软件科学基础

# Separation Logic & Incorrectness Logic
# a Quick Look

熊英飞
北京大学

# 扩展霍尔逻辑验证指针

```
void swap(int *x, int* y) {
    *x = (*x)^(*y);
    *y = (*x)^(*y);
    *x = (*x)^(*y);
}
```

下面的霍尔三元组成立吗?

$\{*\,\mathrm{x} = a \wedge *\,\mathrm{y} = b\}\mathrm{swap}(\mathrm{x},\mathrm{y})\{*\,\mathrm{x} = b \wedge *\,\mathrm{y} = a\}$

# 扩展霍尔逻辑验证指针

- 不成立，因为x和y可能是别名（指向同一块地址）
- 一个成立的霍尔三元组如下所示

$$\{x \neq y \rightarrow * \mathrm{x} = a \wedge * \mathrm{y} = b\}\mathrm{swap}(\mathrm{x}, \mathrm{y})\{x \neq y \rightarrow * \mathrm{x} = b \wedge * \mathrm{y} = a\}$$

- 对于指针程序的验证需要反复论证别名
  - 如链表需要无环，两个链表连接不能连接自己的一部分
- 能否把别名的论证变成逻辑的一部分?

# 分离逻辑Separation Logic

- 扩展了霍尔逻辑，将“无别名”作为逻辑符号的一部分
- 源于John C. Reynolds, Peter O‘Hearn, Samin Ishtiaq and Hongseok Yang等人在99-01年期间发表的四篇论文
- 是Facebook的程序分析工具Infer的理论基础（理论上）
- 本世纪程序验证上最重要的工作之一
  - 获得2016年的CAV Award和哥德尔奖

# 分离逻辑概念

- Store：从变量名到变量值的映射
  - 表示栈
- Heap：从内存地址到内存值的部分映射
  - 表示堆
  - $h_1 \perp h_2$表示两个堆不交，即定义域交集为空
  - $h_1 \cup h_2$表示合并两个不交的堆

# 分离逻辑断言

- 语法

$$
\begin{array}{lll}
Ass & := & \mathbf{emp} \\
 & | & Exp_1 \mapsto Exp_2 \\
 & | & Ass_1 * Ass_2 \\
 & | & Ass_1 -\!\!* \; Ass_2 \\
 & | & \text{其他一阶逻辑式子} \\
Exp & := & \text{程序表达式}
\end{array}
$$

- 语义
  - 三元关系$s, h \vDash P$表示谓词P在Store s和Heap h上成立

# emp

- $s, h \vDash \mathbf{emp}$
  - 表示h为空，对所有内存地址都未定义

# $Exp_1 \mapsto Exp_2$

- $s, h \vDash e \mapsto e'$
  - 表示地址$e$保存了值$e'$，即$h([e]_s) = [e']_s$且$h$对$[e]_s$以外的值都没有定义，其中$[e]_s$表示$e$在s上求值的结果
  - 如$x \mapsto$1表示指针x所指向的地址保存了值1，且堆中没有其他地址
  - 如$x \mapsto n + 1$表示指针x所指向的地址保存的值等于变量n保存的值加上1，且堆中没有其他地址

# $Ass_1 * Ass_2$ （分离合取、星号）

- $s, h \vDash P * Q$
  - $\exists h_1, h_2$，使得$h = h_1 \cup h_2 \wedge h_1 \perp h_2$
    - $s, h_1 \vDash P$
    - $s, h_2 \vDash Q$
  - 即$P$和Q都成立，但其中地址不相交
  - 分离逻辑中最重要的逻辑符号
  - 如$x \mapsto a * y \mapsto b$说明x和y不是别名
  - 如：下面递归定义了判断p是否为链表的谓词，*用来保证链表不会出现环
    - $list(p) = p = null \rightarrow \mathbf{emp}$
      $\wedge\, p \neq null$
      $\rightarrow \exists a, p', p \mapsto \{value\!: a, next\!: p'\} * list(p')$
    - 练习：用分离合取定义二叉树的谓词
  - 性质：$P = \mathbf{emp} * P = P * \mathbf{emp}$

# $Ass_1 -\!\!* \; Ass_2$（分离蕴含、魔法棒）

- $s, h \vDash P -\!\!* \; Q$
  - $\forall h', (h' \perp h) \wedge \big((s, h' \vDash P) \longrightarrow (s, h' \cup h \vDash Q)\big)$
  - 即给h补上一块满足P的堆就能满足Q
  - 如$(x \mapsto 1) -\!\!* \; Q$表示当前堆不包括x，但加上一个x指向1的堆就满足Q

- 分离合取和分离蕴含具有一些普通合取和蕴含的性质
  - $\frac{\vDash P \wedge (P \rightarrow Q)}{\vDash Q}$ $\qquad \frac{s,h \vDash P * (P -\!* Q)}{s,h \vDash Q}$

# 扩展霍尔三元组

- {P}c{Q}
  - 表示对任意满足P的s和h，如果执行语句c之后得到了s’和h’，那么s’和h’满足Q

- 如：
  - $\{x \mapsto a * y \mapsto b\}$swap(x,y)$\{x \mapsto b * y \mapsto a\}$

# 推导规则-新增语句相关

- Store: $\{x \mapsto -\}$*x=v$\{x \mapsto v\}$
- Load: $\{x \mapsto v\}$a=*x$\{a = v \wedge x \mapsto v\}$
- Alloc: $\{\mathbf{emp}\}$x=malloc()$\{x \mapsto -\}$
- AllocFail: $\{\mathbf{emp}\}$x=malloc()$\{x \mapsto - \vee x = 0\}$
- DeAlloc: $\{x \mapsto -\}$free(x)$\{\mathbf{emp}\}$

- “$x \mapsto -$”表示“$\exists v, x \mapsto v$”

# 推导规则-Frame规则

- Frame:

- $$\frac{\{P\}c\{Q\}}{\{P*R\}c\{Q*R\}}$$

# 完整性

- 以上推导规则，加上断言上的公理（本课程跳过）、部分霍尔逻辑规则和一阶逻辑规则，可以推导出所有的为真的三元组

# 证明举例1

- 证明定理：$\{x \mapsto a * y \mapsto b\}$t=*x;*x=*y;*y=t;$\{x \mapsto b * y \mapsto a\}$
- 根据Load可得
  - $\{x \mapsto a\}t =* x\{t = a \wedge x \mapsto a\}$
- 注意$t = a$不涉及到堆，根据Consequence可得
  - $\{x \mapsto a\}t =* x\{(t = a \wedge \mathbf{emp}) * x \mapsto a\}$
- 再根据Frame可得
  - $\{x \mapsto a * y \mapsto b\}t =* x\{(t = a \wedge \mathbf{emp}) * x \mapsto a * y \mapsto b\}$
- 根据Load和Store可得
  - $\{x \mapsto - * y \mapsto b\}$*x=*y$\{x \mapsto b * y \mapsto b\}$
- 根据Frame和Consequence可得
  - $\{(t = a \wedge \mathbf{emp}) * x \mapsto a * y \mapsto b\}$*x=*y$\{(t = a \wedge \mathbf{emp}) * x \mapsto b * y \mapsto b\}$
- 类似可得
  - $\{(t = a \wedge \mathbf{emp}) * x \mapsto b * y \mapsto b\}$*y=t$\{x \mapsto b * y \mapsto a\}$
- 根据Seq，原命题得证

# 证明举例2

- void delete(p) {
```
void delete(p) {
  if (p = null) return;
  else {
    delete(p->next);
    free(p);
  }
}
```
- 证明：$\{list(p)\}$delete(p)$\{\mathbf{emp}\}$

# 证明举例2

- 先考虑then分支，根据Skip可得
  - $\{\mathbf{emp}\}skip\{\mathbf{emp}\}$
- 再根据Consequence可得
  - $\{list(p) \wedge p = null\}skip\{\mathbf{emp}\}$
- 再考虑else分支，根据DeAlloc可得
  - $\{p \mapsto -\}\text{free(p)}\{\mathbf{emp}\}$
- 根据归纳假设可得
  - $\{list(p \rightarrow next)\}$delete(p->next)$\{\mathbf{emp}\}$
- 从上面两项，根据Frame可得
  - $\{list(p \rightarrow next) * p \mapsto -\}$delete(p->next)$\{p \mapsto - * \mathbf{emp}\}$
  - $\{p \mapsto - * \mathbf{emp}\}\text{free(p)}\{\mathbf{emp} * \mathbf{emp}\}$
- 再根据Consequence可得
  - $\{(list(p) \wedge p \neq null)\}$delete(p->next)$\{p \mapsto - * \mathbf{emp}\}$
  - $\{p \mapsto - * \mathbf{emp}\}\text{free(p)}\{\mathbf{emp}\}$
- 最后根据If，原命题得证

# 教材和分离逻辑相关内容

- 教材第5卷：一套用分离逻辑验证C语言的证明系统VST
- 教材第6卷：一套基于Ocaml程序的验证介绍分离逻辑的教程

# VST

- 用分离逻辑验证C语言的证明系统
- VST的逻辑是higher-order impredicative concurrent separation logic
  - separation logic——用于处理指针
  - concurrent separation logic——用于处理并行
  - higher-order impredicative program logic——用于处理函数指针、面向对象重载等
- 上海交通大学曹钦翔老师为主力开发和维护人员之一
- 可以高效验证实际的C程序

曹钦翔
上海交大副教授
CMO、NOI双全国第一
北大哲学本科、普林斯顿博士

# VST

- Decorated Program面向实际C程序的终极强化版
  - 提供Coq数据结构表示C程序的AST
  - 提供Coq类型表示断言和霍尔三元组
  - 提供Coq证明策略用于证明霍尔三元组
- 重新在Coq中定义出一套新的面向C的证明语言

# VST 与 VST-A 部件框架

| | |
|---|---|
| VST-A | 依据程序断言自动构建 Hoare logic 证明框架 |
| VST-Floyd | 构建自动符号执行指令库 |
| | 引入复合谓词 data_at |
| VST Verifiable C | 引入基本谓词 mapsto 描述内存状态 |
| | 定义分离合取等逻辑连接词，以支持分离逻辑 |
| CompCert | C编译器验证框架 |

# VST验证举例——程序

```
struct list {int head; struct list *tail;};

struct list *reverse (struct list *p) {
  struct list *w, *t, *v;
  w = NULL;
  v = p;
  while (v) {
    t = v->tail;
    v->tail = w;
    w = v;
    v = t;
  }
  return w;
}
```

# 一个 VST-Floyd 验证的例子

\\ Given $l_1, l_2, w$ and $v$. Assume $v \neq \text{NULL}$.
\\ $\{l = \text{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \rightsquigarrow l_2\}$
\\ $\{\exists h\ l_2'\ y.\ l = \text{rev}(l_1) \cdot [h] \cdot l_2' \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l_2'\}$

\\ Given $h, l_2'$ and $y$. Assume $l = \text{rev}(l_1) \cdot [h] \cdot l_2'$.
\\ $\{[\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l_2'\}$
**t = v→ tail;**
\\ $\{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l_2'\}$
**v→ tail = w;**
**w = v;**
**v = t;**
\\ $\{\exists\ l_1\ l_2\ w\ v.\ l = \text{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \rightsquigarrow l_2\}$

rewrite (listrep_isptr l2) by auto.
Intros h l2’ y.
forward.

# 一个 VST-Floyd 验证的例子

\\ Given $l_1, l_2, w$ and $v$. Assume $v \neq$ NULL.
\\ $\{l = \mathrm{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ *\ v \rightsquigarrow l_2\}$
\\ $\{\exists h\, l_2'\, y.\ l = \mathrm{rev}(l_1) \cdot [h] \cdot l_2' \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ *\ v \xmapsto[\texttt{list}]{} h, y \ *\ y \rightsquigarrow l_2'\}$

\\ Given $h, l_2'$ and $y$. Assume $l = \mathrm{rev}(l_1) \cdot [h] \cdot l_2'$.
\\ $\{[\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ *\ v \xmapsto[\texttt{list}]{} h, y \ *\ y \rightsquigarrow l_2'\}$
**t = v→ tail;**
\\ $\{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ *\ v \xmapsto[\texttt{list}]{} h, y \ *\ y \rightsquigarrow l_2'\}$
**v→ tail = w;**
\\ $\{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ *\ v \xmapsto[\texttt{list}]{} h, w \ *\ y \rightsquigarrow l_2'\}$
**w = v;**
**v = t;**
\\ $\{\exists\, l_1\, l_2\, w\, v.\ l = \mathrm{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ *\ v \rightsquigarrow l_2\}$

forward.

# 一个 VST-Floyd 验证的例子

$\backslash\backslash$ Given $l_1, l_2, w$ and $v$. Assume $v \neq \text{NULL}$.
$\backslash\backslash \{l = \text{rev}(l_1) \cdot l_2 \ \wedge \ [\![\mathbf{w}]\!] = w \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \rightsquigarrow l_2\}$
$\backslash\backslash \{\exists h \ l'_2 \ y. \ l = \text{rev}(l_1) \cdot [h] \cdot l'_2 \ \wedge \ [\![\mathbf{w}]\!] = w \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l'_2\}$

$\backslash\backslash$ Given $h, l'_2$ and $y$. Assume $l = \text{rev}(l_1) \cdot [h] \cdot l'_2$.
$\backslash\backslash \{[\![\mathbf{w}]\!] = w \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l'_2\}$
**t = v→tail;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge \ [\![\mathbf{w}]\!] = w \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l'_2\}$
**v→tail = w;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge \ [\![\mathbf{w}]\!] = w \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, w \ * \ y \rightsquigarrow l'_2\}$
**w = v;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge \ [\![\mathbf{w}]\!] = v \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \xmapsto[\texttt{list}]{} h, w \ * \ y \rightsquigarrow l'_2\}$
**v = t;**
$\backslash\backslash \{\exists \ l_1 \ l_2 \ w \ v. \ l = \text{rev}(l_1) \cdot l_2 \ \wedge \ [\![\mathbf{w}]\!] = w \ \wedge \ [\![\mathbf{v}]\!] = v \ \wedge \ w \rightsquigarrow l_1 \ * \ v \rightsquigarrow l_2\}$

forward.

# 一个 VST-Floyd 验证的例子

$\backslash\backslash$ Given $l_1, l_2, w$ and $v$. Assume $v \neq \mathrm{NULL}$.

$\backslash\backslash\ \{l = \mathrm{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \leadsto l_2\}$

$\backslash\backslash\ \{\exists h\, l_2'\, y.\ l = \mathrm{rev}(l_1) \cdot [h] \cdot l_2' \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \underset{\mathtt{list}}{\longmapsto} h, y \ *\ y \leadsto l_2'\}$

$\backslash\backslash$ Given $h, l_2'$ and $y$. Assume $l = \mathrm{rev}(l_1) \cdot [h] \cdot l_2'$.

$\backslash\backslash\ \{[\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \underset{\mathtt{list}}{\longmapsto} h, y \ *\ y \leadsto l_2'\}$

**t = v→ tail;**

$\backslash\backslash\ \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \underset{\mathtt{list}}{\longmapsto} h, y \ *\ y \leadsto l_2'\}$

**v→ tail = w;**

$\backslash\backslash\ \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \underset{\mathtt{list}}{\longmapsto} h, w \ *\ y \leadsto l_2'\}$

**w = v;**

$\backslash\backslash\ \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = v \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \underset{\mathtt{list}}{\longmapsto} h, w \ *\ y \leadsto l_2'\}$

**v = t;**

$\backslash\backslash\ \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = v \ \wedge\ [\![\mathbf{v}]\!] = y \ \wedge\ w \leadsto l_1 \ *\ v \underset{\mathtt{list}}{\longmapsto} h, w \ *\ y \leadsto l_2'\}$

$\backslash\backslash\ \{\exists\, l_1\, l_2\, w\, v.\ l = \mathrm{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \leadsto l_1 \ *\ v \leadsto l_2\}$

forward.

# 一个 VST-Floyd 验证的例子

$\backslash\backslash$ Given $l_1, l_2, w$ and $v$. Assume $v \neq \text{NULL}$.
$\backslash\backslash \{l = \text{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \rightsquigarrow l_2\}$
$\backslash\backslash \{\exists h\, l_2'\, y.\, l = \text{rev}(l_1) \cdot [h] \cdot l_2' \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xrightarrow[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l_2'\}$

$\backslash\backslash$ Given $h, l_2'$ and $y$. Assume $l = \text{rev}(l_1) \cdot [h] \cdot l_2'$.
$\backslash\backslash \{[\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xrightarrow[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l_2'\}$
**t = v→ tail;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xrightarrow[\texttt{list}]{} h, y \ * \ y \rightsquigarrow l_2'\}$
**v→ tail = w;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xrightarrow[\texttt{list}]{} h, w \ * \ y \rightsquigarrow l_2'\}$
**w = v;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = v \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xrightarrow[\texttt{list}]{} h, w \ * \ y \rightsquigarrow l_2'\}$
**v = t;**
$\backslash\backslash \{[\![\mathbf{t}]\!] = y \ \wedge\ [\![\mathbf{w}]\!] = v \ \wedge\ [\![\mathbf{v}]\!] = y \ \wedge\ w \rightsquigarrow l_1 \ * \ v \xrightarrow[\texttt{list}]{} h, w \ * \ y \rightsquigarrow l_2'\}$
$\backslash\backslash \{[\![\mathbf{w}]\!] = v \ \wedge\ [\![\mathbf{v}]\!] = y \ \wedge\ v \rightsquigarrow [h] \cdot l_1 \ * \ y \rightsquigarrow l_2'\}$
$\backslash\backslash \{\exists\, l_1\, l_2\, w\, v.\, l = \text{rev}(l_1) \cdot l_2 \ \wedge\ [\![\mathbf{w}]\!] = w \ \wedge\ [\![\mathbf{v}]\!] = v \ \wedge\ w \rightsquigarrow l_1 \ * \ v \rightsquigarrow l_2\}$

```
entailer!. Exists (h::l1,l2',v,y).
entailer!.
 + simpl. rewrite app_ass. auto.
 + unfold listrep at 3; fold listrep. Exists w. entailer!.
```

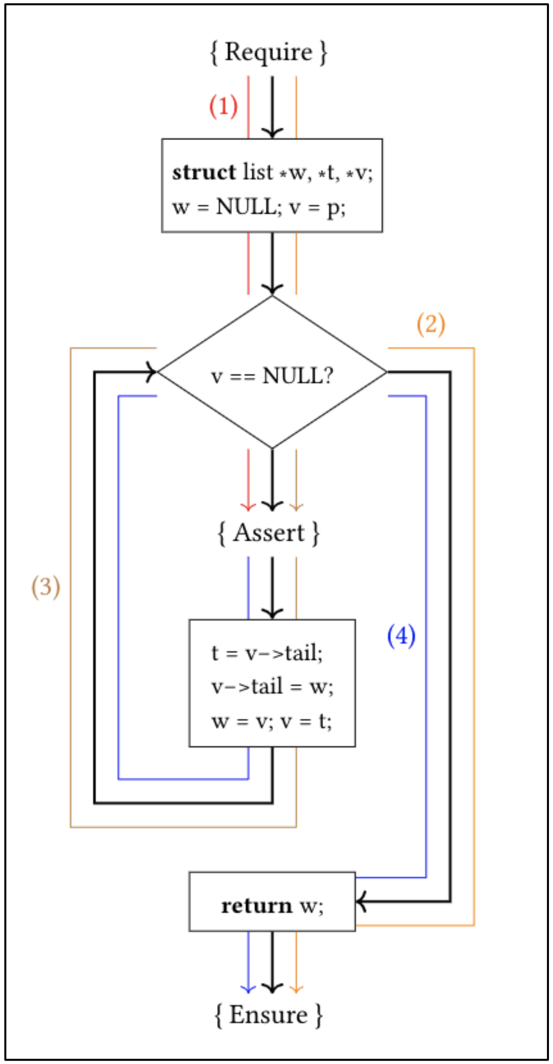
# VST-A 简介

- VST-A的使用
  - 使用VST-Floyd证明指令 → 使用C程序断言描述证明
  - 使用VST-Floyd canonical form → 使用更简明易读的断言语言
- VST-A的理论基础
  - 如果所有控制流图上的straight line Hoare triple都可证，
    那么完整程序的Hoare triple就可证
    - Straight line：无分支、循环、函数调用

# VST-A 的例子

```
struct list {unsigned head; struct list *tail;};

struct list *reverse (struct list *p) {
  /*@ With sh l
      Require
        writable_share(sh) && listrep(sh, l, p)
      Ensure
        listrep(sh, rev(l), __return)
  */
  struct list *w, *t, *v;
  w = (void *) 0;
  v = p;
  while (v) {
    /*@ Assert
          exists l1 x l2 u,
            writable_share(sh) &&
            l == rev(l1) ++ x :: l2 &&
            data_at_list(sh, {x, u}, v) *
            listrep(sh, l1, w) * listrep(sh, l2, u)
     */
    t = v->tail;
    v->tail = w;
    w = v;
    v = t;
  }
  return w;
}
```



# Incorrectness Logic

- Peter W. O’Hearn指示团队用分离逻辑开发Infer
- 但发现团队不听他的，反而写了很多启发式规则查错
- 这些启发式规则没有理论基础，但居然很有效
- 逻辑学背景的O’Hearn不能容忍这件事，决定要做一套新的逻辑

# 下近似三元组

- [presumption]code[result]

- 对任意满足result的状态s，一定存在一个满足presumption的状态，使得执行code之后到达s
  - 即：后条件覆盖状态是后状态的子集

- 对比：
  - 霍尔三元组：上近似三元组
  - 后条件覆盖状态是后状态的超集

# 例子

```
/* presumes: [z==11] */
  if (x is even) {
    if (y is odd) {
       z=42;
  }  }
/* achieves: [z==42 && (x is even) && (y is odd) ] */
```

# 带返回类型的下近似三元组

- [presumption]code[re:result]
  - re可以是ok或er或者其他类型
  - ok表示正常返回
  - er表示程序执行了error()

```
void foo(char* str)
/*  presumes: [*str[]==s]
    achieves: [er: *str[]==s && length(s) > 16 ] */
  { char buf[16];
    strcpy(buf,str);
  }

int main(int argc, char *argv[])
  { foo(argv[1]); }
```

# 一般性规则

*Empty under-approximates*

$$[p]C[\epsilon{:}\, false]$$

*Consequence*

$$\frac{p' \Leftarrow p \quad [p]C[\epsilon{:}\, q] \quad q \Leftarrow q'}{[p']C[\epsilon{:}\, q']}$$

*Disjunction*

$$\frac{[p_1]C[\epsilon{:}\, q_1] \quad [p_2]C[\epsilon{:}\, q_2]}{[p_1 \vee p_2]C[\epsilon{:}\, q_1 \vee q_2]}$$

*Unit*

$$[p]\mathtt{skip}[ok{:}\, p][er{:}\, false]$$

*Sequencing* (short-circuit)

$$\frac{[p]C_1[er{:}\, r]}{[p]C_1; C_2[er{:}\, r]}$$

*Sequencing* (normal)

$$\frac{[p]C_1[ok{:}\, q] \quad [q]C_2[\epsilon{:}\, r]}{[p]C_1; C_2[\epsilon{:}\, r]}$$

*Iterate zero*

$$[p]C^\star[ok{:}\, p]$$

*Iterate non-zero*

$$\frac{[p]C^\star; C[\epsilon{:}\, q]}{[p]C^\star[\epsilon{:}\, q]}$$

*Backwards Variant* (where $n$ fresh)

$$\frac{[p(n) \wedge nat(n)]C[ok{:}\, p(n+1) \wedge nat(n)]}{[p(0)]C^\star[ok{:}\, \exists n.p(n) \wedge nat(n)]}$$

*Choice* (where $i = 1$ or $2$)

$$\frac{[p]C_i[\epsilon{:}\, q]}{[p]C_1 + C_2[\epsilon{:}\, q]}$$

*Error*

$$[p]\mathtt{error}()[ok{:}\, false][er{:}\, p]$$

*Assume*

$$[p]\mathtt{assume}\ B[ok{:}\, p \wedge B][er{:}\, false]$$

$$\mathtt{while}\ B\ \mathtt{do}\ C \quad =_{def} \quad (\mathtt{assume}(B); C)^\star; \mathtt{assume}(\neg B)$$

$$\mathtt{if}\ B\ \mathtt{then}\ C\ \mathtt{else}\ C' \quad =_{def} \quad (\mathtt{assume}(B); C) + (\mathtt{assume}(\neg B); C')$$

$$\mathtt{assert}(B) \quad =_{def} \quad \mathtt{assume}(B) + (\mathtt{assume}(\neg B); \mathtt{error}())$$

# 变量规则

*Assignment*

$$[p]x = e[ok\colon \exists x'.p[x'/x] \wedge x = e[x'/x]][er\colon \mathit{false}]$$

*Constancy*

$$\frac{[p]C[\epsilon\colon q]}{[p \wedge f]C[\epsilon\colon q \wedge f]} \quad \mathit{Mod}(C) \cap \mathit{Free}(f) = \emptyset$$

*Substitution I*

$$\frac{[p]C[\epsilon\colon q]}{([p]C[\epsilon\colon q])(e/x)} \quad \big(\mathit{Free}(e) \cup \{x\}\big) \cap \mathit{Free}(C) = \emptyset$$

*Substitution II*

$$\frac{[p]C[\epsilon\colon q]}{([p]C[\epsilon\colon q])(y/x)} \quad y \notin \mathit{Free}(p, C, q)$$

Mod(C)：语句C可以修改的变量集合
Free(f)：f中的自由变量

原霍尔逻辑规则不保证下近似
比如：{42=y}x=42{x=y}
但x=y=3也满足后条件

# 例子

```
/* presumes: [z==11] */
  if (x is even) {
    if (y is odd) {
       z=42;
  }  }
/* achieves: [z==42 && (x is even) && (y is odd) ] */
```

- 展开if定义，得到
  - assume(x is even); (assume (y is odd); z=42 + assume (¬ y is odd); skip) + assume(¬ x is even); skip
- 根据choice，选择
  - assume(x is even); assume (y is odd); z=42
- 根据assume，得到：
  - [True]assume(x is even)[ok: x is even]
  - [x is even]assume(y is odd)[ok: x is even /\ y is odd]
- 根据assignment，得到：
  - [x is even /\ y is odd]z=42[x is even /\ y is odd /\ z==42]
- 根据sequencing，原命题得证

# 性质

- 正确性：所有推出的三元组都是下近似三元组
- 完整性：所有有限支撑（finitely-supported）下近似三元组都能推出
  - 有限支撑：一个谓词只依赖于有限个变量