# The Rule of Constancy (Derived Frame Rule)

- **The following derived rule is used on the next slide**

**The rule of constancy**

$$\frac{\vdash\ \{P\}\ C\ \{Q\}}{\vdash\ \{P \wedge R\}\ C\ \{Q \wedge R\}}$$

**where no variable assigned to in $C$ occurs in $R$**

- **Outline of derivation**
  - **prove $\{R\}\ C\ \{R\}$ by induction on $C$**
  - **then use Specification Conjunction**

- **Assume $C$ doesn't modify $V$ and $\vdash\ \{P\}\ C\ \{P[V{+}1/V]\}$ then:**

$$\vdash\ \{P \wedge V{=}v\}\ C\ \{P[V{+}1/V] \wedge V{=}v\} \qquad \text{(assumption + constancy rule)}$$
$$\vdash\ \{P[V{+}1/V] \wedge V{=}v\}\ V := V{+}1\ \{P \wedge V{=}v{+}1\} \qquad \text{(assign. ax + pre. streng.)}$$
$$\vdash\ \{P \wedge V{=}v\}\ C;\ V := V{+}1\ \{P \wedge V{=}v{+}1\} \qquad \text{(sequencing)}$$

- **So $C;\ V := V{+}1$ has $P$ as an invariant and increments $V$**

# Towards the FOR-Rule

- **If $e_1 \leq e_2$ the FOR-command is equivalent to:**

$$\texttt{BEGIN VAR } V\texttt{;}\ V\texttt{:=}e_1\texttt{;}\ \ldots\ C\ \texttt{;}\ V\texttt{:=}V{+}1\texttt{;}\ \ldots\ V\texttt{:=}e_2\texttt{;}\ C\ \texttt{END}$$

- **Assume $C$ doesn't modify $V$ and $\vdash\ \{P\}\ C\ \{P[V{+}1/V]\}$**

- **Hence:**

$$
\begin{array}{ll}
\vdash\ \{P[e_1/V]\}\ V\texttt{:=}e_1\ \{P \wedge V{=}e_1\} & \text{(assign. ax + pre. streng.)} \\
\vdots & \\
\vdash\ \{P \wedge V{=}v\}\ C\texttt{;}\ V\texttt{:=}V{+}1\ \{P \wedge V{=}v{+}1\} & \text{(last slide; } V = e_1, e_1{+}1, \ldots, e_2{-}1) \\
\vdots & \\
\vdash\ \{P \wedge V{=}v\}\ C\texttt{;}\ V\texttt{:=}V{+}1\ \{P \wedge V{=}e_2{+}1\} & \\
\vdash\ \{P \wedge V{=}e_2\}\ C\ \{P[V{+}1/V] \wedge V{=}e_2\} & \text{(assign. ax + assumption + constancy)} \\
\vdash\ \{P \wedge V{=}e_2\}\ C\ \{P[e_2{+}1/V]\} & \text{(post. weak.)}
\end{array}
$$

- **Hence by the sequencing and block rules**

$$\frac{\vdash \{P\}C\{P[V{+}1/V]\}}{\vdash \{P[e_1/V]\}\texttt{BEGIN VAR } V\texttt{;}V\texttt{:=}e_1\texttt{;}\ \ldots\ C\texttt{;}V\texttt{:=}V{+}1\texttt{;}\ \ldots\ V\texttt{:=}e_2\texttt{;}C\ \texttt{END}\{P[e_2{+}1/V]\}}$$

# The FOR-Rule

- **To rule out the problems that arise when the controlled variable or variables in the bounds expressions, are changed by the body, we simply impose a side condition on the rule that stipulates that it cannot be used in these situations**

**The FOR-rule**

$$\frac{\vdash \{P \wedge (E_1 \leq V) \wedge (V \leq E_2)\}\ C\ \{P[V\texttt{+1}/V]\}}{\vdash \{P[E_1/V] \wedge (E_1 \leq E_2)\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P[E_2\texttt{+1}/V]\}}$$

**where neither $V$, nor any variable occurring in $E_1$ or $E_2$, is assigned to in the command $C$.**

- **Note $(E_1 \leq V) \wedge (V \leq E_2)$ in precondition of rule hypothesis**
  - **added to strengthen rule to allow proofs to use facts about $V$'s range of values**

- **Can be tricky to think up $P$**

# Comment on the FOR-Rule

- **The FOR-rule does not enable anything to be deduced about FOR-commands whose body assigns to variables in the bounds expressions**

- **This precludes such assignments being used if commands are to be reasoned about**

- **Only defining rules of inference for non-tricky uses of constructs motivates writing programs in a perspicuous manner**

- **It is possible to devise a rule that does cope with assignments to variables in bounds expressions**

- **Consider the rule below ($e_1$, $e_2$ are fresh auxiliary variables):**

$$\frac{\vdash \{P \wedge (e_1 \leq V) \wedge (V \leq e_2)\}\ C\ \{P[V\texttt{+1}/V]\}}{\vdash \{P[E_1/V] \wedge (E_1 \leq E_2) \wedge (E_1 = e_1) \wedge (E_2 = e_2)\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P[e_2\texttt{+1}/V]\}}$$

# The `FOR`-axiom ✓

- **To cover the case when $E_2 < E_1$, we need the `FOR`-axiom below**

**The `FOR`-axiom**

$$\vdash\ \{P \wedge (E_2 < E_1)\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P\}$$

- **This says that when $E_2$ is less than $E_1$ the `FOR`-command has no effect**

## Ensuring Soundness

- **It is clear from the discussion of the `FOR`-rule that it is not always straightforward to devise correct rules of inference**

- **It is important that the axioms and rules be sound. There are two approaches to ensure this**

  (i) **define the language by the axioms and rules of the logic**

  (ii) **prove that the logic is sound for the language**

- **Approach (i) is called** *axiomatic semantics*

  - **the idea is to** *define* **the semantics of the language by requiring that it make the axioms and rules of inference true**

  - **it is then up to implementers to ensure that the logic matches the language**

- **Approach (ii) is proving soundness of the logic**

# Axiomatic Semantics

- One snag with axiomatic semantics is that most existing languages have already been defined in some other way
  - usually by informal and ambiguous natural language statements
- The other snag with axiomatic semantics is that by Clarke's Theorem it is known to be impossible to devise relatively complete Floyd-Hoare logics for languages with certain constructs
  - it could be argued that this is not a snag at all but an advantage, because it forces programming languages to be made logically tractable
- An example of a language defined axiomatically is Euclid

7.1. (module rule)

(1) $Q \supset Q0(A/t)$,

(2) $P1\{\textbf{const } K; \textbf{ var } V; S_4\}\ Q4(A/t) \wedge Q$,

(3) $P2(A/t) \wedge Q\ \{S_2\}\ Q2(A/t) \wedge Q$,

(4) $\exists\, g1(P3(A/t) \wedge Q\ \{S_3\}\ Q3(A/t) \wedge g = g1(A, c, d))$,

(5) $\exists\, g(P3(A/t) \wedge Q \supset Q3(A/t))$,

(6) $P6(A/t) \wedge Q\ \{S_6\}\ Q1$,

(7) $P \supset P1(a/c)$,

(8.1) $[Q0(a/c, x/t, x'/t') \supset (P2(x/t, x'/t', a2/x2, e2/c2, a/c) \wedge$
$(Q2(x2\sharp/t, x'/t', a2\sharp/x2, e2/c2, a/c, y2\sharp/y2, a2/x2', y2/y2') \supset$
$R1(x2\sharp/x, a2\sharp/a2, y2\sharp/y2)))\ \{x\,.\,p(a2, e2)\}\ R1 \wedge Q0(a/c, x/t, x'/t')$,

(8.2) $(Q0(a/c, x/t) \supset P3(x/t, a3/c3, a/c)) \supset$
$Q3(x/t, a3/c3, a/c, f(a3, d3)/g) \wedge Q0(a/c, x/t)$,

(8.3) $P1(a/c) \wedge (Q4(x4\sharp/t, x'/t', a/c, y4\sharp/y4, y4/y4') \supset R4(x4\sharp/x, y4\sharp/y4))$
$\{x\,.\,Initially\}\ R4 \wedge Q0(a/c, x/t, x'/t')$,

(8.4) $(Q0(a/c, x/t, x'/t') \supset P6(x/t, x'/t', a/c)) \wedge (Q1(a/c, y6\sharp/y6, y6/y6') \supset$
$R(y6\sharp/y6))\ \{x\,.\,Finally\}\ R]$

$\vdash$

(8.5) $P(x\sharp/x)\ \{x\,.\,Initially;\ S;\ x\,.\,Finally\}\ R(x\sharp/x)$

---

$P\{\textbf{var } x\colon T(a);\ S\}\ R \wedge Q1$

# Array assignments ✓

- **Syntax:** $V(E_1)$:=$E_2$

- **Semantics: the state is changed by assigning the value of the term $E_2$ to the $E_1$-th component of the array variable $V$**

- **Example:** `A(X+1) := A(X)+2`

  - **if the the value of `X` is $x$**

  - **and the value of the $x$-th component of `A` is $n$**

  - **then the value stored in the $(x+1)$-th component of `A` becomes $n+2$**

# Naive Array Assignment Axiom Fails

- **The axiom**

  $$\vdash \ \{P\texttt{[}E_2/A(E_1)\texttt{]}\}\ A(E_1)\texttt{:=}E_2\ \{P\}$$

  **doesn't work**

- **Take** $P \equiv$ '`X=Y` $\wedge$ `A(Y)=0`', $E_1 \equiv$ '`X`', $E_2 \equiv$ '`1`'

  - **since** `A(X)` **does not occur in** $P$
  - **it follows that** $P$`[1/A(X)] = `$P$
  - **hence the axiom yields:** $\vdash$ {`X=Y` $\wedge$ `A(Y)=0`} `A(X):=1` {`X=Y` $\wedge$ `A(Y)=0`}

- **Must take into account possibility that changes to** `A(X)` **may change** `A(Y)`, `A(Z)` **etc**

  - **since** `X` **might equal** `Y`, `Z` **etc (i.e. aliasing)**

- **Related to the** *Frame Problem* **in AI**

# Reasoning About Arrays

- **The naive array assignment axiom**

$$\vdash \ \{P[E_2/A(E_1)]\}\ A(E_1)\texttt{:=}E_2\ \{P\}$$

**does not work: changes to** $A(X)$ **may also change** $A(Y)$**,** $A(Z)$**,** $\ldots$

- **The solution, due to Hoare, is to treat an array assignment**

$$A(E_1)\texttt{:=}E_2$$

**as an ordinary assignment**

$$A\ \texttt{:=}\ A\{E_1 \leftarrow E_2\}$$

**where the term** $A\{E_1 \leftarrow E_2\}$ **denotes an array identical to** $A$**, except that the** $E_1$**-th component is changed to have the value** $E_2$

# Array Assignment axiom

- **Array assignment is a special case of ordinary assignment**

$$A := A\{E_1 \leftarrow E_2\}$$

- **Array assignment axiom just ordinary assignment axiom**

$$\vdash\ \{P[A\{E_1 \leftarrow E_2\}/A]\}\ A := A\{E_1 \leftarrow E_2\}\ \{P\}$$

- **Thus:**

**The array assignment axiom**

$$\vdash\ \{P[A\{E_1 \leftarrow E_2\}/A]\}\ A(E_1) := E_2\ \{P\}$$

**Where $A$ is an array variable, $E_1$ is an integer valued expression, $P$ is any statement and the notation $A\{E_1 \leftarrow E_2\}$ denotes the array identical to $A$, except that $A(E_1) = E_2$.**

# Array Axioms ✓

- **In order to reason about arrays, the following axioms, which define the meaning of the notation $A\{E_1 \leftarrow E_2\}$, are needed**

**The array axioms**

$$\vdash \; A\{E_1 \leftarrow E_2\}(E_1) \; = \; E_2$$

$$\vdash \; E_1 \neq E_3 \;\; \Rightarrow \;\; A\{E_1 \leftarrow E_2\}(E_3) \; = \; A(E_3)$$

- **Second of these is a *Frame Axiom***
  - **don't confuse with Frame Rule of Separation Logic (later)**
  - **"frame" is a rather overloaded word!**

# New Topic: Separation logic ✓

- **One of several competing methods for reasoning about pointers**
- **Details took 30 years to evolve**
- **Shape predicates due to Rod Burstall in the 1970s**
- **Separation logic: by O'Hearn, Reynolds and Yang around 2000**
- **Several partially successful attempts before separation logic**
- **Very active research area**
  - **QMUL, UCL, Cambridge, Harvard, Princeton, Yale**
  - **Microsoft**
  - **startup Monoidics bought by Facebook**

# Pointers and the state

- **So far the state just determined the values of variables**
  - **values assumed to be numbers**
  - **preconditions and postconditions are first-order logic statements**
  - **state same as a valuation** $s : \mathit{Var} \to \mathit{Val}$
- **To model pointers – e.g. as in `C` – add** *heap* **to state**
  - **heap maps** *locations* **(pointers) to their contents**
  - *store* **maps variables to values (previously called state)**
  - **contents of locations can be locations or values**

$$\mathtt{X} \underset{\mathit{store}}{\mapsto} l_1 \underset{\mathit{heap}}{\mapsto} l_2 \underset{\mathit{heap}}{\mapsto} v$$

**Heap semantics**

$\mathit{Store} = \mathit{Var} \to \mathit{Val}$ (assume $\mathit{Num} \subseteq \mathit{Val}$, $\mathtt{nil} \in \mathit{Val}$ and $\mathtt{nil} \notin \mathit{Num}$)
$\mathit{Heap} = \mathit{Num} \rightharpoonup_{\mathit{fin}} \mathit{Val}$
$\mathit{State} = \mathit{Store} \times \mathit{Heap}$

- **Note: store also called** *stack* **or** *environment*; *heap* **also called** *store*

# Adding pointer operations to our language ✓

**Expressions:**

$E ::= \quad N \ | \ V \ | \ E_1 + E_2 \ | \ E_1 - E_2 \ | \ E_1 \times \ E_2 \ | \ \ldots$

**Boolean expressions:**

$B ::= \quad \mathtt{T} \ | \ \mathtt{F} \ | \ E_1 {=} E_2 \ | \ E_1 \leq E_2 \ | \ \ldots$

**commands:**

| | | |
|---|---|---|
| $C ::=$ | $V$ `:=` $E$ | **value assignments** |
| $\mid$ | $V$`:=[`$E$`]` | **fetch assignments** |
| $\mid$ | `[`$E_1$`]:=`$E_2$ | **heap assignments (heap mutation)** |
| $\mid$ | $V$`:=cons`$(E_1, \ldots, E_n)$ | **allocation assignments** |
| $\mid$ | `dispose(`$E$`)` | **pointer disposal** |
| $\mid$ | $C_1$ `;` $C_2$ | **sequences** |
| $\mid$ | `IF` $B$ `THEN` $C_1$ `ELSE` $C_2$ | **conditionals** |
| $\mid$ | `WHILE` $B$ `DO` $C$ | **while commands** |

# Pointer manipulation constructs and faulting ✓

- Commands executed in a state $(s, h)$
- Reading, writing or disposing pointers might *fault*
- Fetch assignments: $V$ `:=[`$E$`]`
  - evaluate $E$ to get a location $l$
  - fault if $l$ is not in the heap
  - otherwise assign contents of $l$ in heap to the variable $V$
- Heap assignments: `[`$E_1$`]:=`$E_2$
  - evaluate $E_1$ to get a location $l$
  - fault if the $l$ is not in the heap
  - otherwise store the value of $E_2$ as the new contents of $l$ in the heap
- Pointer disposal: `dispose(`$E$`)`
  - evaluate $E$ to get a pointer $l$ (a number)
  - fault if $l$ is not in the heap
  - otherwise remove $l$ from the heap

# Allocation assignments ✓

- **Allocation assignments:** $V$:=`cons`$(E_1, \ldots, E_n)$
  - **choose** $n$ **consecutive locations that are not in the heap, say** $l, l{+}1, \ldots$
  - **extend the heap by adding** $l, l{+}1, \ldots$ **to it**
  - **assign** $l$ **to the variable** $V$ **in the store**
  - **make the values of** $E_1, E_2, \ldots$ **be the new contents of** $l, l{+}1, \ldots$ **in the heap**
- **Allocation assignments never fault**
- **Allocation assignments are** *non-deterministic*
  - **any suitable** $l, l{+}1, \ldots$ **not in the heap can be chosen**
  - **always exists because the heap is finite**

# Example (different from the background reading) ✓

`X:=cons(0,1,2); [X]:=Y+1; [X+1]:=Z; Y:=[Y+Z]`

- `X:=cons(0,1,2)` **allocates three new pointers, say** $l$, $l{+}1$, $l{+}2$
  - $l$ **initialised with contents** `0`, $l{+}1$ **with** `1` **and** $l{+}2$ **with** `2`
  - **variable** `X` **is assigned** $l$ **as its value in store**
- `[X]:=Y+1` **changes the contents of** $l$
  - $l$ **gets value of** `Y+1` **as new contents in heap**
- `[X+1]:=Z` **changes the contents of** $l{+}1$
  - $l{+}1$ **gets the value of** `Z` **as new contents in heap**
- `Y:=[Y+Z]` **changes the value of** `Y` **in the store**
  - `Y` **assigned in the store the contents of** `Y+Z` **in the heap**
  - **faults if the value of** `Y+Z` **is not in the heap**

# Local Reasoning and Separation Logic ✓

- **Want to just reason about just those locations being modified**
  - assume all other locations unchanged
- **Solution: separation logic**
  - small and forward assignment axioms + separating conjunction
  - small means just applies to fragment of heap (*footprint*)
  - forward means Floyd-style forward rules that support symbolic execution
  - non-faulting semantics of Hoare triples
  - symbolic execution used by tools like smallfoot
  - separating conjunction solves frame problem - like rule of constancy for heap
- **Need new kinds of assertions to state separation logic axioms**

# Sneak preview of the Frame Rule ✓

**The frame rule**

$$\frac{\vdash\ \{P\}\,C\,\{Q\}}{\vdash\ \{P \star R\}\,C\,\{Q \star R\}}$$

where no variable modified by $C$ occurs free in $R$.

- **Separating conjunction $P \star Q$**
  - **heap can be split into two disjoint components**
  - **$P$ is true of one component and $Q$ of the other**
  - **$\star$ is commutative and associative**

# Local Reasoning and Separation Logic

- **Want to just reason about just those locations being modified**
  - **assume all other locations unchanged**
- **Solution: separation logic**
  - **small and forward assignment axioms + separating conjunction**
  - **small means just applies to fragment of heap (*footprint*)**
  - **forward means Floyd-style forward rules that support symbolic execution**
  - **non-faulting semantics of Hoare triples**
  - **symbolic execution used by tools like smallfoot**
  - **separating conjunction solves frame problem - like rule of constancy for heap**
- **Need new kinds of assertions to state separation logic axioms**

# Separation logic assertions: `emp`

- `emp` is an atomic statement of separation logic
- `emp` is true iff the heap is empty
- The semantics of `emp` is:

  `emp` $(s, h) \;\Leftrightarrow\; h = \{\}$ (where $\{\}$ is the empty heap)

- Abbreviation: $E_1 \doteq E_2 \;\;=_{def}\;\; (E_1 = E_2) \;\wedge\;$ `emp`
- From the semantics: $(E_1 \doteq E_2)\;(s, h) \;\Leftrightarrow\; E_1(s) = E_2(s) \;\wedge\; h = \{\}$
- $E_1 = E_2$ is independent of the heap and only depends on the store
- Semantics of $E_1 = E_2$ is:

  $(E_1 = E_2)(s, h) \Leftrightarrow E_1(s) = E_2(s)$

  no constraint on the heap – any $h$ will do

# Separation logic: small axioms and faulting ✓

- **One might expect a heap assignment axiom to entail:**

  $\vdash \{\texttt{T}\}\,\texttt{[0]:=0}\{0 \mapsto 0\}$

  **i.e. after executing** `[0]:=0` **the contents of location** $0$ **in the heap is** $0$

- **Recall the sneak preview of the frame rule:**

**The frame rule**

$$\frac{\vdash\ \{P\}\,C\,\{Q\}}{\vdash\ \{P \star R\}\,C\,\{Q \star R\}}$$

where no variable modified by $C$ occurs free in $R$.

- **Taking** $R$ **to be the points-to statement** $0 \mapsto 1$ **yields:**

  $\vdash \{\texttt{T}\ \star\ 0 \mapsto 1\}\,\texttt{[0]:=0}\{0 \mapsto 0\ \star\ 0 \mapsto 1\}$

  **something is wrong with the conclusion!**

- **Solution: define Hoare triple so** $\vdash \{\texttt{T}\}\,\texttt{[0]:=0}\{0 \mapsto 0\}$ **is not true**

# Non-faulting interpretation of Hoare triples

- The *non-faulting semantics* of Hoare triples $\{P\}\,C\,\{Q\}$ is:

  if $P$ holds then
  
  (i) executing $C$ doesn't fault and
  
  (ii) if $C$ terminates then $Q$ holds

$$\models \{P\}C\{Q\} \;=\; \forall s\ h.\ P(s,h) \Rightarrow\ \neg(C(s,h)\mathtt{fault})\ \wedge\ \forall s'\ h'.\ C(s,h)(s',h') \Rightarrow Q(s',h')$$

- Now $\vdash \{\mathtt{T}\}\,\mathtt{[0]:=0}\{0 \mapsto 0\}$ is not true as $(\mathtt{[0]:=0})(s,\{\})\mathtt{fault}$
- Recall the sneak preview of the frame rule:

**The frame rule**

$$\frac{\vdash\ \{P\}\,C\,\{Q\}}{\vdash\ \{P \star R\}\,C\,\{Q \star R\}}$$

where no variable modified by $C$ occurs free in $R$.

- So can't use frame rule to get $\vdash \{\mathtt{T}\ \star\ 0 \mapsto 1\}\,\mathtt{[0]:=0}\{0 \mapsto 0\ \star\ 0 \mapsto 1\}$

# Store assignment axiom

**Store assignment axiom**

$$\vdash\ \{V \doteq v\}\, V := E\, \{V \doteq E[v/V]\}$$

where $v$ is an auxiliary variable not occurring in $E$.

- $E_1 \doteq E_2$ **means value of** $E_1$ **and** $E_2$ **equal in the store and heap is empty**
- **In Hoare logic (no heap) this is equivalent to the assignment axiom**

| | |
|---|---|
| $\vdash \{V{=}v\}\, V := E\, \{V{=}E[v/V]\}$ | store assign. ax. |
| $\vdash \{V{=}v \wedge Q[E[v/V]/V]\}\, V := E\, \{V{=}E[v/V] \wedge Q[E[v/V]/V]\}$ | rule of constancy |
| $\vdash \{\exists v.\ V{=}v \wedge Q[E[v/V]/V]\}\, V := E\, \{\exists v.\ V{=}E[v/V] \wedge Q[E[v/V]/V]\}$ | exists introduction |
| $\vdash \{\exists v.\ V{=}v \wedge Q[E[V/V]/V]\}\, V := E\, \{\exists v.\ V{=}E[v/V] \wedge Q[V/V]\}$ | predicate logic |
| $\vdash \{\exists v.\ V{=}v \wedge Q[E/V]\}\, V := E\, \{\exists v.\ V{=}E[v/V] \wedge Q\}$ | $[V/V]$ is identity |
| $\vdash \{(\exists v.\ V{=}v) \wedge Q[E/V]\}\, V := E\, \{(\exists v.\ V{=}E[v/V]) \wedge Q\}$ | predicate logic: $v$ not in $E$ |
| $\vdash \{\mathtt{T} \wedge Q[E/V]\}\, V := E\, \{(\exists v.\ V{=}E[v/V]) \wedge Q\}$ | predicate logic |
| $\vdash \{Q[E/V]\}\, V := E\, \{Q\}$ | rules of consequence |

- **Separation logic: exists introduction valid, rule of constancy invalid**

# Fetch assignment axiom ✓

**Fetch assignment axiom**

$$\vdash \ \{(V = v_1) \wedge E \mapsto v_2\}\, V\texttt{:=[}E\texttt{]}\, \{(V = v_2) \wedge E\texttt{[}v_1\texttt{/}V\texttt{]} \mapsto v_2\}$$

where $v_1$, $v_2$ are auxiliary variables not occurring in $E$.

- **Precondition guarantees the assignment doesn't fault**
- $V$ **is assigned the contents of** $E$ **in the heap**
- **Small axiom: precondition and postcondition specify singleton heap**
- **If neither** $V$ **nor** $v$ **occur in** $E$ **then the following holds:**

  $\vdash \ \{E \mapsto v\}\, V\texttt{:=[}E\texttt{]}\, \{(V = v) \wedge E \mapsto v\}$

  **(proof: instantiate** $v_1$ **to** $V$ **and** $v_2$ **to** $v$ **and then simplify)**

# Heap assignment axiom ✓

**Heap assignment axiom (heap mutation)**

$$\vdash \ \{E \mapsto \_\} \, \texttt{[}E\texttt{]:=}F \, \{E \mapsto F\}$$

- **Precondition guarantees the assignment doesn't fault**
- **Contents of $E$ in heap is updated to be value of $F$**
- **Small axiom: precondition and postcondition specify singleton heap**

# Pointer allocation ✓

**Allocation assignment axiom**

$$\vdash \ \{V \doteq v\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{V \mapsto E_1\texttt{[}v\texttt{/}V\texttt{]},\ldots,E_n\texttt{[}v\texttt{/}V\texttt{]}\}$$

where $v$ is an auxiliary variable not equal to $V$ or occurring in $E_1,\ldots,E_n$

- **Never faults**
- **If $V$ doesn't occur in $E_1,\ldots,E_n$ then:**

$\vdash \ \{V \doteq v\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{V \mapsto E_1\texttt{[}v\texttt{/}V\texttt{]},\ldots,E_n\texttt{[}v\texttt{/}V\texttt{]}\}$ alloc. assign. ax

$\vdash \ \{V \doteq v\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{V \mapsto E_1,\ldots,E_n\}$ V not in $E_i$ assump.

$\vdash \ \{\exists v.\ V \doteq v\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{\exists v.\ V \mapsto E_1,\ldots,E_n\}$ exists intro.

$\vdash \ \{\exists v.\ V{=}v \wedge \texttt{emp}\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{\exists v.\ V \mapsto E_1,\ldots,E_n\}$ definition of $\doteq$

$\vdash \ \{\texttt{emp}\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{V \mapsto E_1,\ldots,E_n\}$ predicate logic

- **Which is a derivation of:**

**Derived allocation assignment axiom**

$$\vdash \ \{\texttt{emp}\}\, V\texttt{:=cons}(E_1,\ldots,E_n)\, \{V \mapsto E_1,\ldots,E_n\}$$

where $V$ doesn't occur in $E_1,\ldots,E_n$.

# Pointer deallocation ✓

**Dispose axiom**

$$\vdash\ \{E \mapsto \_\}\,\texttt{dispose(}E\texttt{)}\,\{\texttt{emp}\}$$

- **Attempting to deallocate a pointer not in the heap faults**
- **Small axiom: singleton precondition heap, empty postcondition heap**
- **Sanity checking example proof:**

| | |
|---|---|
| $\vdash\ \{E_1 \mapsto \_\}\,\texttt{dispose(}E_1\texttt{)}\,\{\texttt{emp}\}$ | dispose axiom |
| $\vdash\ \{\texttt{emp}\}\,V\texttt{:=cons}(E_2)\,\{V \mapsto E_2\}$ | derived allocation assignment axiom |
| $\vdash\ \{E_1 \mapsto \_\}\,\texttt{dispose(}E_1\texttt{);}V\texttt{:=cons}(E_2)\,\{V \mapsto E_2\}$ | sequencing rule |

# Compound command rules

- **Following rules apply to both Hoare logic and separation logic**

**The sequencing rule**

$$\frac{\vdash\ \{P\}\ C_1\ \{Q\}, \qquad \vdash\ \{Q\}\ C_2\ \{R\}}{\vdash\ \{P\}\ C_1 ; C_2\ \{R\}}$$

**The conditional rule**

$$\frac{\vdash\ \{P \wedge S\}\ C_1\ \{Q\}, \qquad \vdash\ \{P \wedge \neg S\}\ C_2\ \{Q\}}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

**The `WHILE`-rule**

$$\frac{\vdash\ \{P \wedge S\}\ C\ \{P\}}{\vdash\ \{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \{P \wedge \neg S\}}$$

- **For separation logic, need to think about faulting**