真实世界的程序验证

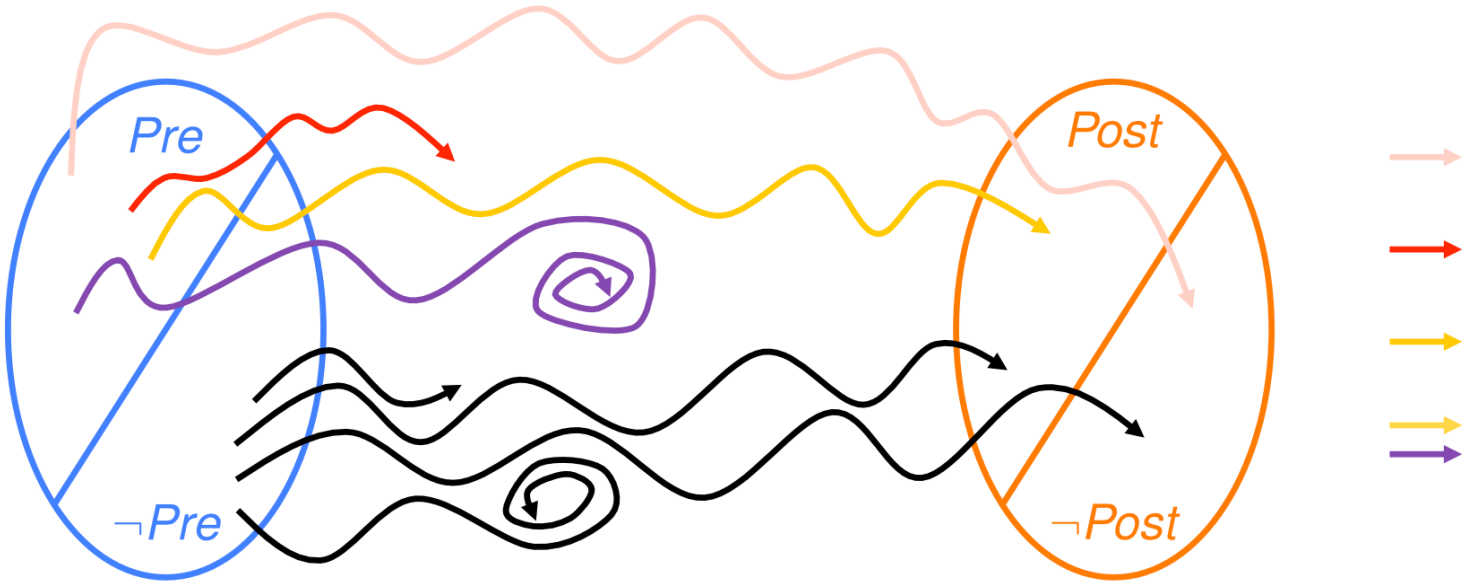
第三次习题课

曹奕远

2023.5.25

基础霍尔逻辑

- 三元组：对程序可能状态和执行路径的抽象表述
- 作为逻辑系统：可靠性和完备性



基础霍尔逻辑的问题

- 问题一：推理规则是声明式而不是算法式的
  - 结构规则的调用时机
  - 顺序计算的中间状态
- 问题二：内存的全局推理而不能局部推理
  - 前后条件都是对全局内存状态的描述
  - 包含与当前程序片段不相关的部分
- 问题三：考虑更加丰富的语言特性
  - 非结构化的控制流
  - 并发执行

算法化的霍尔逻辑：最弱前条件谓词转换器

$$wp\,t\,Q \;\equiv\; \min_{(\vdash)} \{\, H \mid \{H\}\,t\,\{Q\} \,\}$$
$$\big(\{wp\,t\,Q\}\,t\,\{Q\}\big) \;\wedge\; \big(\forall H.\ \{H\}\,t\,\{Q\} \Rightarrow H \vdash wp\,t\,Q\big)$$
$$wp\,t\,Q \;\equiv\; \lambda h.\ \big(\{\lambda h'.\, h' = h\}\,t\,\{Q\}\big)$$
$$wp\,t\,Q \;\equiv\; \exists H.\ H \star \big[\{H\}\,t\,\{Q\}\big]$$
$$H \vdash wp\,t\,Q \;\Leftrightarrow\; \{H\}\,t\,\{Q\}$$

- WP t : Post -> Pre
- {H} t {Q} <==> H => WP t Q
  - => : WP t Q不强于使得后条件Q成立的任意前条件
  - <= : WP t Q的确是程序t使得后条件Q成立的前条件
  - 综上，WP t Q是程序t使得后条件Q成立的最弱前条件
- 霍尔逻辑的算法形式：WP演算（由Dijkstra提出）
  - 即计算WP t的结构化算法
  - 将构造{H} t {Q}的推导树转化为蕴涵式检查H => WP t Q
- 展示：F*中简单语言的WP演算

半自动程序验证器实现

- 规约描述语言：一般是一阶逻辑的扩展
- 关键组件：验证条件生成器（基于WP演算）
- 半自动的证明方法
  - 用户提供注释：循环不变式和关键的中间断言
  - 生成的验证条件被交给后端的求解器尝试自动求解
- 演示
  - Binary search in Dafny：容易安装和使用
    - VS Code插件
  - Quicksort in Why3：拥有不同后端；调用操作验证条件的证明策略
  - F*：基于精化类型和依赖类型可以在类型层面嵌入WP演算
    - 称为Dijkstra monads

局部推理的霍尔逻辑：分离逻辑

- 资源的所有权（ownership）
  - 知情权（knowledge about the resource）
  - 处置权（permission to change the resource）
- 资源的特点：不可复制和随意丢弃
  - 线性逻辑
- 局部推理的上下文无关性
  - 资源的分离合取

$$\{H\}\, t\, \{Q\} \quad \equiv \quad \forall H'.\ ^{HOARE}\{H \star H'\}\, t\, \{Q \star H'\}$$

To understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged.

资源敏感的断言语言

- 只需要描述当前使用的局部资源而不是全局资源
- 用于推理时结合frame规则在更多资源的情况下调用

$$\frac{\{H\}\ t\ \{Q\}}{\{H \star H'\}\ t\ \{Q \star H'\}}\ \textsc{frame-for-commands}$$

Operator	**Notation**	**Definition**
empty predicate	$[\,]$	$\lambda h.\ h = \varnothing$
pure fact	$[P]$	$\lambda h.\ h = \varnothing\ \wedge\ P$
singleton	$p \mapsto v$	$\lambda h.\ h = (p \to v)\ \wedge\ p \neq \mathit{null}$
separating conjunction	$H_1 \star H_2$	$\lambda h.\ \exists h_1 h_2.\ h_1 \perp h_2\ \wedge\ h = h_1 \uplus h_2\ \wedge\ H_1\, h_1\ \wedge\ H_2\, h_2$
existential quantifier	$\exists x.\ H$	$\lambda h.\ \exists x.\ H\, h$
universal quantifier	$\forall x.\ H$	$\lambda h.\ \forall x.\ H\, h$

基本操作的规约

$$
\begin{array}{lll}
\textit{REF:} & \{[\,]\} \quad (\texttt{ref}\, v) & \{\lambda r.\ \exists p.\ [r = p] \star (p \hookrightarrow v)\} \\
\textit{GET:} & \{p \hookrightarrow v\} \quad (\texttt{get}\, p) & \{\lambda r.\ [r = v] \star (p \hookrightarrow v)\} \\
\textit{SET:} & \{p \hookrightarrow v\} \quad (\texttt{set}\, p\, v') & \{\lambda \_.\ (p \hookrightarrow v')\} \\
\textit{FREE:} & \{p \hookrightarrow v\} \quad (\texttt{free}\, p) & \{\lambda \_.\ [\,]\} \\
\textit{ADD:} & \{[\,]\}\ ((+)\, n_1\, n_2) & \{\lambda r.\ [r = n_1 + n_2]\} \\
\textit{DIV:} \quad n_2 \neq 0 \ \Rightarrow & \{[\,]\}\ ((\div)\, n_1\, n_2) & \{\lambda r.\ [r = n_1 \div n_2]\}
\end{array}
$$

分离逻辑验证工具实现

- 交互式验证
  - CFML from Inria
  - VST from Princeton

  Higher-order: nested triples (facilitates higher-order reasoning)
- 程序分析工具
  - Infer from Meta

  Large scale analysis
- 演示：CFML中的incr和incr2

更多语言与特性的霍尔逻辑

- 面向汇编程序的霍尔逻辑
- 并发分离逻辑（CSL）
- 对资源消耗进行推理的霍尔逻辑
- Effect handler的分离逻辑

其他有趣的东西

- Language embedding mechanisms: shallow and deep embedding
- Derived forms: macro-expressiveness of language constructs
- Proof engineering: intrinsic verification and extrinsic verification
- Resource-aware (low-level) programming languages