *elimdup*(*R*1 • *R*2) and *αk*(*R*

) ∈ *αkT*

*m*+1

(*P* ) ⊆

*αkRulem*+1(*P* );

* 1. If there exists no *R*'' ∈ *αkX Resolutionm*+1(*P* ) such that *R*'' ⇒

'

*R*1, then

' ' *k*

*m*+1

*k m*+1

*αk*(*R*1) ⇒ *elimdup*(*R*1 • *R*2) and *αk*(*R*1) ∈ *α T*

(*P* ) ⊆ *α Rule*

(*P* ).

' '

case2: *R*1 • *R*2 ⇒ *elimdup*(*R*1 • *R*2).

' ' *k*

*m*+1

1. Since *elimdup*(*αk*(*R*1) ◦ *R*2) ∈ *α X Resolution* (*P* ), if there exists no

*R*'' ∈ *αkRulem*(*P* ) such that *R*'' ⇒ ' ◦ ' ' ◦

*elimdup*(*αk*(*R* )

1

'

' '

*R*2), if *elimdup*(*αk*(*R*1)

*R*2) ∈ *SolvedForm*,then *αk*(*elimdup*(*αk*(*R*1) ◦ *R*2)) ⇒ *elimdup*(*R*1 •

' ' *k*

*n*+1

*k m*+1

*R*2) and *αk*(*elimdup*(*αk*(*R*1) ◦ *R*2)) ∈ *α T* (*P* ) ⊆ *α Rule* (*P* ); if

' ' ' '

*elimdup*(*αk*(*R*1) ◦ *R*2) ∈ *UnSolvedForm*, then *elimdup*(*αk*(*R*1) ◦ *R*2) ⇒

*elimdup*(*R*1

'

* *R*2) and *elimdup*(*αk*(*R*1)

'

* *R*2)

∈ *αkCn*+1(*P* ) ⊆ *αkRulem*+1(*P* );

1. if there exists *R*'' ∈ *αkRulem*(*P* ) such that *R*'' ⇒*elimdup*(*α* (*R*' )

*k* 1

'

* *R*2), then

'' ' ''

'

*R* ⇒

*elimdup*(*αk*(*R*1) • *R*2) ⇒ *elimdup*(*R*1 • *R*2), if *R*

∈ *SolvedForm*, then

''

*αk*(*R*

''

) ⇒ *elimdup*(*R*1 • *R*2) and *αk*(*R*

) ∈ *αkT*

*m*+1

(*P* ) ⊆ *αk*

*Rulem*+1

(*P* ),if

'' ''

*R* ∈ *UnSolvedForm*, then *R* ⇒ *elimdup*(*R*1

2

* *R* ) and *R*'' ∈ *αkCm*+1(*P* ) ⊆

*αkRulem*+1(*P* ).

Thus the conclusion holds for *n* = *m* + 1.

With lemma 5.3, the following theorem proves *αkf ixpoint*(*P* ) is a safe approxi- mation of *f ixpoint*(*P* ), the main idea is that: for each closed atom *F* , if there exists a derivation tree of *F* from *f ixpoint*(*P* ), then there exists a derivation tree of *F* from *αkf ixpoint*(*P* ).

Theorem 5.4 *Let P be a deﬁnite logic program and F be a closed atom, if F is derivable from f ixpoint*(*P* )*, then F is also derivable from αkf ixpoint*(*P* )*.*

Proof. *F* is derivable from *f ixpoint*(*P* ), then there exists a derivable tree *T* of *F* from *f ixpoint*(*P* ). For each node *m* in *T* , assume the node *m* is labelled by *R* ∈ *f ixpoint*(*P* ) with an incoming edge labelled by *F*0 and *n* outgoing edges labelled by *F*1*,* ··· *, Fn*, then *R* ⇒ *F*1 ∧ ··· ∧ *Fn* → *F*0, since *R* ∈ *f ixpoint*(*P* )= {*T n*(*P* )|*n* ≥ 0} ∩ *UnMarked*, by lemma 5.3, there exists *R*' ∈ {*αkRulen*(*P* )|*n* ≥ 0} such that

' '

*R* ⇒ *R* ⇒ *F*1 ∧ ··· ∧ *Fn* → *F*0, since *R*

'

⇒ *R* and *R* ∈ *SolvedForm*, then *R* ∈

*SolvedForm* and *R*' ∈ {*αkT n*(*P* )|*n* ≥ 0}, if *R*' ∈ *αkf ixpoint*(*P* ), then replace *R* by

' ' *k*

*k k n*

*R* in *T* , if *R* /∈ *α f ixpoint*(*P* ), by the definition *α f ixpoint*(*P* )= {*α T*

(*P* )|*n* ≥

0}∩ *UnMarked*, then there exists *R*'' ∈ *αkf ixpoint*(*P* ) such that *R*'' ⇒

''

'

*R* , replace

*R* by *R* in *T* . Repeat this procedure until all the rules in *f ixpoint*(*P* ) are replaced

by rules in *αkf ixpoint*(*P* ), then the derivation tree of *F* from *αkf ixpoint*(*P* ) is constructed, thus *F* is derivable from *αkf ixpoint*(*P* ).

Theorem 5.4 shows that if *F* is not derivable from *αkf ixpoint*(*P* ), then it is not derivable from *f ixpoint*(*P* ) also; If *F* is derivable from *αkf ixpoint*(*P* ), then it may or may not derivable from *f ixpoint*(*P* ). If *f ixpoint*(*P* ) does not terminate, it can be replaced by *αkf ixpoint*(*P* ) in the derivation algorithm as follows:

'

if ∃*R*

'

∈ *B, R*

⇒ *R*, then *derivablerec*(*R, B, P* )= ∅

else if *R* =→ *C*, then *derivablerec*(*R, B, P* )= {→ *C*}

' '

else *derivablerec*(*R, B, P* )= {*derivablerec*(*elimdup*(*R* •*R*)*,* {*R*}∪*B, P* )|*R* ∈

*αkf ixpoint*(*P* )}

By theorem 5.2, *αkf ixpoint*(*P* ) terminates, then by lemma 5.3, the derivation al- gorithm which uses *αkf ixpoint*(*P* ) terminates.

* 1. *Fixpoint reﬁnement*

Let *P* be a definite logic program and *αkf ixpoint*(*P* ) be the abstract solved-form fixpoint, the set of logic rules in *αkf ixpoint*(*P* ) which are not abstracted by *αk*, denoted by *UnAbstract*, is defined inductively as follows:

' ' '

1. Let *R* = *H* → *q*(*M* ) ∈ *P* ∩ *SolvedForm*, if *βk*(*M* )= *M* , then *αk*(*elimdup*-

(*R*)) ∈ *UnAbstract*;

1. If *R* ∈ *αkC*0(*P* ), then *R* ∈ *UnAbstract*;
2. If there exists *R*' ∈ *αkT n*(*P* ) ∩ *UnAbstract* and *R*'' ∈ *αkCn*(*P* ) ∩ *UnAbstract*

such that *R* = *elimdup*(*R*' ◦ *R*''), then *R* ∈ *UnAbstract*,

Definition 5.5 Let *P* be a definite logic program, define *αkpartialf ixpoint*(*P* )=

{*αkT n*(*P* )|*n* ≥ 0}∩ *UnAbstract*, *αkpartialf ixpoint*(*P* ) is called the partial solved- form fixpoint of *P* .

The partial solved-form fixpoint *αkpartialf ixpoint*(*P* ) of *P* consists of all the solved form logic rules whose derivation is not abstracted by *αk*.

If the derivation algorithm with *αkf ixpoint*(*P* ) shows the closed atom *F* is derivable from the definite logic program, we run the derivation algorithm with *αkpartialf ixpoint*(*P* ) as follows:

'

if ∃*R*

'

∈ *B, R*

⇒ *R*, then *derivablerec*(*R, B, P* )= ∅

else if *R* =→ *C*, then *derivablerec*(*R, B, P* )= → *C*

'

else *derivablerec*(*R, B, P* )= {*derivablerec*(*elimdup*(*R*

*αkpartialf ixpoint*(*P* )}

'

* *R*)*,R* ∪ *B, P* )|*R* ∈

By theorem 3.9, if *F* is derivable, the derivation witness can be constructed from *αkpartialf ixpoint*(*P* ) by the approach presented in [[4](#_bookmark7)]. If *F* is derivable from *αkf ixpoint*(*P* ), but *F* is not derivable from *αkpartialf ixpoint*(*P* ), we increase the threshold of the term depth bound *k*, compute *αk*+1*f ixpoint*(*P* ), and run the derivation algorithm with *αk*+1*f ixpoint*(*P* ) again.

The following theorem shows *αk*+*spartialf ixpoint*(*P* )(*s* ≥ 0) is a refinement of

*αkpartialf ixpoint*(*P* ).

Theorem 5.6 *Let P be a deﬁnite logic program, then for each s* ≥ 0*, αkpartialf ixpoint*(*P* ) ⊆ *αk*+*spartialf ixpoint*(*P* )*.*

Proof. For each *n* ≥ 0, we prove that *αkT n*(*P* ) ∩ *UnAbstract* ⊆ *αk*+*sT n*(*P* ) ∩

*UnAbstract* and*αkCn*(*P* ) ∩ *UnAbstract* ⊆ *αk*+*sCn*(*P* ) ∩ *UnAbstract*.

If *n* = 0, by the definition of *αk*, *αkT* 0(*P* ) ∩ *UnAbstract* = *αk*+*sT* 0(*P* ) ∩ *UnAbstract*, *αkC*0(*P* ) ∩ *UnAbstract* = *αk*+*sC*0(*P* ) ∩ *UnAbstract*, the conclusion holds.

Assume that the conclusion holds when *n* = *m* ≥ 0, in the case of *n* = *m* + 1, let *R* ∈ *αkT m*+1(*P* ) ∩ *UnAbstract*, then *R* ∈ *αkT m*(*P* ) ∩ *UnAbstract* or *R* ∈ *αkX Re*- *solutionm*+1(*P* ) ∩ *UnAbstract*. If *R* ∈ *αkT m*(*P* ) ∩ *UnAbstract*, by the induction assumption, *R* ∈ *αk*+*sT m*(*P* ) ∩ *UnAbstract*. If *R* ∈ *αkX Resolutionm*+1(*P* ) ∩

' '' '

*UnAbstract*, then *R* = *elimdup*(*R* ◦ *R* ), where *R* ∈ *αkT m*(*P* ) ∩ *UnAbstract*,

*R*'' ∈ *αkCm*(*P* ) ∩ *UnAbstract*, by the induction assumption, *R*' ∈ *αk*+*sT m*(*P* ) ∩ *UnAbstract*, *R*'' ∈ *αk*+*sCm*(*P* )∩*UnAbstract*, thus *R* ∈ *αk*+*sX Resolutionm*+1(*P* )∩ *UnAbstract*. By the fact *R* ∈ *αk*+*sT m*(*P* ) ∩ *UnAbstract* or *R* ∈ *αk*+*sX Resoluti*- *onm*+1(*P* ) ∩ *UnAbstract*, then *R* ∈ *αk*+*sT m*+1(*P* ) ∩ *UnAbstract*.

The fact that *αkCn*(*P* ) ∩ *UnAbstract* ⊆ *αk*+*sCn*(*P* ) ∩ *UnAbstract* can be proved in the similar way.

Since *f ixpoint*(*P* ) ⊆ ∪*k*≥0{*αkT n*(*P* )} and *f ixpoint*(*P* ) ⊆ *UnAbstract*, it is easy to see that *f ixpoint*(*P* ) ⊆ ∪*k*≥0*αkpartialf ixpoint*(*P* ), which means that the derivation witness can be constructed from *αkpartialf ixpoint*(*P* ) if the value of *k* is large enough.

Compared with the counterexample-driven abstraction refinement iteration anal- ysis 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.

# Experiments

To demonstrate the effectiveness of our approach,we have implemented these algo- rithms in our verifier prototype SPVT[[12](#_bookmark14)] for security protocols, and the security protocols in [[10](#_bookmark13)] 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 Π3 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 Π1 | true | true |
| Woo-Lam Authentication Protocol Π2 | true | true |
| Woo-Lam Authentication Protocol Π3 | 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 Π3 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 Π3

*host*(*kIS*)(*host*(*kAS*)) → *host*(*kBS*): *host*(*kAS*);

*host*(*kBS*) → *host*(*kIS*)(*host*(*kAS*)) : *N* [*i*1 *, host*(*kAS*)];

*B*

*host*(*kIS*)(*host*(*kAS*)) → *host*(*kBS*): *N* [*i*1 *, host*(*kAS*)];

*B*

*host*(*kBS*) → *host*(*kIS*)(*host*(*kSS*)) : *encrypt*(2*tuple*(*host*(*kAS*)*,N* [*i*1 *, host*(*kAS*)])*,*

*B*

*kBS*);

*host*(*kIS*)(*host*(*kSS*)) → *host*(*kBS*): *encrypt*(2*tuple*(*host*(*kAS*)*,N* [*i*1 *, host*(*kAS*)])*,*

*B*

*kBS*);

The above counterexample is the attack of Π3 described in [[10](#_bookmark13)].

# 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 com- bined together to solve the derivation problem more efficiently. The experimental results show the termination prediction algorithm is practical, and validate the ef- fectiveness 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.

1. Clark, J. A. and J. L. Jacob., *A survey of authentication protocol literature*, Technical Report 1.0 (1997).
2. 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 .
3. Mengjun Li, Zhoujun Li, HuoWang Chen, *SPVT: An efficient verification tool for security protocol*, Chinese Journal of Software 17 (2006), pp. 898–906.