Hoare Logic

hoare logic 是程序验证系统的基础，是命令式语言的一种公理语义

Specifications

hoare logic 最基础的组成部分是三元组

- partial correctness specification: $\{p\}c\{q\}$
- total correctness specification: $[p]c[q]$

其中 p, q 称为 assertion，p 为 precondition，q 为 postcondition

对于 partial correctness，其有效 $\iff$ c 在满足 p 的状态下开始执行，且如果 c 能终止，则状态满足 q

total correctness 的区别在于 c 必须终止

不严谨地说，total correctness = partial correctness + termination

logical variable 是仅出现在 assertion 而不出现在 program 中的变量，其值是不变的

语法层面：$\vdash p$ 表示按照推断规则，存在 p 的推导或是说证明

语义层面：$\sigma \models p$ 表示 p 在 σ 下得到满足

于是一个 assertion p 被称为 valid 表示在所有 σ 下 p 都是满足的，而不可满足即是指 $\neg p$ valid

Inference Rules

形式化证明需要公理和推导规则，证明过程的组成或是一个公理或是从之前的结果根据规则得出

定义 judgement，包括三类

- 谓词逻辑的公式：$x + 1 > x$
- partial correctness specification
- total correctness specification

则 $\vdash J$ 表示 judgement 可证明，谓词逻辑的证明已知，后两者的证明由 hoare logic 给出

e.g. 最简单的赋值规则

$$\frac{}{\{p[e/x]\}x := e\{p\}}$$

但是前向的规则要复杂一些（v 为 x 的旧值）

$$\frac{}{\{p\}x := e\{\exists v.\, x = e[v/x] \wedge p[v/x]\}}$$

也可以增强 precondition 或是减弱 postcondition

$$\frac{p \implies q \quad \{q\}c\{r\}}{\{p\}c\{r\}}$$

$$\frac{\{p\}c\{q\} \quad q \implies r}{\{p\}c\{r\}}$$

需要注意的是 while 循环在证明 total correctness 和 partial correctness 时规则不同

partial correctness 不需要循环能终止

$$\frac{\{i \wedge b\}c\{i\}}{\{i\}\textbf{while}\ b\ \textbf{do}\ c\{i \wedge \neg b\}}$$

total correctness 则需保证循环能够终止

$$\frac{[i \wedge b \wedge (e = x_0)]c[i \wedge (e < x_0)] \quad i \wedge b \implies e \geqslant 0}{[i]\textbf{while}\ b\ \textbf{do}\ c[i \wedge \neg b]}$$

除了循环不变量还需要一个循环变量

所有的推导规则如下

$$\frac{}{\{p[e/x]\}\ x := e\ \{p\}}\ (\text{AS}) \qquad \frac{}{\{p\}\mathbf{skip}\{p\}}\ (\text{SK})$$

$$\frac{\{p\}c_1\{r\} \qquad \{r\}c_2\{q\}}{\{p\}c_1\ ;\ c_2\{q\}}\ (\text{SC}) \qquad \frac{\{p \wedge b\}c_1\{q\} \qquad \{p \wedge \neg b\}c_2\{q\}}{\{p\}\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\{q\}}\ (\text{CD})$$

$$\frac{\{i \wedge b\}\ c\ \{i\}}{\{i\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{i \wedge \neg b\}}\ (\text{WHP})$$

$$\frac{[i \wedge b \wedge (e = x_0)]\ c\ [i \wedge (e < x_0)] \qquad i \wedge b \Rightarrow e \geq 0}{[i]\ \mathbf{while}\ b\ \mathbf{do}\ c\ [i \wedge \neg b]}\ (\text{WHT})$$

$$\frac{p \Rightarrow q \qquad \{q\}c\{r\}}{\{p\}c\{r\}}\ (\text{SP}) \qquad \frac{\{p\}c\{q\} \qquad q \Rightarrow r}{\{p\}c\{r\}}\ (\text{WC})$$

$$\frac{\{p\}c\{q\} \qquad \{p'\}c\{q'\}}{\{p \wedge p'\}c\{q \wedge q'\}}\ (\text{CA}) \qquad \frac{\{p\}c\{q\} \qquad \{p'\}c\{q'\}}{\{p \vee p'\}c\{q \vee q'\}}\ (\text{DA})$$

推导规则可以混合起来使用，产生新的推导规则，如

$$\frac{p \implies q[e/x]}{\{p\}x := e\{q\}}$$

对于多条语句的程序，考虑规则 SC，需要找到一个满足条件的 assertion r，同样，对于 while 循环的规则，需要找出一个显式的不变量 i，可以提前将这类 assertion 标注进程序，保证程序运行到该点 assertion 一定为真，有助于证明

混合推导规则或是对程序做 annotation 都有助于证明，而这些技术也可以用于自动化验证

Automated Program Verification

目标：自动化 hoare logic 的证明过程

流程

- 输入为 specification，用户进行一些辅助的 annotation（需要技巧和对程序的理解）
- 自动生成一系列纯数学公式，称为 verification conditions (VC)
- VC 被传入定理证明器进行自动证明
- 如果 $\{p\}c\{q\}$ 生成的 VC 能被全部证明，则 $\vdash \{p\}c\{q\}$

一个 properly annotated command 满足在以下位置有 assertion

- 所有不是赋值语句的语句之前
- 所有 while 语句的 do 之后

这些 assertion 需要在任何情况下都能满足

生成 VC 的过程是递归的，
$VC(C(c_1, c_2)) = \{vc_1, vc_2 \ldots vc_n\} \cup VC(c_1) \cup VC(c_2)$，且每一步生成的 VC 是根据语法结构决定的，这个过程是确定性的

而根据 VC 可证明得出 specification 可证明是反向，即归纳，从原子命令开始到复合命令

- $\{p\}\mathbf{skip}\{q\}$ 的 VC 为 $p \implies q$
- $\{p\}x := e\{q\}$ 的 VC 为 $p \implies q[e/x]$
- $\{p\}\mathbf{if}\, b\, \mathbf{then}\, c_1\ \mathbf{else}\ c_2\{q\}$ 的 VC 是两个子语句的 VC 的并：$\{p \wedge b\}c_1\{q\}, \{p \wedge \neg b\}c_2\{q\}$
- $\{p\}c_1; c_2 \ldots c_{n-1}; \{r\}c_n\{q\}$ 的 VC 是 $\{p\}c_1; c_2 \ldots c_{n-1}; \{r\}$ 和 $\{r\}c_n\{q\}$ 的 VC 的并
- $\{p\}c_1; c_2 \ldots c_{n-1}; x := e\{q\}$ 的 VC 是 $\{p\}c_1; c_2 \ldots c_{n-1}\{q[e/x]\}$ 的 VC
- $\{p\}\mathbf{while}\, b\, \mathbf{do}\ \{i\}c\{q\}$ 的 VC 是 $p \implies i$，$i \wedge \neg b \implies q$ 和 $\{i \wedge b\}c\{i\}$ 的 VC 的并
- total correctness 的循环只需要加上循环变量的 VC：$p \implies i$，$i \wedge \neg b \implies q$，$i \wedge b \implies e \geqslant 0$，以及 $\{i \wedge b \wedge e = x_0\}c\{i \wedge e < x_0\}$

自动验证的流程中 VC 是自动生成的，大部分 VC 也可以自动证明，人可以帮助 annotation 和证明 VC

Soundness and Completeness

用 $\vdash$ 表示语法，用 $\models$ 表示语义

系统的可靠性（soundness）表示如果 $\vdash \{p\}c\{q\}$ 则有 $\models \{p\}c\{q\}$，total correctness 同理

系统的完备性（completeness）表示如果 $\models \{p\}c\{q\}$ 则有 $\vdash \{p\}c\{q\}$，total correctness 同理

hoare logic 是可靠的，但谓词逻辑是不完备的（哥德尔不完备定理）

hoare logic 的语义是

- 如果 c 在满足 $\sigma \models p$ 的状态 σ 下执行
- c 的执行从 σ 开始，在 σ' 终结（语句的语义）
- 则 $\sigma' \models q$

形式化的定义为

$$\models \{p\}c\{q\} \iff \forall \sigma, \sigma'. (\sigma \models p) \land ((c, \sigma) \to^* (\mathbf{skip}, \sigma')) \implies (\sigma' \models q)$$
$$\models [p]c[q] \iff \forall \sigma. (\sigma \models p) \implies \exists \sigma'. ((c, \sigma) \to^* (\mathbf{skip}, \sigma')) \land (\sigma' \models q)$$

hoare logic 是不完备的，考虑 $\models \{\mathbf{true}\}\mathbf{skip}\{p\} \iff \models p$，如果完备性成立则有 $\vdash p$，与哥德尔不完备定理矛盾，或是 $\models \{\mathbf{true}\}c\{\mathbf{false}\} \iff c$ 不停机，而停机问题不可判定

造成不完备的原因是 SP 与 WC 两条规则基于谓词逻辑中蕴含的有效性，因此可以将证明系统的逻辑（hoare logic）和 assertion 分离，即如果能给定 assertion 的有效性，则可以推导出所有的 partial correctness，称为 relative completeness：如果 $\models \{p\}c\{q\}$，则 $\Gamma \vdash \{p\}c\{q\}$，其中 $\Gamma = p$ 或 $\Gamma \models p$

weakest precondition：给定命令 c 和 assertion q，则 weakest precondition $wp(c, q)$ 是满足

$$\sigma \models wp(c, q) \iff (\forall \sigma'. (c, \sigma) \to^* (\mathbf{skip}, \sigma') \implies \sigma' \models q)$$

的 assertion，于是有

$$\forall p, c, q. \models \{p\}c\{q\} \iff \models (p \implies wp(c, q))$$

而一个 assertion language 如果被称为 expressive，说明任取 c, q，有 $wp(c, q)$ 是该语言中的一个 assertion

于是有引理

$$\forall c, q. \vdash \{wp(c, q)\}c\{q\}$$

根据此引理即可证明 relative completeness

- $\vdash \{wp(c, q)\}c\{q\}$
- 由于 $\models \{p\}c\{q\}$，有 $\models p \implies wp(c, q)$
- 于是 Γ 中有 $p \implies wp(c, q)$
- 根据 SP，有 $\Gamma \vdash \{p\}c\{q\}$

即只要能得知 $p \implies wp(c, q)$ ，且 wp 可被表示，就能使得 hoare logic 相对完备