《软件分析与验证》

程序终止性

贺飞
清华大学软件学院

2024 年 5 月 17 日

上节课：过程

IMP 语言扩展

- 断言语句、假设语句、过程调用语句
- 过程契约

过程程序验证的基本方法

- 将程序分解为若干基本路径
- 基本路径中只含赋值语句和假设语句
- 分别为每条基本路径生成验证条件
- 如果所有基本路径的验证条件都是有效式，则程序满足规约

程序终止性（termination）：程序是否对所有输入都停机

程序终止性是程序的基本性质

程序终止性问题是不可判定问题[1]

阿兰 · 图灵

```
void f(int n) {
   while (n > 1)
      if (n % 2 == 0)
         n = n / 2;
      else
         n = 3 * n + 1;
}
```

对于左边的程序，至今仍无法证明它是终止的，也无法证明它是不终止的。

[1] Alan Turing. On computable numbers, with an application to the Entscheidungs problem. London Mathematical Society, 1936.

终止性的不同定义

关于终止性的不同定义:

- 程序总是都停机，即对所有输入都停机
- 程序有可能停机，即在某个输入下停机

我们一般采用第一种定义，这种终止性常常也称为“普遍终止性”(universal termination)。

此外，在终止性的表述中，常常把程序停机表述为程序运行到退出位置。

完全正确性 = 部分正确性 + 终止性

在前面讨论程序正确性时，都限定在程序终止的情况下进行讨论，这种正确性称程序的部分正确性 (partial correctness)。

定义

程序的完全正确性 (total correctness)，记为

$$[\varphi]\ st\ [\psi]$$

表示：

- 从满足 φ 的状态执行语句 st,
- 一定终止，
- 且结束时的状态满足 ψ。

例：解释规约

请写出下列霍尔三元组的含义?

1. $[\top]\ st\ [\psi]$：st 终止，且后状态满足 ψ
2. $[\varphi]\ st\ [\top]$：如果前状态满足 φ，那么 st 终止
3. $[\top]\ st\ [\top]$：st 终止

下面的霍尔三元组是有效的吗?

1. $[\top]\ \textbf{while}\ (0 < x)\ \{x := x + 1\}\ [x \le 0]$

良基关系

定义

集合 S 上的二元关系 $\prec$ 称为**良基关系**（well-founded relation），当且仅当集合 S 中不存在关于 $\prec$ 的无穷下降序列，即在集合 S 中找不到一个无穷序列 $s_1, s_2, \ldots$，使得对于任意的 $i > 0$ 有 $s_{i+1} \prec s_i$。

注意上面定义中的两个关键词："无穷"和"下降"。

设 $(1,2) \in \prec$，那么从 1 到 2 是沿着该关系上升的方向，而从 2 到 1 则是沿着该关系下降的方向。

示例：良基关系

下列哪些二元关系是良基关系?

- 自然数 $\mathbb{N}$ 上的后继关系 $\{(m, m+1) \mid m \in \mathbb{N}\}$;
  是
- 自然数 $\mathbb{N}$ 上的前驱关系 $\{(m+1, m) \mid m \in \mathbb{N}\}$;
  否，因为存在一条无穷下降序列：$0, 1, 2, \ldots$
- 自然数 $\mathbb{N}$ 上的小于关系 $<$;
  是
- 自然数 $\mathbb{N}$ 上的大于关系 $>$;
  否，因为存在一条无穷下降序列：$0, 1, 2, \ldots$
- 整数 $\mathbb{Z}$ 上的小于关系 $<$;
  否，因为存在一条无穷下降序列：$0, -1, -2, \ldots$

练习。。。

示例：良基关系

- 自然数 $\mathbb{N}$ 的幂集上的真子集关系 $\subset$。
  否，因为存在一条无穷下降序列：$\mathbb{N}, \mathbb{N} \setminus \{0\}, \mathbb{N} \setminus \{0, 1\}, \ldots$

练习。。。

- 有限语法树集合上的真子树关系 $\prec$。
  是

定义

设 $\prec_i$ $(1 \leq i \leq n)$ 是定义在集合 S_i 上的二元关系，令 $S = S_1 \times \cdots \times S_n$，并定义 S 上的字典序关系 (lexicographic order) $\prec$ 为:

$$(s_1, \ldots, s_n) \prec (t_1, \ldots, t_n) \Leftrightarrow \bigvee_{i=1}^{n} \left(s_i \prec_i t_i \wedge \bigwedge_{j=1}^{i-1} s_j = t_j \right)$$

$(s_1, \ldots, s_n) \prec (t_1, \ldots, t_n)$ 当且仅当存在某个位置 i，（1）$s_i \prec_i t_i$，且（2）在 i 之前的任意位置 j，$s_j = t_j$。

引理

如果关系 $\prec_1, \ldots, \prec_n$ 都是良基的，则字典序关系 $\prec$ 也是良基的。

秩函数

定义 (秩函数)

设 $\prec$ 是定义在集合 W 上的良基关系。考虑循环 **while** $(p)\{st\}$，设 $\mathcal{S}$ 是程序中所有可达状态的集合，如果存在一个从 $\mathcal{S}$ 到 W 的映射函数 δ，使得对程序经过循环头的任意相邻两次状态 s 和 s'，$(\delta(s'), \delta(s)) \in \prec$ 成立，则称 δ 为该循环的秩函数（ranking function）。

例

```
while (x + y < 100)
{
  x := x + 1;
}
```

考虑 $\mathbb{N}$ 上的良基关系 $<$，函数

$$\delta(s) = 100 - s(x) - s(y)$$

是该程序的一个秩函数。该函数常常也写作

$$\delta(x, y) = 100 - x - y$$

在一些资料中 [2]，秩函数也被称作循环变式（loop variant）。

直观理解，秩函数是一个定义在程序变量上的算术表达式：

- 值有下界。例如

$$\delta(x, y) = 100 - x - y \geq 0$$

- 循环每迭代一次，其值严格下降，且下降幅度不会趋向于无穷小。例如

$$\begin{aligned}\delta(x', y') &= 100 - x' - y' \\ &= 100 - (x + 1) - y \\ &= \delta(x, y) - 1\end{aligned}$$

[2] https://en.wikipedia.org/wiki/Loop_variant

定理 (程序终止性)

如果循环有秩函数，则循环是终止的。

证明.

归谬法：假定循环不终止，则必存在一条无限长的程序状态序列；通过秩函数可以将该无穷状态序列映射到集合 W 中关于 $\prec$ 的一个无穷下降序列，与 $\prec$ 是 W 的良基关系矛盾。 $\square$

定理

如果程序中每个循环都有秩函数，则程序是终止的。

证明.

略。 $\square$

证明程序终止性的基本步骤:

1. 确定集合 W 以及一个定义在 W 上的良基关系 $\prec$
   - 例如，$\mathbb{N}$ 和定义在 $\mathbb{N}$ 上的小于关系 $<$
   - 必要时，引入 $(\mathbb{N}, <)$ 的 n 元组，基于其字典序良基关系来寻找秩函数。
2. 确定一个从程序状态集到 W 的映射函数 $\delta: \mathcal{S} \to W$
3. 证明 δ 关于 $\prec$ 在每个基本路径上都是下降的。

P_1

```
while (x>=0) {
  x := x - 1;
}
```

P_2

```
while (x>=0) {
  assume y >= 1;
  x := x - y;
}
```

P_3

```
while (x>=0) {
  havoc y;
  assume y >= 1;
  x := x - y;
}
```

注意：基于 $(\mathbb{N}, <)$ 寻找秩函数不太方便，例如

- 对于 P_1 和 P_2，$\delta(x) = x$ 并非一个定义在 $(\mathbb{N}, <)$ 上的秩函数。
  - 考虑最后一次迭代得到的程序状态！
  - $\delta(x) = x + 1$ 是定义在 $(\mathbb{N}, <)$ 上的 P_1 的秩函数
  - $\delta(x) = x + y$ 是定义在 $(\mathbb{N}, <)$ 上的 P_2 的秩函数
- 对于 P_3，不存在定义在 $(\mathbb{N}, <)$ 上的秩函数。

P_1

```
while (x>=0) {
  x := x - 1;
}
```

P_2

```
while (x>=0) {
  assume y >= 1;
  x := x - y;
}
```

P_3

```
while (x>=0) {
  havoc y;
  assume y >= 1;
  x := x - y;
}
```

注意：更常用的是 $(\mathbb{Z}, <_{\mathbb{N}})$，其中

$$x <_{\mathbb{N}} y \text{ 当且仅当 } x < y \text{ 且 } y \in \mathbb{N}$$

例如：采用 $(\mathbb{Z}, <_{\mathbb{N}})$，$\delta(x) = x$ 对程序 P_1, P_2 和 P_3 都是秩函数。

标注秩函数

- 在基本路径中使用 $\downarrow$ 标注秩函数;
- 所有循环都需要标注秩函数;
- 例如:
  - $\downarrow i$
  - $\downarrow u - l + 1$
  - $\downarrow (i + 1, j - 1)$

$$
\begin{array}{c}
\hline
\{\varphi\} \\
\downarrow \delta(x) \\
st_1; \\
\vdots \\
st_n; \\
\downarrow \delta(x) \\
\hline
\end{array}
$$

- 只需要关注在开头和结尾都标注了秩函数的基本路径
- 一条基本路径上的两个秩函数可能不一样（想一想嵌套循环的情况）

对应的验证条件如下：

$$\varphi \to wp(st_1; \cdots ; st_n, \delta(x) \prec \delta(x'))[x' \mapsto x]$$

- 秩函数 $\delta(x)$ 在路径结束处的值比其在路径开始处的值要小
- 以 x' 表示 x 在路径开始处的值
- 完成 wp 计算后，需将 x' 重命名回其原名 x。

示例：终止性验证条件

$$\{i \geq 1\}$$
$$\downarrow i$$
$$i := i - 1;$$
$$\downarrow i$$

对应的验证条件如下：

$$\begin{aligned}
& i \geq 1 \rightarrow wp(i := i - 1, i < i')[i' \mapsto i] \\
\Leftrightarrow\ & i \geq 1 \rightarrow (i - 1 < i')[i' \mapsto i] \\
\Leftrightarrow\ & i \geq 1 \rightarrow i - 1 < i
\end{aligned}$$

有效式

$$\{\varphi : i+1 \geq 0 \wedge i-j \geq 0\}$$
$$\downarrow (i+1, i-j)$$
$$st_1 : \textbf{assume} \;\; j \geq i$$
$$st_2 : \;\; i := i-1$$
$$\downarrow (i+1, i+1)$$

对应的验证条件如下:

$$\varphi \to wp(st_1; st_2, (i+1, i+1) <_2 (i' + 1, i' - j'))[i' \mapsto i, j' \mapsto j]$$

其中

$$\begin{aligned} & wp(st_1; st_2, (i+1, i+1) <_2 (i' + 1, i' - j')) \\ \Leftrightarrow\ & j \geq i \to (i, i) <_2 (i' + 1, i' - j') \end{aligned}$$

$$\Leftrightarrow \varphi \to \left(j \geq i \to (i, i) <_2 (i' + 1, i' - j')\right) [i' \mapsto i, j' \mapsto j]$$

循环示例

示例：计算年份

```
int year = 1980;
/*@ loop invariant days>0;
    loop variant days; */
while(days > 365)
{
    if(!is_leap_year(year)) {
        days = days - 365;
        year = year + 1;
    } else if(days > 366) {
        days = days - 366;
        year = year + 1;
    }
}
return year;
```

示例：计算年份

第一条基本路径：

$$
\begin{array}{c}
\{days > 0\} \\
\downarrow days \\
st_0 : \textbf{assume} \quad days > 365 \\
st_1 : \textbf{assume} \quad \neg is\_leap\_year(year) \\
st_2 : \quad days := days - 365 \\
st_3 : \quad year := year + 1 \\
\downarrow days
\end{array}
$$

其终止性验证条件为：

$$
\begin{aligned}
& days > 0 \rightarrow wp(st_0; st_1; st_2; st_3, days < days')[days' \mapsto days] \\
\Leftrightarrow\ & days > 0 \wedge days > 365 \wedge \neg is\_leap\_year(year) \rightarrow days - 365 < days
\end{aligned}
$$

有效式

第二条基本路径：

$$
\begin{array}{c}
\{days > 0\} \\
\downarrow days \\
st_0 : \textbf{assume} \;\; days > 365 \\
st_1 : \textbf{assume} \;\; is\_leap\_year(year) \\
st_2 : \textbf{assume} \;\; days > 366 \\
st_3 : \;\; days := days - 366 \\
st_4 : \;\; year := year + 1 \\
\downarrow days
\end{array}
$$

其终止性验证条件为：

$$
\begin{aligned}
& days > 0 \to wp(st_0; st_1; st_2; st_3; st_4, days < days')[days' \mapsto days] \\
\Leftrightarrow\ & days > 0 \wedge days > 365 \wedge is\_leap\_year(year) \wedge days > 366 \\
& \qquad \to days - 366 < days
\end{aligned}
$$

有效式

第三条基本路径:

$$
\begin{array}{c}
\hline
\{days > 0\} \\
\downarrow days \\
st_0 : \textbf{assume} \quad days > 365 \\
st_1 : \textbf{assume} \quad is\_leap\_year(year) \\
st_2 : \textbf{assume} \quad days \leq 366 \\
\downarrow days \\
\hline
\end{array}
$$

其终止性验证条件为:

$$
\begin{aligned}
& days > 0 \to wp(st_0; st_1; st_2, days < days')[days' \mapsto days] \\
\Leftrightarrow\ & days > 0 \wedge days > 365 \wedge is\_leap\_year(year) \wedge days \leq 366 \\
& \qquad \to days < days \\
\Leftrightarrow\ & days = 366 \wedge is\_leap\_year(year) \to days < days
\end{aligned}
$$

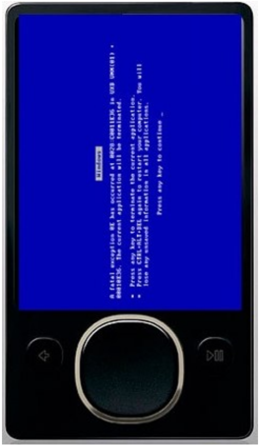
不是有效式

Zune 闰年漏洞

- 这个漏洞真实地发生在 Zune 的 MP3 播放器中
- 当下面条件成立时，第三条基本路径的验证条件为假：

$$days = 366 \wedge is\_leap\_year(year)$$

- 对于 Zune 用户而言，满足该条件的日期是：2008 年 12 月 31 日
- 此时，程序陷入死循环，Zune 播放器蓝屏死机！



修复 Zune 漏洞


```
int year = 1980;
/*@ loop invariant days>0;
    loop variant days; */
while(days > 365)
{
    if(!is_leap_year(year)) {
        days = days - 365;
        year = year + 1;
    } else if(days >= 366) {
        days = days - 366;
        year = year + 1;
    }
}
return year;
```

如何确认该修复确实排除了 Zune 漏洞？

第三条基本路径:

$$\begin{array}{c} \{days > 0\} \\ \downarrow days \\ st_0 : \textbf{assume} \quad days > 365 \\ st_1 : \textbf{assume} \quad is\_leap\_year(year) \\ st_2 : \textbf{assume} \quad days < 366 \\ \downarrow days \end{array}$$

其终止性验证条件为:

$$\begin{aligned} & days > 0 \rightarrow wp(st_0; st_1; st_2;, days < days')[days' \mapsto days] \\ \Leftrightarrow\ & days > 0 \wedge days > 365 \wedge is\_leap\_year(year) \wedge days < 366 \\ & \qquad \rightarrow days < days \\ \Leftrightarrow\ & false \wedge is\_leap\_year(year) \rightarrow days < days \end{aligned}$$

有效式

递归过程示例

类似于循环，递归过程的入口位置也会被程序反复经过，因此也需要标注秩函数，以保证该过程不会被无限递归地调用。

定义 (秩函数)

设 $\prec$ 是定义在集合 W 上的良基关系。考虑递归过程 $m(x_1, \ldots, x_n)$，设 $\mathcal{S}$ 是程序中所有可达状态的集合，如果存在一个从 $\mathcal{S}$ 到 W 的映射函数 δ，使得在任意两次相邻的对 m 过程的调用中，它们的入口状态 s 和 s' 都满足 $(\delta(s'), \delta(s)) \in \prec$，则称 δ 为该过程的秩函数 (ranking function)。

示例：二分查找


```
/*@ requires u - l + 1 >= 0;
    ensures true;
    decreases u - l + 1; */
int BinarySearch(int a[], int len, int l, int u, int e)
{
    if (l > u) return 0;
    int m = (l + u) / 2;
    if (a[m] == e) return 1;
    else if (a[m] < e)
        return BinarySearch(a, len, m + 1, u, e);
    else
        return BinarySearch(a, len, l, m - 1, e);
}
```

第一条基本路径：

$$\{u-l+1\geq 0\}$$
$$\downarrow u-l+1$$
$$\textbf{assume} \quad l>u;$$
$$rv := 0;$$

第二条基本路径：

$$\{u-l+1\geq 0\}$$
$$\downarrow u-l+1$$
$$\textbf{assume} \quad l\leq u;$$
$$m := (l+u)/2;$$
$$\textbf{assume} \quad a[m] = e;$$
$$rv := 1;$$

- 只有一端有秩函数的基本路径不需考虑
- 事实上，这两条基本路径一定终止，无需为其生成终止性验证条件

第三条基本路径：

$$
\begin{array}{c}
\hline
\{u-l+1 \geq 0\} \\
\downarrow u-l+1 \\
\textbf{assume} \quad l \leq u; \\
m := (l+u)/2; \\
\textbf{assume} \quad a[m] \neq e; \\
\textbf{assume} \quad a[m] < e; \\
\downarrow u-(m+1)+1 \\
\hline
\end{array}
$$

其终止性验证条件为：

$$
\begin{aligned}
& u-l+1 \geq 0 \wedge l \leq u \wedge a[(l+u)/2] \neq e \wedge a[(l+u)/2] < e \\
& \qquad \rightarrow u-(((l+u)/2)+1)+1 < u-l+1
\end{aligned}
$$

有效式

第四条基本路径：

$$
\begin{array}{c}
\{u-l+1 \geq 0\} \\
\downarrow u-l+1 \\
\textbf{assume} \quad l \leq u; \\
m := (l+u)/2; \\
\textbf{assume} \quad a[m] \neq e; \\
\textbf{assume} \quad a[m] \geq e; \\
\downarrow (m-1)-l+1
\end{array}
$$

其终止性验证条件为：

$$
\begin{aligned}
& u-l+1 \geq 0 \wedge l \leq u \wedge a[(l+u)/2] \neq e \wedge a[(l+u)/2] \geq e \\
& \qquad \rightarrow (((l+u)/2)-1)-l+1 < u-l+1
\end{aligned}
$$

有效式

总结

- 终止性证明的基础——良基关系
- 终止性证明的依据——秩函数
- 终止性证明的示例

- 推导循环不变式

谢谢!