Automatic Loop Summarization via Path Dependency Analysis

Xiaofei Xie et al.

July 28, 2020

Overview of the paper

- Introduction to loop summary and its application.
- Path dependency automaton and basic thought of using it for summarizing.
- Loop analysis and construction of PDA.
- Summarization of different kinds of loops.
- Experimental results.

Contributions

- Propose a path dependency automaton to model the relatioships betwen paths of loops.
- Propose classification defines what types of multi-path loops we can handle.
- Develop the tool PROTEUS for loop summarization.

Introduction to Loop Summary

Loop summary captures the relationship between the inputs and outputs of a loop as a set of symbolic constraints.
This means that the summary of a loop is a $\phi(X, X')$ s.t. ...
Applications:

- Loop bound analysis.
- Program verification. $\models \phi(X, X') \wedge \neg p$.
- Test case generation.

Path Dependency Automaton

Definition (PDA)

Given a loop with its CFG $\mathcal{G} = (L, E, l_{pre}, L_h, L_e)$, the path dependency automaton is a tuple $\mathcal{A} = (Q, \mathcal{L}, q_0, accept, T)$, where

- Q is a finite set of states.
- $\mathcal{L} : Q \to \Pi_{\mathcal{G}}$. We use $\mathcal{L}_q$ to represent $\mathcal{L}(q)$.
- $T = \{(q, q') \in Q \times Q \mid tail(\mathcal{L}_q) = head(\mathcal{L}_{q'}) \wedge \mathcal{L}_q \neq \mathcal{L}_{q'} \wedge (\exists k : (\bigwedge_{0 \leq i < k} \theta_{\mathcal{L}_q}[X_{\mathcal{L}_q{}^i}/X]) \wedge \theta_{\mathcal{L}_{q'}}[X_{\mathcal{L}_q{}^k}/X])$ is satisfiable $\}$ is a finite set of transitions.

Feasible Traces

Definition (Traces)

Given a loop with its PDA $\mathcal{A} = (Q, \mathcal{L}, q_0, accept, T)$, $E_{\mathcal{A}} = \{(q_0, \ldots, q_n) \mid q_n \in accept \wedge \exists k_0, \ldots, k_{n-1} : \bigwedge_{0 \leq i < n}((\bigwedge_{0 \leq j < k_i} \theta_{\mathcal{L}_{q_i^j}}[X_{\mathcal{L}_{q_i^j}}/X]) \wedge \theta_{\mathcal{L}_{q_{i+1}}}[X_{\mathcal{L}_{q_i^{k_i}}}/X])\}$ is satisfiable, where $X_{\mathcal{L}_{q_0^0}} = X \wedge \forall 0 < i \leq n : X_{\mathcal{L}_{q_i^0}} = X_{\mathcal{L}_{q_{i-1}^{k_{i-1}}}}\}$ is the set of feasible traces.

Loop Summarization

Definition (Loop summary)

Given a PDA $\mathcal{A}$ and a set of variables X, the summary of a trace $\tau \in E_{\mathcal{A}}$ is the constraints denoted as $\phi(X, X_\tau)$.
The disjunctive loop summary(DLS) of $\mathcal{A}$, denoted as $S_{\mathcal{A}}$ is $\bigvee_{\tau \in E_{\mathcal{A}}} \{\phi(X, X_\tau)\}$.
Problem: What if the set $E_{\mathcal{A}}$ is infinite?
We have to analysize the loop to find out:

- Whether $E_{\mathcal{A}}$ is infinite?
- If it is infinite, whether we can inductively generate the DLS?

In order to inductively generate DLS, we need to specify the variables, loop conditions and different types of path interleaving.

Patterns of Value Changes and Path Conditions

Definition (n-th Term of Variable)

Given a variable x and an iterative path σ, the n-th term of the variable x in the path σ is the value of x after n consecutive execution of σ.

Example

If the x is an arithmetic update, we can compute $x_n = x_0 + n * d$ where x_0 is the initial value of x and d is the common difference of variable x.

Criterias of Variables

Definition (IV, MIV, NIV)

- A variable x is an induction variable(IV) in a path σ if the value of x after n consecutive execution of σ can be computed with the initial value x_0 and the number of iteration n.
- Specifically, x is a monotonic induction variable(MIV) in a path if x is an IV and the value of x is strictly monotonic.
- Otherwise, we call the variable a non-induction variable(NIV).

Example

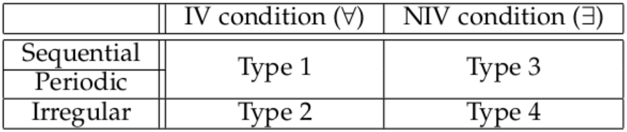
- $\texttt{for}(x <= n)\{x := x + 1;\}$ here x is a (M)IV.
- $\texttt{for}(x[0] <= n)\{x[1] := x[1] + 1;\}$. Here $x[0]$ is a NIV for it is a nondeterministic value in the array.

Patterns of Path Interleaving

- Sequential Execution. There is no cycle in each trace $\tau \in E_{\mathcal{A}}$.
- Periodic Execution. If all the cycles in every trace are periodic.
- Irregular Execution. Otherwise.

Classification of the Loop

Based on the different patterns of variables, conditions and path interleaving, a classification of the loop is given below.



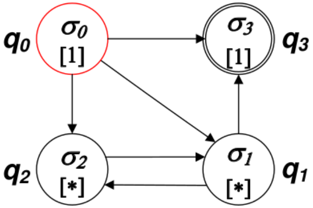
	IV condition (∀)	NIV condition (∃)
Sequential	Type 1	Type 3
Periodic		
Irregular	Type 2	Type 4

Examples of Different Types of Loops

Type1 loop:

```
int n=*;
int x=*;
int z=*;
while(x<n){
 if(z>x)
    x++;
 else
    z++;
}
```

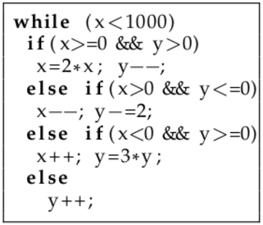
(a) Loop [9]



This example is a type 1 loop:

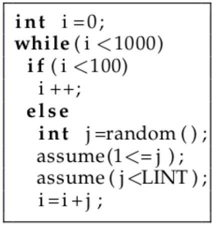
- The variables are inductive variables and so are the conditions.
- Loop in the PDA (q_1, q_2) is a periodic loop.

Examples of Different Types of Loops



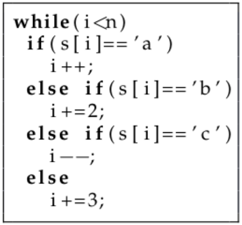
```
while  (x<1000)
 if(x>=0 && y>0)
  x=2*x;  y--;
 else  if(x>0 && y<=0)
  x--; y-=2;
 else  if(x<0 && y>=0)
  x++;  y=3*y;
 else
   y++;
```

(a) Type 2



```
int  i=0;
while(i <1000)
 if (i <100)
  i ++;
 else
  int  j=random();
  assume(1<=j );
  assume(j<LINT);
  i=i+j;
```

(b) Type 3

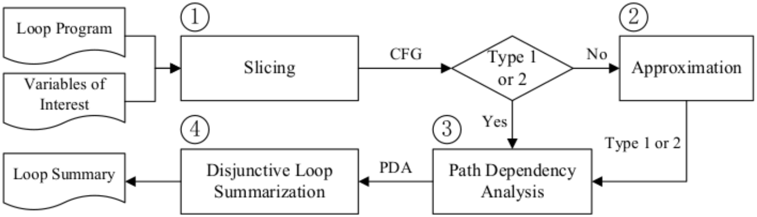


```
while(i<n)
 if(s[i]=='a')
   i++;
 else  if(s[i]=='b')
   i+=2;
 else  if(s[i]=='c')
   i--;
 else
   i+=3;
```

(c) Type 4

- Type 2: variables are IVs but the interleaving of all the branches are irregular.
- Type 3: variables include NIVs for j is dependent on `random()`, the execution is sequential from `if` branch to `else` branch.
- Type 4: conditions includes arrays and hence is NIVs. The array also induces the irregular transition of the branches.

Overview of the Tool: PROTEUS



As shown by the flow graph, the tool first slice the program to find the loops together with the variables of interest.
After obtaining the loops from slicing, we use above types definition to categoritze them into respective types.
Then construct PDA and do disjunctive loop summarization.

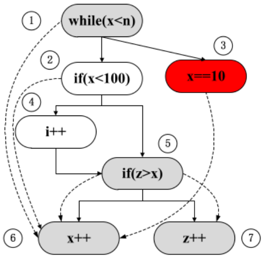
PDA Construction: Slicing

Loop slicing is based on the program dependence graph that combines control flow dependencies and data flow dependencies. It is targeting on reducing irrelevant paths in the constructed CFG and make summarization more efficient.

Example (Slicing)

```
int n=*;
int x=*;
int z=*;
int i=*;
while(x<n){
 if(x<100)
    i++;
 if(z>x)
    x++;
 else
    z++;
}
assert(x==10);
```

(a) Unsliced Loop



(b) PDG

Control flow dependency(solid arrow).Data flow dependecy(dotted arrow).

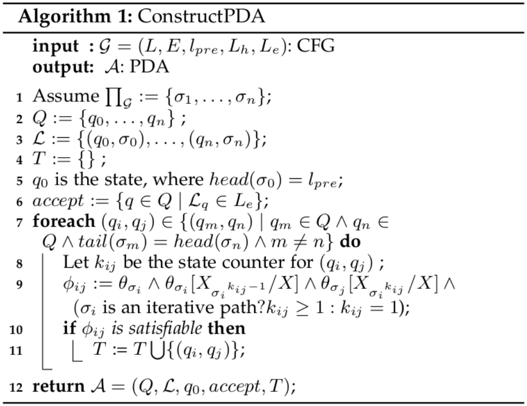
PDA Construction: Computing n-th Term in a Loop

To compute the n-th term computation, we consider the following two kinds of sequences:

- Basic sequence: Arithmetic sequence ($x_n = x_0 + c \times n$), geometric sequence ($x_n = x_0 \times c^n$) and constant sequence.
- Dependent sequence: if a sequence depends on another sequence. e.g. $x_n = x_{n-1} + y_n, x_n = x_{n-1} \times y_n$.

Based on these two kinds of sequence, we can compute value after n-th execution of the loop.

PDA Construction



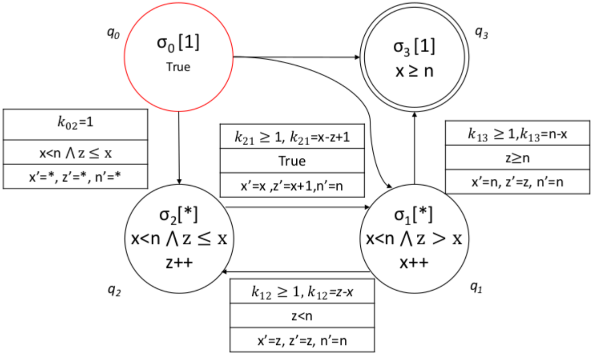
Algorithm 1: ConstructPDA

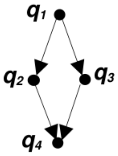
```
input  : $\mathcal{G} = (L, E, l_{pre}, L_h, L_e)$: CFG
output:  $\mathcal{A}$: PDA
Assume $\prod_{\mathcal{G}} := \{\sigma_1, \ldots, \sigma_n\}$;
$Q := \{q_0, \ldots, q_n\}$ ;
$\mathcal{L} := \{(q_0, \sigma_0), \ldots, (q_n, \sigma_n)\}$;
$T := \{\}$ ;
$q_0$ is the state, where $head(\sigma_0) = l_{pre}$;
$accept := \{q \in Q \mid \mathcal{L}_q \in L_e\}$;
foreach $(q_i, q_j) \in \{(q_m, q_n) \mid q_m \in Q \wedge q_n \in Q \wedge tail(\sigma_m) = head(\sigma_n) \wedge m \neq n\}$ do
    Let $k_{ij}$ be the state counter for $(q_i, q_j)$ ;
    $\phi_{ij} := \theta_{\sigma_i} \wedge \theta_{\sigma_i}[X_{\sigma_i}{}^{k_{ij}-1}/X] \wedge \theta_{\sigma_j}[X_{\sigma_i}{}^{k_{ij}}/X] \wedge$ ($\sigma_i$ is an iterative path?$k_{ij} \geq 1 : k_{ij} = 1$);
    if $\phi_{ij}$ is satisfiable then
        $T := T \bigcup \{(q_i, q_j)\}$;
return $\mathcal{A} = (Q, \mathcal{L}, q_0, accept, T)$;
```

Theorem

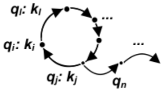
If ϕ_{ij} in Algorithm is satisfiable, then q_i can transit to q_j.

Examples and Structure of the Loops

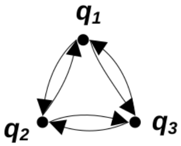




(a) Acyclic



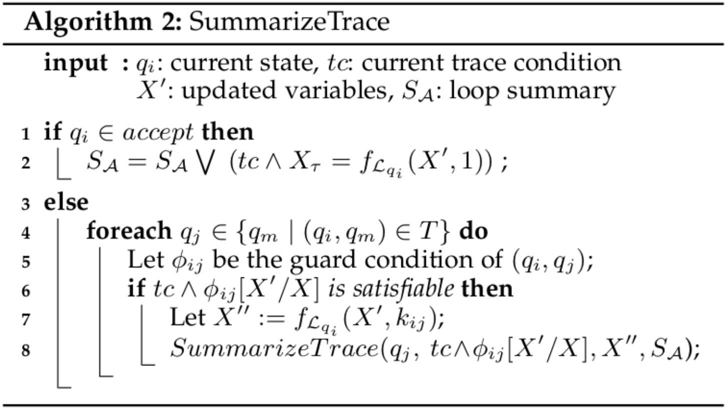
(b) Simple Cycle



(c) Connected Cycles

Summarization of Acyclic PDA

Summarization for type 1 acyclic PDA:
Varibles are IVs and all traces can be traversed.

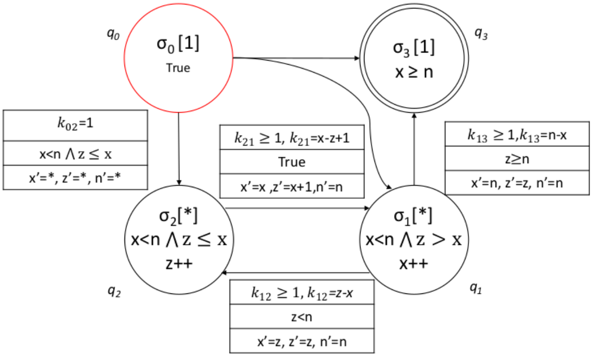


Algorithm 2: SummarizeTrace

```
input : $q_i$: current state, $tc$: current trace condition
        $X'$: updated variables, $S_{\mathcal{A}}$: loop summary
if $q_i \in accept$ then
    $S_{\mathcal{A}} = S_{\mathcal{A}} \bigvee (tc \wedge X_\tau = f_{\mathcal{L}_{q_i}}(X', 1))$ ;
else
    foreach $q_j \in \{q_m \mid (q_i, q_m) \in T\}$ do
        Let $\phi_{ij}$ be the guard condition of $(q_i, q_j)$;
        if $tc \wedge \phi_{ij}[X'/X]$ is satisfiable then
            Let $X'' := f_{\mathcal{L}_{q_i}}(X', k_{ij})$;
            $SummarizeTrace(q_j,\ tc \wedge \phi_{ij}[X'/X], X'', S_{\mathcal{A}})$;
```

Example of Acyclic PDA

If we ignore transition between q_1 and $q-2$.



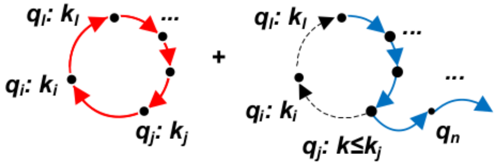
Summarization of PDA

Summarization for type 1 loops with simples cycles
Difficulty: the execution count of the loop is uncertain.
But if the variables are updated regularly in the cycle, we can reduce the cycle into one state.
Determine periodic cycle (IV conditions and IV state counters):
For a cycle $c = (q_l, q_{l+1}, \ldots, q_i)$, it can be regarded as q_c in the PDA.
We use k_n to represent the state counter from $q_n \in c$ to its successor in the cycle. Then the path condition can be represented as

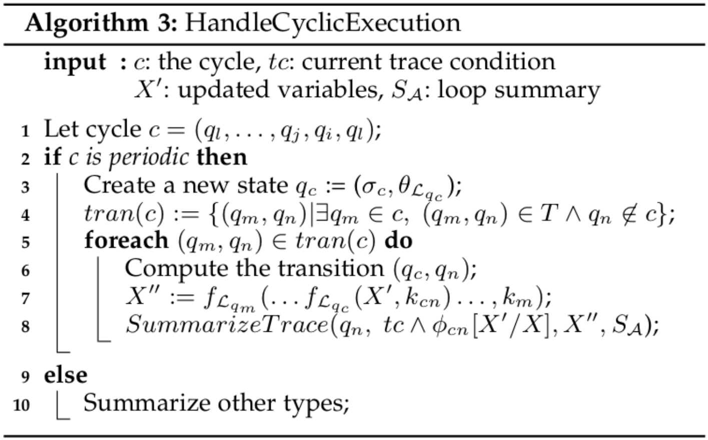
$$\theta_{\mathcal{L}_{q_c}} = \theta_{\mathcal{L}_{q_l}} \wedge \theta_{\mathcal{L}_{q_{l+1}}}[f_{\mathcal{L}_{q_l}}(X, k_l)/X] \wedge \cdots$$
$$\wedge \theta_{\mathcal{L}_{q_i}}[f_{\mathcal{L}_{q_i - 1}}(\ldots(f_{\mathcal{L}_{q_l}}(X, k_l))\ldots, k_{i-1})/X]$$

Summarization of PDA

Besides, a cycle may have several outports. We find the state that may exit the cycle and use the summarization of acyclic PDA to summarize the rest part.



Summarization of Periodic Cycles

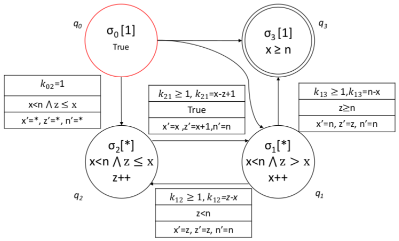


Algorithm 3: HandleCyclicExecution

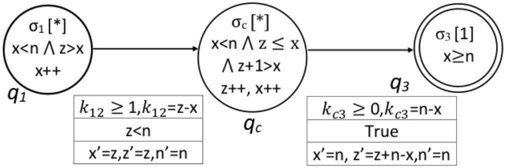
```
input : $c$: the cycle, $tc$: current trace condition
        $X'$: updated variables, $S_{\mathcal{A}}$: loop summary
Let cycle $c = (q_l, \dots, q_j, q_i, q_l)$;
if $c$ is periodic then
    Create a new state $q_c := (\sigma_c, \theta_{\mathcal{L}_{q_c}})$;
    $tran(c) := \{(q_m, q_n) | \exists q_m \in c,\ (q_m, q_n) \in T \wedge q_n \notin c\}$;
    foreach $(q_m, q_n) \in tran(c)$ do
        Compute the transition $(q_c, q_n)$;
        $X'' := f_{\mathcal{L}_{q_m}}(\dots f_{\mathcal{L}_{q_c}}(X', k_{cn}) \dots, k_m)$;
        $SummarizeTrace(q_n,\ tc \wedge \phi_{cn}[X'/X], X'', S_{\mathcal{A}})$;
else
    Summarize other types;
```

Here k_is are from the summary in the subtrace $(q_l, q_{l+1}, \dots, q_l)$.

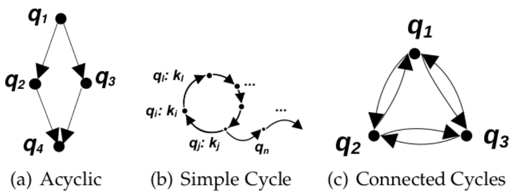
Example



$trace$	tc	X'
$(\boldsymbol{q_1}, q_2)$	$x < n \wedge z > x \wedge z < n$	$k_{12} = z\text{-}x,$ $x' = z,\ z' = z,\ n' = n$
$(q_1, \boldsymbol{q_2}, q_1)$	$x < n \wedge z > x \wedge z < n \wedge true$	$k_{12} = z\text{-}x,\ k_{21} = 1,$ $x' = z,\ z' = z{+}1,\ n' = n$
$(q_1, q_2, \boldsymbol{q_1}\ q_2)$	$x < n \wedge z > x \wedge z < n \wedge z + 1 < n$	$k_{12} = z - x = 1,\ k_{21} = 1,$ $x' = z{+}1,\ z' = z{+}1,\ n' = n$



Summarization for PDA with Connected Cycles



It it non-trivial to summarize connected cycles. Take above (c) as an example.
Idea: We can compute the dependency between cycles. If the execution of the cycles are sequential under certain precondition, we can summarize the loop by summarizing a sequence of simple cycles.

Soundness

Theorem

Given a Type 1 loop with the PDA $\mathcal{A}$ and the summary $\bigcup_{\tau \in E_{\mathcal{A}}} \{\phi(X, X_\tau)\}$, the summarized constraints $\phi(X, X_\tau)$ for each trace τ are satisfiable after each concrete execution of τ .

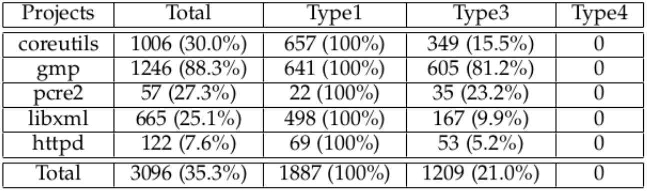
What if the loops are non-terminating?

Summarization of Type 2, 3 and 4 Loops

It is non-trivial to precisely summarize these types of loops because NIVs and NIV conditions may cause unpredictable value changes. Idea:

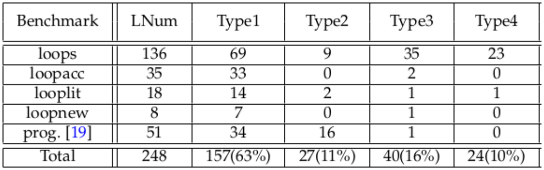
- interval approximation to compute the summary. e.g. $x := x + c$ where c is NIV. but we know $c \in [1, 5]$. Then we can approximate $x_n \in [x_0 + n, x_0 + 5n]$.
- Some specific NIV conditions which make the path only execute once. e.g. `if(bv){bv=!bv; x++;}`
- Some NIV conditions' values depend on the context and inputs but not the exection of the loop. We can use $true$ as the replacement.

Experimental Results: Loop Bound

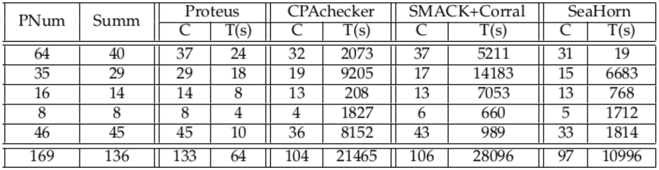


Projects	Total	Type1	Type3	Type4
coreutils	1006 (30.0%)	657 (100%)	349 (15.5%)	0
gmp	1246 (88.3%)	641 (100%)	605 (81.2%)	0
pcre2	57 (27.3%)	22 (100%)	35 (23.2%)	0
libxml	665 (25.1%)	498 (100%)	167 (9.9%)	0
httpd	122 (7.6%)	69 (100%)	53 (5.2%)	0
Total	3096 (35.3%)	1887 (100%)	1209 (21.0%)	0

Experimental Results: Program Verification



Benchmark	LNum	Type1	Type2	Type3	Type4
loops	136	69	9	35	23
loopacc	35	33	0	2	0
looplit	18	14	2	1	1
loopnew	8	7	0	1	0
prog. [19]	51	34	16	1	0
Total	248	157(63%)	27(11%)	40(16%)	24(10%)



PNum	Summ	Proteus		CPAchecker		SMACK+Corral		SeaHorn	
		C	T(s)	C	T(s)	C	T(s)	C	T(s)
64	40	37	24	32	2073	37	5211	31	19
35	29	29	18	19	9205	17	14183	15	6683
16	14	14	8	13	208	13	7053	13	768
8	8	8	4	4	1827	6	660	5	1712
46	45	45	10	36	8152	43	989	33	1814
169	136	133	64	104	21465	106	28096	97	10996

Experimental Results: Test Case Generation

Tool	functions_false.c	phases_false.c	multivar_false.c
KLEE	23 min	T/O	11.97 s
Pex	F	F	0.5 s
Proteus	0.06 s	0.18 s	0.05 s

Tool	functions_false.c	phases_false.c	multivar_false.c
KLEE	23 min	T/O	11.97 s
Pex	F	F	0.5 s
Proteus	0.06 s	0.18 s	0.05 s