《软件分析与验证》

程序的自动机表示

贺飞
清华大学软件学院

2024 年 5 月 24 日

在前面的学习中我们了解到：

- 给定合适的循环不变式，程序验证问题可以被归结为对应验证条件的可满足性判定问题
- 如何找到合适的循环不变式是一个非常困难的问题

问题：有没有可能不提供循环不变式也能证明程序正确性？

答案：是肯定的，对应的方法称自动化程序验证（Automatic Program Verification）；这类方法中用到了较多模型检验的技术，常常也称软件模型检验（Software Model Checking）

在此之前，我们先对 IMP 语言进一步扩展，并引入程序的另一种形式化表示——基于自动机的表示

目录

IMP 语言扩展

非确定性

程序中的非确定性:

```
scanf("%d", c);           // 不确定的用户输入
x = libxxx(...);          // 未知的库函数
y = read(shared_var);     // 可能被其他进程修改了的共享内存
```

能否提供一种机制对上述情况进行方便的建模?

定义

IMP 的抽象语法递归定义如下：

$$
\begin{aligned}
e \in \mathit{AExp} &::= c \in \mathbb{Z} \mid x \in \mathit{Var} \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid a[e] \\
p \in \mathit{BExp} &::= \textbf{true} \mid \textbf{false} \mid e_1 = e_2 \mid e_1 \le e_2 \mid \neg p \mid p_1 \wedge p_2 \\
st \in \mathit{Stmt} &::= \textbf{skip} \mid x := e \mid a[e_1] := e_2 \\
&\quad \mid \textbf{assert}\ p \mid \textbf{assume}\ p \\
&\quad \mid \textcolor{red}{\textbf{havoc}\ x} \\
&\quad \mid st_1; st_2 \\
&\quad \mid \textbf{if}\ (p)\ \{st_1\}\ \textbf{else}\ \{st_2\} \\
&\quad \mid \textbf{while}\ (p)\ \{st\} \\
&\quad \mid m(e_1, \ldots, e_n)
\end{aligned}
$$

在 IMP 中扩展 havoc 语句

在 IMP 中引入 **havoc** 语句，用于建模非确定性

havoc 语句的语义：

$$[\![\textbf{havoc}\ x]\!] = \{(s, s') \mid s' = s[x \mapsto \star]\}$$

其中 $\star$ 代表任意值，**havoc** x 即给 x 赋任意值

havoc 语句的最弱前置条件：

$$wp(\textbf{havoc}\ x, \psi) = \forall x.\ \psi$$

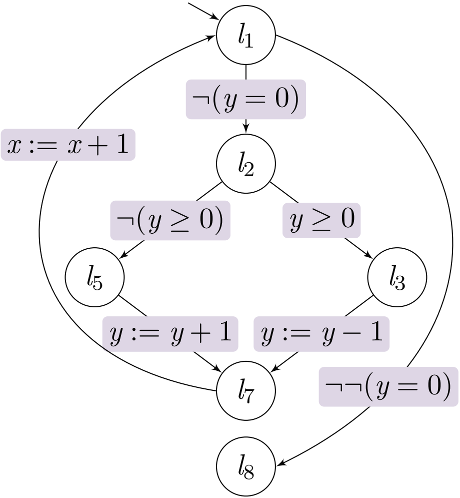
控制流自动机

控制流自动机举例

程序 P_{xy} 的代码：

```
while (!(y == 0)) {
    if (y >= 0) {
        y = y - 1;
    } else {
        y = y + 1;
    }
    x = x + 1;
}
```

程序 P_{xy} 的控制流自动机：



控制流自动机 vs. 控制流图

定义 (控制流自动机)

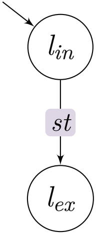
控制流自动机 (control flow automaton, CFA) 是一个四元组 $G = (Loc, \Delta, l_{in}, l_{ex})$，其中

- Loc 是程序位置的有限集合
- Δ 是一个由三元组 (l, st, l') 组成的集合，其中 $l, l' \in Loc$，st 为以下基本语句之一：赋值语句，数组赋值语句，Havoc 语句，或 Assume 语句
- $l_{in} \in Loc$ 为初始位置
- $l_{ex} \in Loc$ 为退出位置

简单语句的控制流自动机

令 st 为以下基本语句之一:

- 赋值语句
- 数组赋值语句
- Havoc 语句
- Assume 语句

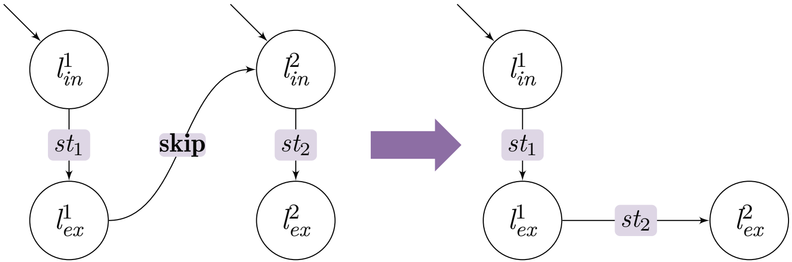


语句 st 的控制流自动机 $G = (Loc, \Delta, l_{in}, l_{ex})$,
其中:

- $Loc = \{l_{in}, l_{ex}\}$
- $\Delta = \{(l_{in}, st, l_{ex})\}$
- $l_{in} \neq l_{ex}$

顺序组合语句的控制流自动机

给定 st_1 和 st_2 的控制流自动机，求 $st_1; st_2$ 的控制流自动机



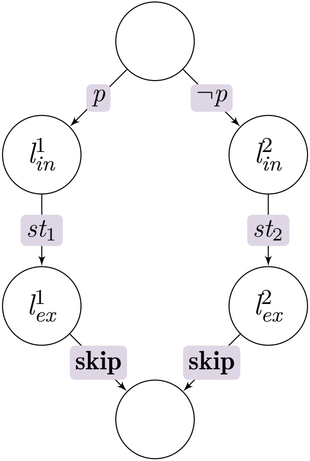
令 $G^1 = (Loc^1, \Delta^1, l^1_{in}, l^1_{ex})$，$G^2 = (Loc^2, \Delta^2, l^2_{in}, l^2_{ex})$ 分别为语句 st_1, st_2 的控制流自动机，并且 $Loc^1 \cap Loc^2 = \emptyset$

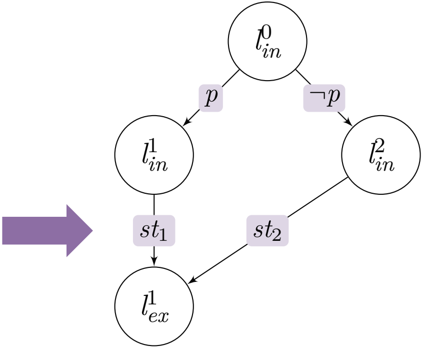
将 G^2 中的 l^2_{in} 替代为 l^1_{ex}，得到 $G^3 = (Loc^3, \Delta^3, l^3_{in}, l^3_{ex})$，其中

$$
\begin{aligned}
Loc^3 &= (Loc^2 \setminus \{l^2_{in}\}) \cup l^1_{ex} \\
\Delta^3 &= \{(l^1_{ex}, st, l') \mid (l^2_{in}, st, l') \in \Delta^2\} \\
&\qquad \cup \{(l, st, l^1_{ex}) \mid (l, st, l^2_{in}) \in \Delta^2\} \\
&\qquad \cup \{(l, st, l') \mid (l, st, l') \in \Delta^2 \text{ s.t. } l \neq l^2_{in} \text{ and } l' \neq l^2_{in}\} \\
l^3_{in} &= l^1_{ex},\ l^3_{ex} = l^2_{ex}
\end{aligned}
$$

$G = (Loc^1 \cup Loc^3, \Delta^1 \cup \Delta^3, l^1_{in}, l^3_{ex})$ 为顺序组合语句 $st_1; st_2$ 的控制流自动机

给定 st_1 和 st_2 的控制流自动机，求 **if** (p) $\{st_1\}$ **else** $\{st_2\}$ 的控制流自动机





设 $G^1 = (Loc^1, \Delta^1, l^1_{in}, l^1_{ex})$，$G^2 = (Loc^2, \Delta^2, l^2_{in}, l^2_{ex})$ 分别为语句 st_1 和 st_2 的控制流自动机，并且 $Loc^1 \cap Loc^2 = \emptyset$。

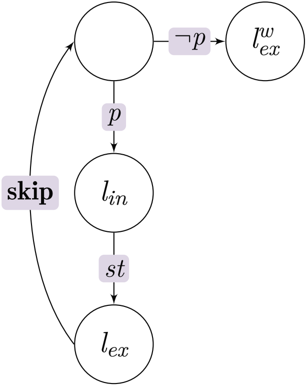
练习

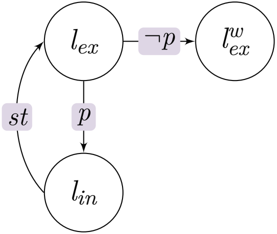
给出条件语句控制流自动机的形式化定义：

$$\textbf{if}\ (p)\ \{st_1\}\ \textbf{else}\ \{st_2\}$$

循环语句的控制流自动机

给定 st 的控制流自动机，求 **while** (p) $\{st\}$ 的控制流自动机





设 $G = (Loc, \Delta, l_{in}, l_{ex})$ 为语句 st 的控制流自动机，且 $l_{ex}^w \notin Loc$，
则 **while** (p) $\{st\}$ 的控制流自动机为：

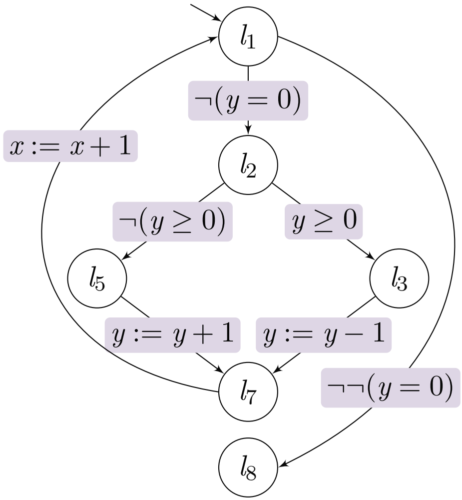
$G^w = (Loc^w, \Delta^w, l_{in}^w, l_{ex}^w)$，其中

- $Loc^w = Loc \cup \{l_{ex}^w\}$
- $\Delta^w = \Delta \cup \{(l_{ex}, \textbf{assume } p, l_{in}), (l_{ex}, \textbf{assume } \neg p, l_{ex}^w)\}$
- $l_{in}^w = l_{ex}$

程序 P_{xy} 的代码：

```
while (!(y == 0)) {
    if (y >= 0) {
        y = y - 1;
    } else {
        y = y + 1;
    }
    x = x + 1;
}
```

程序 P_{xy} 的控制流自动机：



控制流自动机：

- 控制流自动机为程序提供了一种图形化表示
- 控制流自动机刻画了程序的语法结构
- 控制流自动机提供程序当前所处位置的信息，但未提供有关程序变量取值的信息

回忆：程序状态代表程序变量的赋值

能否将控制流自动机与状态相结合来刻画程序的行为?

格局和执行

设 $G=(Loc,\Delta,l_{in},l_{ex})$ 是程序 P 的控制流自动机。

定义

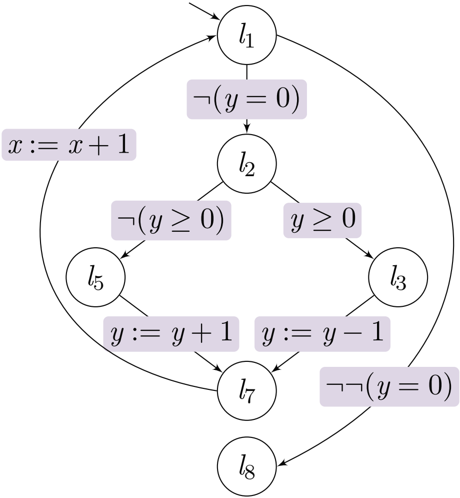
格局 (configuration) 是一个有序对 (l,s)，其中 $l\in Loc$ 是程序位置，$s\in State$ 是程序状态（即对程序所有变量的一组赋值）。

定义

执行（execution）是满足下列条件的格局序列 $(l_0,s_0),\ldots,(l_n,s_n)$：存在一个语句序列 $st_1,\ldots,st_n$，使得对任意 $i\in\{0,\ldots,n-1\}$，下列条件都成立：

- $(l_i,st_{i+1},l_{i+1})\in\Delta$，且
- $(s_i,s_{i+1})\in[\![st_{i+1}]\!]$

程序 P_{xy} 的控制流自动机：



程序 P_{xy} 的一次执行：

$$(l_3, \{x \mapsto 42, y \mapsto 23\})$$
$$(l_7, \{x \mapsto 42, y \mapsto 22\})$$
$$(l_1, \{x \mapsto 43, y \mapsto 22\})$$
$$(l_2, \{x \mapsto 43, y \mapsto 22\})$$
$$(l_3, \{x \mapsto 43, y \mapsto 22\})$$
$$(l_7, \{x \mapsto 43, y \mapsto 21\})$$

注意：在我们的定义中，不要求执行一定从初始位置开始，也不要求执行一定在退出位置结束。

定义

设 $(\varphi_{pre}, \varphi_{post})$ 为程序 P 的前置-后置条件对，(l, s) 为 P 的格局

- 若 $l = l_{in}$ 且 $s \models \varphi_{pre}$，称 (l, s) 为 P 的初始格局
- 若 $l = l_{ex}$ 且 $s \not\models \varphi_{post}$，称 (l, s) 为 P 的错误格局

定理 (基于执行的正确性证明)

程序 P 满足前置-后置条件对 $(\varphi_{pre}, \varphi_{post})$，当且仅当 P 中不存在从初始格局 (l_0, s_0) 到错误格局 (l_n, s_n) 的执行。

可达格局与可达图

例 1：程序执行

下面的程序是否满足给定的前置-后置条件对?

```
while(x % 1337 != 0){
  if(y % 37 != 0){
    x = (3 * x) % (256 * 256);
    y = (-2 * y + 1) % (256 * 256);
  } else {
    tmp = x;
    x := y;
    y := tmp;
  }
}
```

$$\varphi_{pre} : x = 1 \land y = 1$$

$$\varphi_{post} : y \leq 31337$$

如何验证?

1. 基于霍尔证明系统：要求找到合适的循环不变式
2. 基于程序执行：枚举程序的所有执行，看能否到达错误格局
   注意： 该规约的前置条件确定了唯一的初始状态、唯一的执行

例 1：程序执行

```c
#include <stdio.h>

int main(void) {
  unsigned short x = 1;
  unsigned short y = 1;
  while (x % 1337 != 0) {
    if (y % 37 != 0) {
      x = (3 * x); y = (-2 * y + 1);
    } else {
      unsigned short tmp = x;
      x = y; y = tmp;
    }
    printf("value of x is %d\n", x);
    printf("value of y is %d\n", y);
  }
  return 0;
}
```

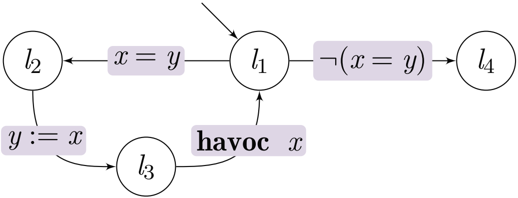
- 以 C 语言实现该程序；输出 8616 行
- 第一行为：$x = 3, y = 65535$
- 最后一行为：$x = 33425, y = 43691$
- 结论：从满足前置条件的状态出发执行程序，程序终止并且终止时的状态不满足给定的后置条件，即程序不满足给定的前置-后置条件对。

下面的 P_{xor} 程序（其中 x, y 为布尔变量）是否满足给定的前置-后置条件对？

$$\varphi_{pre} : x$$

$$\varphi_{post} : x \to \neg y$$

```
while(x == y){
  y = x;
  havoc  x;
}
```



如何验证？

1. 基于霍尔证明系统：循环不变式采用 $\top$ 即可；离开循环时，必有 $x \neq y$.
2. 基于程序执行：注意该程序有无数条执行！无法通过枚举执行的方法来验证该程序的正确性。

想一想：该程序的状态有多少个？

定义 (可达格局)

对任意格局 (l, s)，如果存在一条执行 $(l_0, s_0), \ldots, (l_n, s_n)$ 使得 (l_0, s_0) 为初始格局，$(l_n, s_n) = (l, s)$，则称 (l, s) 为可达格局 (reachable configuration)。记程序所有可达格局的集合为 C_R。

定理 (基于可达格局的正确性证明)

程序满足前置-后置条件对的充要条件是 C_R 中不含错误格局。

证明.

C_R 不含错误格局 $\Leftrightarrow$ 程序不含从初始格局到错误格局的执行 $\square$

如何计算 C_R？

定理

程序可达格局集合 C_R 是满足下列条件的最小集合：

- 所有初始格局都是 C_R 中的元素；
- 若 $(l, s) \in C_R, (l, st, l') \in \Delta$ 且 $(s, s') \in [\![st]\!]$，则 $(l', s') \in C_R$。

我们一般通过构造下面的可达图来计算 C_R：

定义

可达图（reachability graph）由点集 C_R 和边集 T 构成，并且

$$\big((l, s), st, (l', s')\big) \in T \quad \Leftrightarrow \quad (l, st, l') \in \Delta \text{ 且 } (s, s') \in [\![st]\!]$$

注意： C_R 可能是一个无限集合，以状态遍历的方法构造可达图的过程可能不终止！

作业

练习

请构造 P_{xor} 程序的可达图：

```
while(x == y){
  y = x;
  havoc  x;
}
```

总结

- 扩展 havoc 语句
- 控制流自动机
- 格局和执行
- 可达格局与可达图

下节课内容

基于控制流自动机的程序验证

- 最强后置条件
- 抽象可达图
- 精确的抽象可达图
- 有界模型检验

谢谢!