*论文题目：Proteus: Computing Disjunctive Loop Summary via Path Dependency Analysis

*作者：Xiaofei Xie （谢肖飞） Xiaohong Li （李晓红） 天津大学

Bihuan Chen （陈碧欢） Yang Liu （刘杨） 南洋理工大学

Wei Le Iowa State University

*单位：天津大学

联系方式：xiexiaofei@tju.edu.cn,

地址: 天津市津南区海河教育园区天津大学计算机科学与技术学院

*文章发表信息：

论文集名称：Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering

出版社：ACM New York, NY, USA

出版时间：2016 年 11 月

*原文链接地址：http://dl.acm.org/citation.cfm?id=2950340

*正文：

【FSE 论文介绍】Loop Summarization

2016-11-28 谢肖飞 软件工程研究与实践 软件工程研究与实践

SE-China 中国计算机学会软件工程专委与 ACM-CSOFT 主办的学术研究与业界实践交流平台。

研究背景和动机

在程序分析中，循环（Loop）的处理与分析是一个非常重要而且很有挑战的任务。例如，在符号执行（Symbolic Execution）中，循环的不断展开会导致程序路径数量指数级地增长。因此，符号执行通常会陷入不停的循环展开，而不能覆盖到新的程序分支上，最终影响了测试用例生成或者缺陷检测的效率。

```
unsigned int x = __VERIFIER_nondet_uint();
unsigned int y = x + 1;
while (x < 1024) {
   x++;
   y++;
}
if(y>=1024)
   // test case
```

对于上面这个简单的循环，如果我们要生成一个可以覆盖第八行的测试用例，KLEE 需要 10 秒，而 PEX 会报异常。这是因为该程序包含一个需要展开多次的深循环，使得符号执行产生了大量的分支。如果我们继续增大循环的深度，那么 KLEE 与 PEX 的性能会进一步地降低。例如当我们将循环深度增大到 0x0FFFFFFF，KLEE 将需要超过 20 分钟的时间。

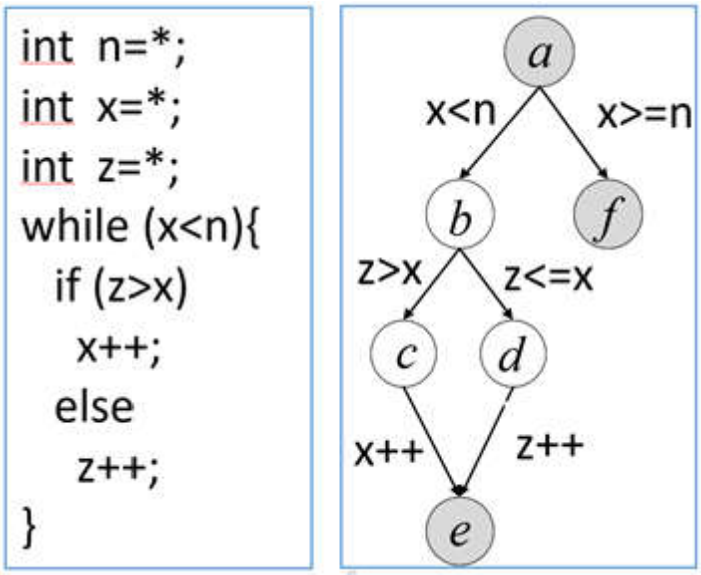
在程序验证（Program Verification）中，对于循环通常需要求得循环不变量（Invariant）来验证某个性质。然而，找到一个足够强的不变量通常需要花费很多的时间去不断展开循环以求得不动点。有界模型检验（Bounded Model Checking）通过给定一个循环展开的最大次数来处理循环，因此 BMC 通常用来发现漏洞而不能用来正确地验证程序性质。

```
unsigned int n = __VERIFIER_nondet_uint();
unsigned int x=n, y=0;
while(x>0)   {
   x--;
   y++;
}
__VERIFIER_assert(y==n);
```

上面的程序是 SVCOMP 2016 中的一个基准程序。从比赛结果中，我们发现大部分的工具都不能正确地验证第七行的性质。具体而言，对于比赛中获得循环分析前三名的工具，CPAchecker 的分析结果为 Unknown，2LS 和 ULTIMATE 结果都为超时（15 分钟）。

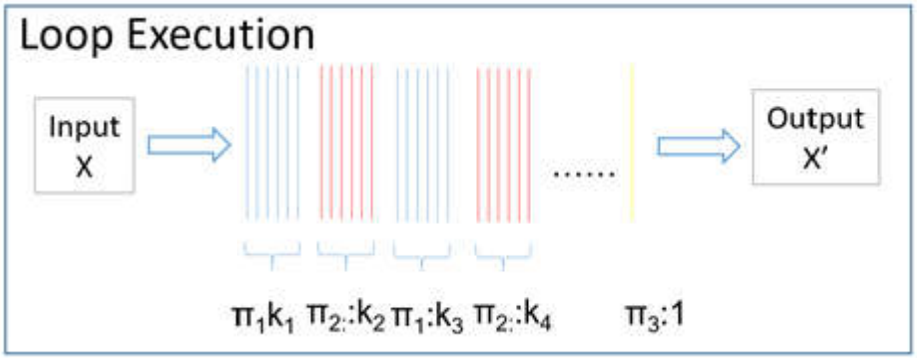
如果我们可以静态地对循环进行概括（Summarization），那么我们就可以用循环总结（Loop Summary）来替换循环。这样就可以避免循环的不断展开，从而极大地提高程序分析的效率。例如，针对上面的两个程序，第一个循环的概括结果是(x>1024^y==x+1) v (x==1024 ^ y==1025)，而第二个循环的概括结果是 x==0^y==n。应用以上的循环总结，我们可以花 0.05s 对第一个程序生成目标测试用例（时间不依赖于循环深度），用 0.04s 对第二个程序进行验证。一些研究学者已经提出了一些循环概括的方法，但是这些方法对于多路径的循环比较受限。因此，本文提出了一种对于多路径循环分析与概括的方法，并将其应用于多个程序分析应用中。

循环概括的理解



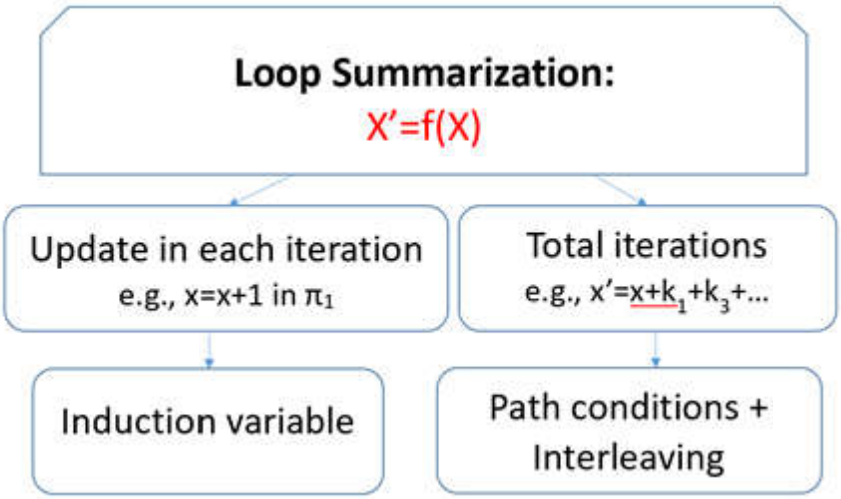
图一：一个循环以及对应的流图

循环概括是一个不可判定的问题。因此，我们首先对循环进行了深入的分析从而发现哪些循环可以被概括，哪些循环比较困难。图一展示了一个循环以及它所对应的流图（flowgraph）。从流图上，我们可以看到该循环具有三条路径：π1=<a,b,c,e>，π2=<a,b,d,e>和 π3=<a,f> 。路径条件（path condition，简写为 pc）是该路径上多个条件的合取。例如 π1 的 pc 是 x<n ^ z>x。循环执行就是多条路径的交错执行；在每次迭代过程中，如果某一条路径的 pc 是可满足的，那么该路径将在本次迭代中执行。



图二： 循环的迭代过程

图二展示了循环的执行过程。给定一些初始值，循环的执行就是这些路径（π1，π2 和 π3）的交错执行（interleaving），最后输出结果。图三我们分析了需要哪些知识来对循环进行概括 **。循环概括就是在不真正执行循环的前提下，求得循环执行后的结果（后置条件）：X'=f(X)**。



图三： 循环概括分析

直观地来讲，如果我们知道了每条路径上**变量的变化**（在 π1 上 x'=x+1）以及**路径执行的总数**（k1+k3+…），我们就可以求得变量的最终结果（x'=x+k1+k3+…）。

1. 对于变量的变化，如果该变量在每条路径上的变化为常量，那么我们定义该变量为可归纳变量（induction variable），否则我们称之为不可归纳变量(non-induction variable)。
2. 某个路径执行的总数是由路径条件与交错执行方式所决定的。例如，对于图二中的 π1，在第一次执行时，它的 pc 是可满足的。π1 将执行 k1 次，直到 π1 的 pc 不满足；之后循环将执行路径 π2，并执行 k2 次，然后又回到 π1。我们可以看出**路径条件决定了该路径每次执**

行多少次，而交错执行的方式决定了该路径执行的频率。这两个因素决定了某条路径执行的总数。

基于以上分析，我们将循环分为了四类，如下表所示。其中，第一行为条件（也就是每个分支上的条件，例如 z>x）。如果一个条件所包含的每个变量都是可归纳变量，那么该条件被称之为 IV Condition，否则该条件时 NIV Condition。第一列为交错方式。如果路径之间的执行是顺序的（例如 a*->b*），我们称为顺序执行。如果路径执行中包含环（例如 a*->b*->a*），并且这些环都是有周期的环（例如(aabb)(aabb) (aabb)），我们称之为有周期的执行。否则，我们称为无规律的执行。

Condition / Interleaving	IV Condition (∀)	NIV Condition (∃)
Sequential	Type 1	Type 3
Periodic		
Irregular	Type 2	Type 4

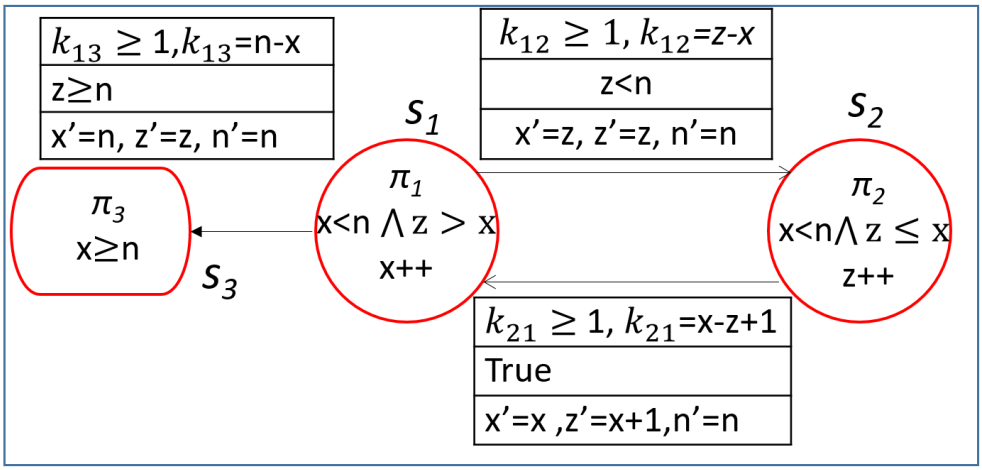
该循环分类帮助我们理解了分析与概括循环的难易程度。在本文中，我们主要对类型一（Type 1）的循环进行概括。Type 1 循环的条件都是 IV Condition，并且执行是顺序或者有周期性的。这使得我们可以静态地分析每条路径执行的次数。对于其它类型的循环，它们的循环概括比较困难，我们也提出了一些基于策略的方法来对这些循环进行近似与抽象。

循环建模

我们提供了一个路径依赖自动机（path dependency automaton，简写为 PDA）来对类型一的循环进行建模。一个 PDA 是一个四元组 A= (S ,I, F, T):

- S = {s | s 对应流图 G 中的某条路径 π },
- I = {s ∈S | Pre(G) ^ θ_π 可满足}
- T = {(s_1,s_2)∈S x S | 存在 i 使得 Pos(π_1^i) ^ θπ_2 可满足}
- F = {s ∈ S | 不存在任意一个 s’ ∈ S 使得 (s,s’) ∈T }

其中 S 是状态集合，每个状态对应于流图中的一条路径；I 是初始状态集合；T 是状态转移集合；而 F 是接受状态集合。θ_π 表示 π 的路径条件，Pre(G)表示循环的前置条件（Precondition），Pos(π^i) 表示路径 π 执行了 i 次后的后置条件（Postcondition）。



图四：图一循环所对应的 PDA

图四展示了图一循环所对应的 PDA。因为循环中有三条路径，所以该 PDA 包含三个状态。因为循环的输入值可以是任意值，所以三条路径都可能被先执行，即三个路径都可以是初始状态。π3 是接受状态。状态转移（s1,s2）表示在 π1 执行多次后，π2 将被执行。

接下来我们将介绍，给定任意的一对状态（s1,s2），s1 是否可以转移到 s2？直观地来讲，s1 可以转移到 s2 意味着：**初始时 π1 的 pc 是可满足的，在执行了一定次数后，π1 的 pc 不可满足而 π2 的 pc 可满足**。因此，我们引入符号变量 k12 表示 π1 执行 k12 次后 π2 将被执行，那么**在 k12-1 次后，π1 的 pc 是可满足的；而在 k12 次后，π2 的 pc 变为可满足的**。

例如，在例子中：

- 在 k12-1 后，π1 的 pc 变为 pc1=（x'<n' ）^（z'>x'） =（x+k12-1 < n） ^（x+k12-1<z）
- 在 k12 后，π2 的 pc 变为 pc2= （x'<n' ）^（z'<=x' ）=（x+k12<n） ^（z<=x+k12）

pc1^pc2 则是（s1,s2）的跳转条件（guard condition）, 如果跳转条件是可满足的，那么 s1 可以转移到 s2。通过约减我们可以得到 pc1^pc2= z<n，k12=z-x。利用 k12 我们可以求出该条转移的后置条件，例如 x'=x+k12=z。图三中每条转移上面有一个包含三行的表格，其中第一行是关于起始状态执行次数 k 的约束，第二行是该转移的跳转条件，而第三行是转移的后置条件。

循环概括

首先我们定义 PDA 上的一个轨迹（trace）是从初始状态到接受状态的一条序列。**一个 trace 的语义就是该循环的一次完整执行**。循环概括则是对 PDA 上的每条 trace 进行概括。例如，图五展示了 PDA 上的一条 trace。给定一个初始值 X0，首先我们可以求得在转移（a,b）后的结果 X1，然后用 X1 作为（b,c）的初始值求得之后的结果为 X2，最终求得 trace 总结 Xn。给定不同的初始值，循环可能对应不同的 trace，因此我们对每一条可能的 trace 进行概括，最终的循环总结就是多个 trace 总结的合取。

图五：trace 概括过程

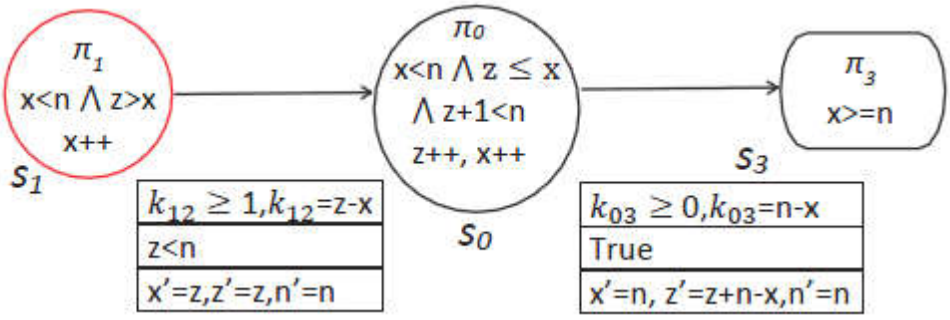
图四一共包含了四条可能的 trace，其对应着循环的四种可能的执行方式。例如 s3 表示该循环没有被执行。

1. s_3
2. (s_1,s_3)
3. $(s_1, (s_2,s_1)^+, s_3)$
4. $(s_2, (s_1,s_2)^+, s_1, s_3)$

前两条 trace 不包含环，因此，我们可以直接求得 trace 总结为：

- **(s_3)**: { x'=x, z'=z, n'=n)
- **(s_1,s_3)**: { k_{13}=n-x, x'=n'=n, z'=z}

第三条 trace 是 $(s_1, (s_2,s_1)^+, s_3)$，由于包含一个环 (s_2,s_1)，该环的出现使得 trace 的长度不是确定的。对于 Type 1 循环，由于其环都是周期性的，因此这里我们将每个周期环合并成一个状态。例如 (s_2,s_1) 是一个周期环（周期环的检测参见论文），在每次该环的执行过程中 s_2 和 s_1 都各执行一次。图六展示了将 (s_2,s_1) 合并成一个状态 s0 后的 trace。直观上 s0 执行一次等价于 s2 和 s1 各执行一次。然后我们可以求 s0 到 s3 的转移，这样便将带有环的 trace 变为了顺序的执行（也就是无环的 trace），再按上面的方法对其进行概括。



图六：周期环合成后的轨迹

基于以上的分析，后两条包含环的 trace 可以概括为：

- **$(s_1, (s_2,s_1)^+, s_3)$** => (s_1,s_0, s_3) : { k_{12}=z-x, k_{03}=n-z,x'=z'=n'=n}
- **$(s_2, (s_1,s_2)^+, s_1, s_3)$**: { k_{21}=x-z+1, k_{01}=n-x-1,k_{13}=1, x'=z'=n'=n}

对于其他类型循环的概括，我们用了一些策略来转换循环。例如，对于非归纳变量，我们用区间近似的方法来求得其变化量[min,max]；对于无规律执行的循环，我们用 path counter 来表示每条路径在循环执行完后执行次数的总和，细节可以参见论文。

总结与展望

我们提出了一个对循环分析及概括的方法。循环概括可以应用到许多程序分析应用中。例如，在循环执行次数上界的分析（loop bound）中，通过将每条 trace 上的变量 k 累加，然后求得它的最大值即是循环的 bound。在符号执行中，我们可以不展开循环，而将循环总结合取到当前的路径条件（path condition）中，以快速覆盖到更多的分支。在程序验证中，我们同样也可以不展开循环或者不求不变量，而用循环总结来验证其性质是否可以被满足。另外，PDA 还可以被潜在地应用于终止性分析（termination analysis）、性能分析等其他应用中。例如，一个循环是否可停就是 PDA 上的每条可能的 trace 的长度是否都是有限的。循环概括是一个不可判定的问题，因此目前我们的方法局限于类型 1 循环的概括。将来我们将尝试应用现有的一些技术（例如，抽象解释，ranking function）去帮助概括其他类型的循环。

作者介绍

本篇论文《Proteus: Computing Disjunctive Loop Summary via Path Dependency Analysis》 发表于 FSE16，作者包括谢肖飞（天津大学博士生）、陈碧欢（南洋理工大学博士后）、刘杨（南洋理工大学助理教授）、Wei Le（Iowa State University 助理教授）、以及李晓红（天津大学教授）。

作者简介：谢肖飞（1989-03），男，山西运城人，博士生，主要研究领域是软件工程，程序分析等。

文章由 CCF 软件工程专业委员会白颖教授推荐。