Separation Logic

separation logic 是 hoare logic 对指针的扩展

Programming Language

为 command 扩展了

$$
\begin{aligned}
c ::= & \ \ldots \\
& \mid x := \mathbf{cons}(e_1, e_2, \ldots, e_n) \\
& \mid \mathbf{dispose}(e) \\
& \mid x := [e] \\
& \mid [e] := e
\end{aligned}
$$

添加的操作为 allocation，deallocation，lookup 和 mutation

状态由两部分组成，store 是变量到值的映射，而 heap 是地址到值的 partial 映射

如果查找和去配超出范围的话程序会产生 fault

新的 command 的操作语义为

$$\frac{h([\![e]\!]_{intexp}\, s) = n}{(x := [e], (s, h)) \longrightarrow (\mathbf{skip}, (s\{x \leadsto n\}, h))}$$

$$\frac{[\![e]\!]_{intexp}\, s \notin \mathrm{dom}(h)}{(x := [e], (s, h)) \longrightarrow \mathbf{abort}}$$

$$\frac{[\![e]\!]_{intexp}\, s = \ell \quad \ell \in \mathrm{dom}(h)}{([e] := e', (s, h)) \longrightarrow (\mathbf{skip}, (s, h\{\ell \leadsto [\![e']\!]_{intexp}\, s\}))}$$

$$\frac{[\![e]\!]_{intexp}\, s \notin \mathrm{dom}(h)}{([e] := e', (s, h)) \longrightarrow \mathbf{abort}}$$

$$\frac{[\![e_1]\!]_{intexp}\, s = n_1 \quad [\![e_2]\!]_{intexp}\, s = n_2 \quad \{\ell, \ell + 1\} \cap \mathrm{dom}(h) = \emptyset}{(x := \mathbf{cons}(e_1, e_2), (s, h)) \longrightarrow (\mathbf{skip}, (s\{x \leadsto \ell\}, h\{\ell \leadsto n_1, \ell{+}1 \leadsto n_2\}))}$$

Assertions

Basis

除了标准的谓词逻辑：$\wedge, \vee, \neg, \implies, \forall, \exists$，新增了

- $\mathbf{emp}$：堆为空
- $e \mapsto e'$：堆有一个 cell，地址为 e 内容为 e'
- $p_1 * p_2$：堆可以被分为两个 disjoint 的部分，其中一部分满足 p_1 另一部分满足 p_2
- $p_1 - *p_2$：如果堆扩展一个 disjoint 的满足 p_1 的部分，则扩展后的堆满足 p_2

除此之外还可以定义一些缩写

- $e \mapsto - = \exists x'. e \mapsto x'$，其中 x' 不是 e 中的自由变量（i.e. e 的值不依赖于 x'）
- $e \hookrightarrow e' = e \mapsto e' * \mathbf{true}$
- $e \mapsto e_1, \dots, e_n = e \mapsto e_1 * \dots e + n - 1 \mapsto e_n$
- $e \hookrightarrow e_1, \dots, e_n = e \hookrightarrow e_1 * \dots * e + n - 1 \hookrightarrow e_n$

需要注意 $\wedge$ 和 $*$ 的区别，$\wedge$ 要求是同一个堆，而 $*$ 并不要求这一点

引入一种新的函数记号：$[x_1 : y_1 \mid x_2 : y_2 \mid \cdots \mid x_n : y_n]$，表示将 x_i 映射到 y_i，也可以用这种记号扩展现有函数 $[f \mid x_1 : y_1 \mid \cdots \mid x_n : y_n]$

引入 $\perp$ 表示两个堆 disjoint，$h_1 \cdot h_2$ 则是两个 disjoint 的堆的并

separation logic 的语义定义也与 hoare logic 类似，$s, h \models p$ 表示 p 在 状态 s，堆 h 下为真，其中 p 的自由变量都在 s 的 domain 内

如果对于任意 s, h 均有 $s, h \models p$ 则称 p 为有效，如果只是对于部分 s, h 有 $s, h \models p$ 则称 p 可满足

Inference Rules

需要注意一个区别

$$\frac{p}{q}$$

sound 表示如果 p 有效则 q 有效，而

$$\frac{}{p \implies q}$$

sound 表示 $p \implies q$ 有效

举一个例子就是 $\frac{p}{\forall v.p}$ sound，而 $\frac{}{p \implies \forall v.p}$ not sound（考虑 p 为 $x = 0$）

新引入的推导规则有

$$
\begin{aligned}
p_0 * p_1 &\Leftrightarrow p_1 * p_0 \\
(p_0 * p_1) * p_2 &\Leftrightarrow p_0 * (p_1 * p_2) \\
p * \mathbf{emp} &\Leftrightarrow p \\
(p_0 \vee p_1) * q &\Leftrightarrow (p_0 * q) \vee (p_1 * q) \\
(p_0 \wedge p_1) * q &\Rightarrow (p_0 * q) \wedge (p_1 * q) \\
(\exists x.\ p_0) * p_1 &\Leftrightarrow \exists x.\ (p_0 * p_1) \quad \text{when } x \text{ not free in } p_1 \\
(\forall x.\ p_0) * p_1 &\Rightarrow \forall x.\ (p_0 * p_1) \quad \text{when } x \text{ not free in } p_1
\end{aligned}
$$

$$
\frac{p_0 \Rightarrow p_1 \qquad q_0 \Rightarrow q_1}{p_0 * q_0 \Rightarrow p_1 * q_1} \quad \text{(monotonicity)}
$$

$$
\frac{p_0 * p_1 \Rightarrow p_2}{p_0 \Rightarrow (p_1 -\!\!* \, p_2)} \quad \text{(currying)} \qquad \frac{p_0 \Rightarrow (p_1 -\!\!* \, p_2)}{p_0 * p_1 \Rightarrow p_2.} \quad \text{(decurrying)}
$$

需要注意有两个公理并不 sound

$$
\begin{aligned}
p &\implies p * p \\
p * q &\implies q
\end{aligned}
$$

反例都是含有 $\mapsto$ 的情况

还有一些关于 $\mapsto$ 的公理

$$
\begin{aligned}
e_1 \mapsto e_1' \wedge e_2 \mapsto e_2' &\Leftrightarrow e_1 \mapsto e_1' \wedge e_1 = e_2 \wedge e_1' = e_2' \\
e_1 \hookrightarrow e_1' * e_2 \hookrightarrow e_2' &\Rightarrow e_1 \neq e_2 \\
\mathbf{emp} &\Leftrightarrow \forall x.\ \neg(x \hookrightarrow -) \\
(e \hookrightarrow e') \wedge p &\Rightarrow (e \mapsto e') * ((e \mapsto e') -\!\!* \, p).
\end{aligned}
$$

Assertion classes

pure assertion：对于任意的 store s 和 heap h, h' 都有

$$
s, h \models p \iff s, h' \models p
$$

即 assertion 与堆无关。以下是一系列满足 pure assertion 的例子

$$
\begin{array}{rl}
p_0 \wedge p_1 \Rightarrow p_0 * p_1 & \text{when } p_0 \text{ or } p_1 \text{ is pure} \\
p_0 * p_1 \Rightarrow p_0 \wedge p_1 & \text{when } p_0 \text{ and } p_1 \text{ are pure} \\
(p \wedge q) * r \Leftrightarrow (p * r) \wedge q & \text{when } q \text{ is pure} \\
(p_0 -\!\!* \, p_1) \Rightarrow (p_0 \Rightarrow p_1) & \text{when } p_0 \text{ is pure} \\
(p_0 \Rightarrow p_1) \Rightarrow (p_0 -\!\!* \, p_1) & \text{when } p_0 \text{ and } p_1 \text{ are pure.}
\end{array}
$$

strictly exact assertion：对于任意的 store s 和 heap h, h' 都有

$$s, h \models p \wedge s, h' \models p \implies h = h'$$

即只对唯一的 s, h 组合满足。以下是一系列满足 strictly exact assertion 的例子

$\mathbf{emp}$.

$e \mapsto e'$.

$p * q$, when p and q are strictly exact.

$p \wedge q$, when p or q is strictly exact.

p, when $p \Rightarrow q$ is valid and q is strictly exact.

如果 q strictly exact，有

$$((q * \mathbf{true}) \wedge p) \implies (q * (q - *p))$$

precise assertion：对于任意的 s, h，最多只有一个 $h' \subseteq h$ 满足

$$s, h' \models q$$

以下是一系列满足 precise assertion 的例子

Strictly exact assertions.

$e \mapsto -$.

$p * q$, when p and q are precise.

$p \wedge q$, when p or q is precise.

p, when $p \Rightarrow q$ is valid and q is precise.

list α e and $\exists \alpha$. list α e.

有两个推论只在一个方向成立

- $(p_0 \wedge p_1) * q \implies (p_0 * q) \wedge (p_1 * q)$

- $(\forall x.\, p) * q \implies \forall x.\, (p * q)$，其中 x 不是 q 的自由变量

但如果 q 是 precise，则另一方向也能成立

intuitionistic assertion：对于任意的 store s 和 heap h, h' 都有

$$h \subseteq h' \wedge s, h \models i \implies s, h' \models i$$

对于 intuitionistic assertion i, i' 和任意 assertion p，以下 assertion 是 intuitionistic

Any pure assertion	$p * i$
$p -\!\!* \; i$	$i -\!\!* \; p$
$i \wedge i'$	$i \vee i'$
$\forall v.\ i$	$\exists v.\ i$

Specifications and Inference Rules

Specification

specification 同样分为 partial correctness 和 total correctness

与 hoare logic 的区别在于 specification 如果产生了 memory fault 则视为不满足，这是由内存分配的不确定性导致的（同样的分配并不能保证每次的起始地址一致），而 specification 要能在所有的执行路径上对于 store 和 heap 同时满足

hoare logic 的一条规则在 separation logic 中不适用了

$$\frac{\{p\}c\{q\}}{\{p \wedge r\}c\{q \wedge r\}}$$

如

$$\frac{\{x \mapsto -\}[x] := 4\{x \mapsto 4\}}{\{x \mapsto - \wedge y \mapsto 3\}[x] := 4\{x \mapsto 4 \wedge y \mapsto 3\}}$$

在 $x = y$ 的情况下不成立

相对应的，separation logic 有 frame rule

$$\frac{\{p\}c\{q\}}{\{p * r\}c\{q * r\}}$$

需要没有 r 中的自由变量被 c 修改。frame rule 的关键就是只规约程序访问到的内存，没访问到的保持不变（称为 heap 的 local reasoning）

- 程序指令实际访问到的 variable 和 heap cell 称为其 footprint
- 如果 $\{p\}c\{q\}$ valid，则 p 断言堆中含有所有 c 的 footprint 中的 cell（除了被 c 新分配的）
- 如果 p 断言堆**只**含有 c 的 footprint，则 $\{p\}c\{q\}$ 为 local specification
- 如果语句 c' 包含了 c，则其 footprint 可能更大，此时就需要 frame rule 来扩充 heap

frame rule 的 soundness 与语言的语义有关

定义：如果在状态 s, h 开始，指令 c 的执行不会 abort，则称 c 在 s, h 是 safe 的。如果在状态 s, h 开始，指令 c 的所有执行都能终止不会 abort，则称 c 在 s, h must terminate normally

则可以定义

- safety monotonicity：如果 $\hat{h} \subseteq h$ 且 c 在 $s, h - \hat{h}$ safe，则 c 在 s, h safe；如果 $\hat{h} \subseteq h$ 且 c 在 $s, h - \hat{h}$ must terminate normally，则 c 在 s, h must terminate normally
- frame property：如果 $\hat{h} \subseteq h$，且 c 在 $s, h - \hat{h}$ safe，且某些 c 的从 s, h 开始的执行在 s', h' terminate normally，则有 $\hat{h} \subseteq h'$ 且某些 c 的从 $s, h - \hat{h}$ 开始的执行在 $s', h' - \hat{h}$ terminate normally

用图表示的话就是

$$
\begin{tikzcd}
s,h-\hat{h} \arrow[d, dashed, "c"] & s,h \arrow[l, "\subseteq"'] \arrow[r, "\supseteq"] \arrow[d, "c"] & \hat{h} \\
s',h'-\hat{h} & s',h' \arrow[l, dashed, "\subseteq"'] \arrow[ur, dashed, "\supseteq"'] &
\end{tikzcd}
$$

如果一个语言同时满足 safety monotonicity 和 frame property，则 frame rule 对于 partial 和 total correctness 均为 sound

safety monotonicity 和 frame property 共同称为 locality

Inference Rules

对于 mutation 有三个版本的规则：局部（MUL），全局（MUG），前向（MUBR）

$$\frac{}{\{e \mapsto -\}[e] := e'\{e \mapsto e'\}}$$

$$\frac{}{\{(e \mapsto -) * r\}[e] := e'\{(e \mapsto e') * r\}}$$

$$\frac{}{\{(e \mapsto -) * ((e \mapsto e') -\!* p)\}[e] := e'\{p\}}$$

对于 deallocation 有两个版本的规则：局部（DISL）和全局（DISBR）

$$\frac{}{\{e \mapsto -\}\textbf{dispose}\ e\{\textbf{emp}\}}$$

$$\frac{}{\{(e \mapsto -) * r\}\textbf{dispose}\ e\{r\}}$$

对于 allocation 和 lookup，一般赋值的规则并不适用。

不覆写的 allocation 有两个版本的规则：局部（CONSNOL）和全局（CONSNOG）

如果用 $\overline{e}$ 表示 $e_1, \ldots, e_n$，则有

$$\frac{}{\{\textbf{emp}\}v := \textbf{cons}(\overline{e})\{v \mapsto \overline{e}\}, v \notin FV(\overline{e})}$$

$$\frac{}{\{r\}v := \textbf{cons}(\overline{e})\{(v \mapsto \overline{e}) * r\}, v \notin FV(\overline{e}, r)}$$

一般的 allocation 有三个版本的规则：

局部（CONSL）

$$\frac{}{\{v = v' \wedge \textbf{emp}\}v := \textbf{cons}(\overline{e})\{v \mapsto \overline{e}'\}}$$

其中 $\overline{e}'$ 表示 $\overline{e}[v'/v]$

全局（CONSG）

$$\frac{}{\{r\}v := \textbf{cons}(\overline{e})\{\exists v'.\,(v \mapsto \overline{e}') * r'\}}$$

其中 $v' \notin FV(\overline{e}, r)$，$\overline{e}'$ 表示 $\overline{e}[v'/v]$，r' 表示 $r[v'/v]$

前向（CONSBR）

$$\frac{}{\{\forall v''.\,(v'' \mapsto \overline{e}) -\!* p''\}v := \textbf{cons}(\overline{e})\{p\}}$$

其中 $v'' \notin FV(\overline{e}, p)$，$p''$ 表示 $p[v''/v]$

不覆写的 lookup 有两个版本的规则：

局部（LKNOL）

$$\frac{}{\{e \mapsto v''\}v := [e]\{v = v'' \wedge (e \mapsto v)\}, v \notin FV(e)}$$

全局（LKNOG）

$$\frac{}{\{\exists v''.\,(e \mapsto v'') * p''\}v := [e]\{(e \mapsto v) * p\}}$$

其中 $v \notin FV(e), v'' \notin FV(e) \cup (FV(p) - \{v\})$，$p''$ 表示 $p[v''/v]$

而一般的 lookup 有四种规则

局部（LKL）

$$\frac{}{\{v = v' \wedge (e \mapsto v'')\} v := [e] \{v = v'' \wedge (e' \mapsto v)\}}$$

其中 e' 表示 $e[v'/v]$

全局（LKG）

$$\frac{}{\{\exists v''.\,(e \mapsto v'') * (r[v'/v])\} v := [e] \{\exists v'.\,(e' \mapsto v) * (r[v''/v])\}}$$

其中 $v', v'' \notin FV(e), v \notin FV(r)$，$e'$ 表示 $e[v'/v]$

以及两种前向规则

$$\frac{}{\{\exists v''.\,(e \mapsto v'') * ((e \mapsto v'') -\!* p'')\} v := [e] \{p\}}$$

$$\frac{}{\{\exists v''.\,(e \hookrightarrow v'') \wedge p''\} v := [e] \{p\}}$$

其中 $v'' \notin FV(e) \cup (FV(p) - \{v\})$，$p''$ 表示 $p[v''/v]$