Termination Analysis for Multi-Path Linear Loops

Hui Jin et. al from Tianjin Uni.

July 14, 2020

History of Research

Joint research with Xiaofei Xie of NTU.

- Xiaofei Xie, Bihuan Chen, Liang Zou, Shang-Wei Lin, Yang Liu, and Xiaohong Li. "Loopster: static loop termination analysis." In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 84-94. ACM, ESEC/FSE 2017
- Xiaofei Xie, Bihuan Chen, Liang Zou, Yang Liu, Wei Le, and Xiaohong Li. X. Xie, B. Chen, L. Zou, Y. Liu, W. Le and X. Li, "Automatic Loop Summarization via Path Dependency Analysis." In IEEE Transactions on Software Engineering, vol. 45, no. 6, pp. 537-557, TSE 2019

Contribution

- Extend the path dependency automaton(PDA) to analyze the linear loop and extract paths in CFG as states in the PDA to obtain the dependecy relationship between paths.
- Make a simple classification of cycles in PDA, and proposed methods to determie the termination of the cycle.
- Implement algorithm proposed by Tiwari to determine the termination of linear loops.

Loop Path of CFG

Definition (Control Flow Graph(CFG))

A control flow graph of a loop is a tuple $\mathcal{G} = (V, E, v_s, V_h, V_e, \imath)$, where V_h, V_e are the set of header blocks and exit blocks respectively. $\imath(e)$ is the branch conditio nof the edge $e \in E$.

Definition

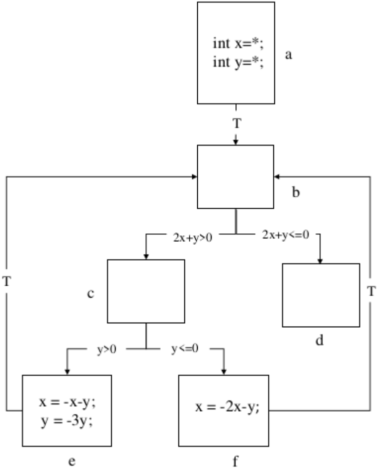
Given a CFG $\mathcal{G} = (V, E, v_s, V_h, V_e, \imath)$, the loop path σ is a finite sequence $v_0, v_1, \ldots, v_k$ of basic blocks where $v_0 \in V_h$ and $v_k \in V_h \cup V_e$ are the head and tail of σ. We call a path iterable path if $head(\sigma) = tail(\sigma)$.

- Path condition: θ_σ is the conjunction of the branch condition of each edge in the path.
- Value change of vars: $\mathcal{V}_\sigma$. $\theta(\sigma_i, \mathcal{V}^n_{\sigma_j}) \to \{true, false\}$.
- $pre(\mathcal{G})$: precondition of loop represent the possible valuation of vars.

Exmaple

```
int x = *;
int y = *;

while(2x+y>0){
    if(y>0){
        x = -x-y;
        y = -3y;
    }else{
        x = -2x+y;
    }
}
```



Loop paths: $\sigma_1 = (b, c, e, b), \sigma_2 = (b, c, f, b), \sigma_3 = (b, d)$

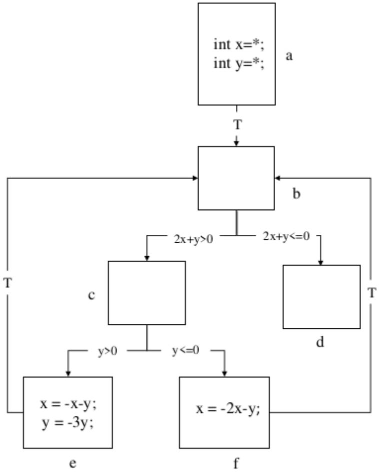
Path Dependency Automaton

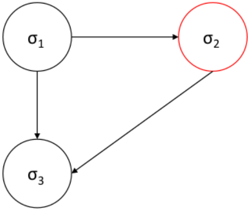
Definition (Path Dependency Automaton(PDA))

Given a loop with CFG $\mathcal{G}$, the path dependency automaton of this loop is $\mathcal{A} = (S, T, init, accept)$ where

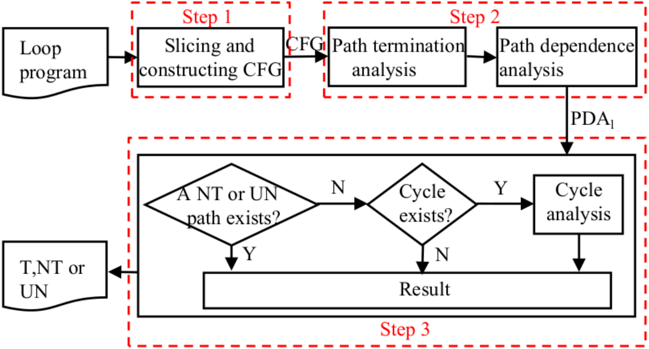
- S is the set of states. Each $\sigma \in S$ corresponds to a path in the loop.
- $T \subseteq S \times S$ is a set of transistions. $(\sigma_i, \sigma_j) \in T$ means that $\exists n > 0$ s.t. $\theta(\sigma_j, \mathcal{V}_{\sigma_i}^n) = true \wedge tail(\sigma_i) = head(\sigma_j)$.
- $init, accept$ are the set of initial states and accepting states respectively.

Example





Overview of MLTERM



The main part of termination analysis is step 3, where we first check if there is NT or UN path reachable. If not, do cycle analysis.

Path Termination Analysis

If the path is not iterative, it is terminating.
Otherwise the path represents a loop.

$$while(Bx > 0)\{x = Ax\}$$

Do termination analysis on this loop and output T if it is terminating.
The method for the terminating analysis is from [Tiwari].

[Tiwari] Termination of Linear Program.

Algorithm 1 PathTermAnalysis(σ,$pre(\sigma)$)

Input: σ, $pre(\sigma)$: the precondition of σ
Output: $\{T, NT, UN\}$

for all $\sigma \in S$ **do**
  if $head(\sigma) \neq tail(\sigma)$ **then**
    return T
  end if

  Construct $while\ Bx > b\ do\ x = Ax + c$ to analysis termination of σ denote as $term$
  if $term = T$ **then**
    return T
  else
    if $pre(\sigma) = true$ **then**
      return NT
    end if
    if $pre(\sigma)$ always satisfiable in σ **then**
      σ.condition.append($pre(\sigma)$)
      return PathTermAnalysis(σ,true)
    end if
    return UN
  end if
end for

Examples

```
int x=*;
int y=*;
while(x>0 && y>0){
    x = x-y;
    y = y-1;
}
```

(a) T

```
int x=*;
int y=*;
while(x>0){
  x = x-y;
  y = y-1;
}
```

(b) NT

```
int x=*;
int y=*;
if(y>0){
    while(x>0){
        x = x-y;
        y = y-1;
    }
}
```

(c) UN

(b) NT with witness $x = 2, y = 1$. (c) $pre(\mathcal{G})$ is not always satisfied.

Path Dependece Analysis

Target of path dependence anaylysis is to determine whether a loop path can transit to another.

There are two kinds of transitions:

- σ_i is terminating and its outdegree is 1.
- Check whether the formula $\exists n.\theta(\sigma_j, \mathcal{V}^n_{\sigma_i})$ is $true$.

Algorithm 2 ComputeTran($\mathcal{G}$)

Input: $\mathcal{G} : CFG$

for all $(\sigma_i, \sigma_j) \in \{(\sigma_m, \sigma_n) | \sigma_m \in S \wedge \sigma_n \in S \wedge tail(\sigma_m) = head(\sigma_n) \wedge m \neq n\}$ **do**
    if $\sigma_i$is termination $\wedge \sigma_i.outdegree = 1$ **then**
        $T = T \cup ((\sigma_i, \sigma_j))$
    else
        if $\theta(\sigma_j, \mathcal{V}^n_{\sigma_i}) == true$ **then**
            $T = T \cup ((\sigma_i, \sigma_j))$
        end if
    end if
end for

Cycle Analysis

Definition (Cycle in PDA)

Let $\mathcal{C} = \{\sigma_1, \sigma_2, \ldots, \sigma_{}\}$. $\mathcal{C}$ is a cycle of PDA if

- $\mathcal{C} \subset S$
- $\sigma_1, \sigma_2, \ldots, \sigma_n$ constitues an SCC in PDA.

Classify cycles into 2 categories:

- Type 1 cycle: All path in $\mathcal{C}$ are one-time paths.
- Type 2 cycle: Otherwise.

Cycle Analysis for Type 1

Idea: Since all paths are simple, we can simply merge them into a new path and do path termination analysis on the new path.

if $\forall \sigma_i \in \mathcal{C} \wedge \sigma_i.iterable = false$ **then**
    $\sigma = \text{merge}(\mathcal{C})$
    return PathTermAnalysis(σ,$pre(\sigma_1)$)
end if

Cycle Analysis for Type 2

assume $\mathcal{C}_{II}$ is a set that all iterable path in $\mathcal{C}$

for all $\sigma \in \mathcal{C}$ **do**
    if $head(\sigma) == tail(\sigma)$ **then**
        $\mathcal{C}_{II}$.append(σ)
    end if
end for
for all $\sigma \in \mathcal{C}_{II}$ **do**
We use Θ_σ to denote the conditional of σ
    if $\exists \theta_i \in \Theta_\sigma$ only change in σ **then**
        $\mathcal{C}$.del(σ)
        $\mathcal{C}_{II}$.del(σ)
    end if
end for
if $\mathcal{C}$ is not a cycle **then**
    return T
end if

if $\mathcal{C}_{II}$.empty() **then**
    return CycleAnalysis($\mathcal{C}$)
end if
return UN

Experimental Results

TABLE I
EXPERIMENTAL RESULTS

	MLTerm	Ultimate	CPAchecker	2LS
CR	67	72	61	61
CUN	10	5	14	16
COT	0	0	2	0
TT	149.1	668.6	6147.9	1582.6

Benchmark: SV-COMP 2020
CR: num of right result.
CUN: num of unknown.
COT: num of timeout.
TT: total time.

Experimental Results