C04 – Weakest Precondition Calculus · Separation Logic

Program Verification

FMI · Denisa Diaconescu · Spring 2025

Recall Hoare Logic - proof rules

The assignment axiom: $\boxed{\{Q[x/\mathbb{E}]\}\ x := \mathbb{E}\ \{Q\}}$

Strengthening Precond. rule: $$\boxed{\dfrac{P_s \rightarrow P_w \qquad \{P_w\}\ \mathtt{S}\ \{Q\}}{\{P_s\}\ \mathtt{S}\ \{Q\}}}$$

Weakening Postcond. rule: $$\boxed{\dfrac{\{P\}\ \mathtt{S}\ \{Q_s\} \qquad Q_s \rightarrow Q_w}{\{P\}\ \mathtt{S}\ \{Q_w\}}}$$

Sequencing rule: $$\boxed{\dfrac{\{P\}\mathtt{S}_1\{Q\} \qquad \{Q\}\mathtt{S}_2\{R\}}{\{P\}\mathtt{S}_1;\mathtt{S}_2\{R\}}}$$

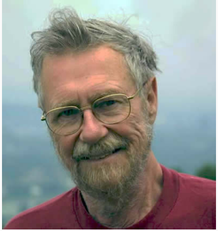
Conditional rule: $$\boxed{\dfrac{\{P \land b\}\ \mathtt{S}_1\ \{Q\} \qquad \{P \land \neg b\}\ \mathtt{S}_2\ \{Q\}}{\{P\}\ \mathtt{if\ b\ then\ S}_1\ \mathtt{else\ S}_2\ \{Q\}}}$$

While rule: $$\boxed{\dfrac{\{P \land b\}\ \mathtt{S}\ \{P\}}{\{P\}\ \mathtt{while\ b\ do\ S}\ \{P \land \neg b\}}}$$

Weakest Precondition Calculus

Weakest precondition calculus

- Edsger W. Dijkstra: introduced another technique for proving properties of imperative programs.
- Weakest Precondition calculus (WP)



Hoare logic presents logic problems:

- Given a precondition P, some code $\mathbb{C}$, and postcondition Q, is the Hoare triple $\{P\}$ $\mathbb{C}$ $\{Q\}$ true?

WP is about evaluating a function:

- Given some code $\mathbb{C}$ and postcondition Q, find the unique P which is the weakest precondition such that Q holds after $\mathbb{C}$.

Weakest precondition calculus

If $\mathbb{C}$ is a code fragment and Q is an assertion about states, then the weakest precondition for $\mathbb{C}$ with respect to Q is an assertion that is true for precisely those initial states from which:

- $\mathbb{C}$ must terminate, and
- executing $\mathbb{C}$ must produce a state satisfying Q.

The weakest precondition P is a function of $\mathbb{C}$ and Q:

$$P = wp(\mathbb{C}, Q)$$

- The function wp is sometimes called predicate transformer.
- The calculus WP is sometimes called Predicate Transformer Semantics.

Relationship with Hoare Logic

Hoare Logic is relational:

- For each Q, there are many P such that $\{P\}\ \mathbb{C}\ \{Q\}$.
- For each P, there are many Q such that $\{P\}\ \mathbb{C}\ \{Q\}$.

WP is functional:

- For each Q, there is exactly one assertion $wp(\mathbb{C}, Q)$.

WP respects Hoare logic: $\{wp(\mathbb{C}, Q)\}\ \mathbb{C}\ \{Q\}$ is true.

Hoare logic is about partial correctness (we don't care about termination).

WP is about total correctness (we do care about termination).

Total correctness = Termination + Partial correctness

Intuition

Example

Consider the code `x := x+1` and postcondition $(x > 0)$.

- One valid precondition is $(x > 0)$, so in Hoare logic the following is true

$$\{x > 0\}\ \texttt{x := x+1}\ \{x > 0\}$$

- Another valid precondition is $(x > -1)$, so

$$\{x > -1\}\ \texttt{x := x+1}\ \{x > 0\}$$

- $(x > -1)$ is weaker than $(x > 0)$ (since $(x > 0) \to (x > -1)$)
- In fact $(x > -1)$ is the weakest precondition

$$wp(\texttt{x := x+1}, x > 0) \ \equiv\ (x > -1)$$

Weakest precondition for Assignment (Rule 1/4)

The Assignment axiom of Hoare Logic is designed to give the "best" (i.e., the weakest) precondition:

$$\{Q[x/\mathbb{E}]\}\ \texttt{x := }\mathbb{E}\ \{Q\}$$

Therefore the rule for Assignment in the weakest precondition calculus corresponds closely:

$$\boxed{wp(\texttt{x := }\mathbb{E}, Q) \equiv Q[x/\mathbb{E}]}$$

(Q is an assertion involving a variable x and $Q[x/\mathbb{E}]$ indicates the same assertion with all occurrences of x replaced by the expression $\mathbb{E}$)

Weakest precondition for Assignment

The rule for Assignment: $\boxed{wp(\texttt{x := } \mathbb{E}, Q) \equiv Q[x/\mathbb{E}]}$

Example

$$\begin{aligned} wp(\texttt{x := y+3}, x > 3) &\equiv y + 3 > 3 && \text{(substitute } y + 3 \text{ for } x\text{)} \\ &\equiv y > 0 && \text{(simplify)} \end{aligned}$$

$$\begin{aligned} wp(\texttt{n := n+1}, n > 5) &\equiv n + 1 > 5 && \text{(substitute } n + 1 \text{ for } n\text{)} \\ &\equiv n > 4 && \text{(simplify)} \end{aligned}$$

Weakest precondition for Sequences (Rule 2/4)

The rule for sequencing compose the effect of the consecutive statements:

$$\boxed{\ wp(\mathbb{C}_1;\ \mathbb{C}_2, Q) \ \equiv\ wp(\mathbb{C}_1, wp(\mathbb{C}_2, Q))}$$

Example

$wp(\texttt{x := x+2; y := y-2}, x + y = 0)$

Weakest precondition for Sequences (Rule 2/4)

The rule for sequencing compose the effect of the consecutive statements:

$$\boxed{\;wp(\mathbb{C}_1\,;\;\;\mathbb{C}_2, Q) \;\equiv\; wp(\mathbb{C}_1, wp(\mathbb{C}_2, Q))}$$

Example

$$\begin{aligned}
& wp(\texttt{x := x+2; y := y-2}, x + y = 0) \\
\equiv\quad & wp(\texttt{x := x+2}, wp(\texttt{y := y-2}, x + y = 0)) \\
\equiv\quad & wp(\texttt{x := x+2}, x + (y - 2) = 0) \\
\equiv\quad & (x + 2) + (y - 2) = 0 \\
\equiv\quad & x + y = 0
\end{aligned}$$

Weakest precondition for Conditionals (Rule 3a/4)

$$\boxed{wp(\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}_1 \texttt{ else } \mathbb{C}_2, Q) \;\equiv\; (\mathbb{B} \rightarrow wp(\mathbb{C}_1, Q)) \wedge (\neg\mathbb{B} \rightarrow wp(\mathbb{C}_2, Q))}$$

Example

$$wp(\texttt{if x > 2 then y := 1 else y := -1}, y > 0)$$

Weakest precondition for Conditionals (Rule 3a/4)

$$\boxed{wp(\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}_1 \texttt{ else } \mathbb{C}_2, Q) \equiv (\mathbb{B} \rightarrow wp(\mathbb{C}_1, Q)) \wedge (\neg\mathbb{B} \rightarrow wp(\mathbb{C}_2, Q))}$$

Example

$$\begin{aligned}
& wp(\texttt{if x > 2 then y := 1 else y := -1}, y > 0) \\
\equiv \quad & ((x > 2) \rightarrow wp(\texttt{y := 1}, y > 0)) \wedge (\neg(x > 2) \rightarrow wp(\texttt{y := -1}, y > 0)) \\
\equiv \quad & ((x > 2) \rightarrow (1 > 0)) \wedge (\neg(x > 2) \rightarrow (-1 > 0)) \\
\equiv \quad & ((x > 2) \rightarrow \top) \wedge (\neg(x > 2) \rightarrow \bot) \\
\equiv \quad & x > 2
\end{aligned}$$

Alternative rule for Conditionals (Rule 3b/4)

It is often easier to deal with disjunctions and conjunctions than implications, so the following equivalent rule for conditionals is usually more convenient.

$$\boxed{wp(\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}_1 \texttt{ else } \mathbb{C}_2, Q) \equiv (\mathbb{B} \wedge wp(\mathbb{C}_1, Q)) \vee (\neg\mathbb{B} \wedge wp(\mathbb{C}_2, Q))}$$

Example

$$\begin{array}{ll}
 & wp(\texttt{if x > 2 then y := 1 else y := -1}, y > 0) \\
\equiv & ((x > 2) \wedge wp(\texttt{y := 1}, y > 0)) \vee (\neg(x > 2) \wedge wp(\texttt{y := -1}, y > 0)) \\
\equiv & ((x > 2) \wedge (1 > 0)) \vee (\neg(x > 2) \wedge (-1 > 0)) \\
\equiv & ((x > 2) \wedge \top) \vee (\neg(x > 2) \wedge \bot) \\
\equiv & (x > 2) \vee \bot \\
\equiv & (x > 2)
\end{array}$$

Proof rule for Conditionals

Exercise:

How would you derive a rule for a conditional statement without else?

$$\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}$$

Loops

Suppose we have a while loop and some postcondition Q.

The precondition P that we seek is the weakest that:

- establishes Q
- guarantees termination

We can take hints for the corresponding rule for Hoare Logic. That is, think in terms of loop invariants.

But termination is a bigger problem!

An undecidable problem

Determining if a program terminates or not on a given input is an **undecidable problem**!

So there's no algorithm to compute $wp(\texttt{while}\ \mathbb{B}\ \texttt{do}\ \mathbb{C}, Q)$ in all cases.

But that doesn't mean there are no techniques to tackle this problem that at least work some of the time!

Guaranteeing termination

The precondition P we seek is the weakest that establishes Q and guarantees termination.

How a loop can terminate?

- If the loop is never entered, then the postcondition Q must already be true and the boolean control expression $\mathbb{B}$ false.
    - We will call this precondition P_0.
    - $P_0 \equiv \neg\mathbb{B} \wedge Q$ i.e, $\{\neg\mathbb{B} \wedge Q\}$ do nothing $\{Q\}$
- Suppose the loop executes exactly once. In this case:
    - $\mathbb{B}$ must be true initially
    - after the first time through the loop, P_0 must become true (so that the loop terminates next time through).
    - $P_1 \equiv \mathbb{B} \wedge wp(\mathbb{C}, P_0)$ i.e., $\{\mathbb{B} \wedge wp(\mathbb{C}, P_0)\}$ $\mathbb{C}$ $\{P_0\}$

Guaranteeing termination

$P_0 \equiv \neg\mathbb{B} \wedge Q$ i.e, $\{\neg\mathbb{B} \wedge Q\}$ do nothing $\{Q\}$

$P_1 \equiv \mathbb{B} \wedge wp(\mathbb{C}, P_0)$ i.e., $\{\mathbb{B} \wedge wp(\mathbb{C}, P_0)\}\ \mathbb{C}\ \{P_0\}$

$P_2 \equiv \mathbb{B} \wedge wp(\mathbb{C}, P_1)$ i.e., $\{\mathbb{B} \wedge wp(\mathbb{C}, P_1)\}\ \mathbb{C}\ \{P_1\}$

$P_3 \equiv \mathbb{B} \wedge wp(\mathbb{C}, P_2)$ i.e., $\{\mathbb{B} \wedge wp(\mathbb{C}, P_2)\}\ \mathbb{C}\ \{P_2\}$

$\ldots$

P_k – the weakest precondition under which the loop terminates with postcondition Q after exactly k iterations.

We can capture the definition of P_k with an inductive definition.

An inductive definition

$$
\begin{array}{rcl}
P_0 & \equiv & \neg\mathbb{B} \wedge Q \\
P_{k+1} & \equiv & \mathbb{B} \wedge wp(\mathbb{C}, P_k)
\end{array}
$$

If any of the P_k is true in the initial state, then we are guaranteed that the loop will terminate and establish the postcondition Q,

i.e. $\{P_0 \vee P_1 \vee \ldots\}$ `while` $\mathbb{B}$ `do` $\mathbb{C}$ $\{Q\}$ is true.

The weakest precondition for while loops (rule 4/4)

$$\boxed{wp(\texttt{while } \mathbb{B} \texttt{ do } \mathbb{C}, Q) \equiv \exists k\ (k \geq 0 \wedge P_k)}$$

where P_k is defined inductively:

$$\begin{aligned} P_0 &\equiv \neg\mathbb{B} \wedge Q \\ P_{k+1} &\equiv \mathbb{B} \wedge wp(\mathbb{C}, P_k) \end{aligned}$$

Interpretation:

- P_k is the weakest precondition that ensures that the body $\mathbb{C}$ executes exactly k times and terminates in a state in which postcondition Q holds.

- If our loop is to terminate with postcondition Q, some P_k must hold before we enter the loop
  i.e. $\{P_0 \vee P_1 \vee \ldots\}$ `while` $\mathbb{B}$ `do` $\mathbb{C}$ $\{Q\}$ is true.

The weakest precondition for while loops

Applying the *wp* function to a while loop and postcondition will produce an assertion of the form

$$\exists k \ (k \geq 0 \land P_k)$$

P_k may be different for each k, so *wp* may produce an infinitely long assertion! Such an assertion is unsuitable for further manipulations.

We can simplify matters by expressing P_k as a single, finite formula that is parameterised by k.

Example
If $P_0 \equiv (n = 0)$, $P_1 \equiv (n = 1)$, $P_2 \equiv (n = 2)$, $\ldots$, then $P_k \equiv (n = k)$.

We must prove correctness of P_k by induction!

The weakest precondition for while loops

$$\boxed{wp(\texttt{while } \mathbb{B} \texttt{ do } \mathbb{C}, Q) \equiv \exists k\ (k \geq 0 \wedge P_k)}$$

$$\begin{aligned} P_0 &\equiv \neg\mathbb{B} \wedge Q \\ P_{k+1} &\equiv \mathbb{B} \wedge wp(\mathbb{C}, P_k) \end{aligned}$$

Example

Suppose we want to find:

$$wp(\texttt{while n > 0 do n := n-1}, n = 0)$$

We start by generating some of the P_k sequence:

- $P_0 \equiv \neg(n > 0) \wedge (n = 0) \equiv (n = 0)$ i.e., $\neg\mathbb{B} \wedge Q$
- $P_1 \equiv (n > 0) \wedge wp(n := n - 1, n = 0) \equiv (n = 1)$ i.e., $\mathbb{B} \wedge wp(\mathbb{C}, P_0)$
- $P_2 \equiv (n > 0) \wedge wp(n := n - 1, n = 1) \equiv (n = 2)$ i.e., $\mathbb{B} \wedge wp(\mathbb{C}, P_1)$
- $\ldots$

so it looks pretty likely that $P_k \equiv (n = k)$

The weakest precondition for while loops

Example

Suppose we want to find:

$$wp(\texttt{while n > 0 do n := n-1}, n = 0)$$

We prove by induction that $P_k \equiv (n = k)$:

- We already checked the base case:

  $P_0 \quad \equiv \quad \neg(n > 0) \wedge (n = 0) \equiv (n = 0)$

- Now for our induction step:

  We assume $P_i \equiv (n = i)$ for some $i \geq 0$.

  Recall that $P_{i+1} \equiv \mathbb{B} \wedge wp(\mathbb{C}, P_i)$.

$$\begin{aligned} P_{i+1} \quad &\equiv \quad (n > 0) \wedge wp(n := n - 1, n = i) \\ &\equiv \quad (n > 0) \wedge (n - 1 = i) \\ &\equiv \quad (n > 0) \wedge (n = i + 1) \\ &\equiv \quad (n = i + 1) \end{aligned}$$

The weakest precondition for while loops

Example

Therefore we have

$$wp(\texttt{while n > 0 do n := n-1}, n = 0) \equiv \exists k\ (k \geq 0 \wedge n = k)$$

We can still simplify it further!

Useful trick: $\exists k\ ((k \geq 0) \wedge P_k) \equiv P_0 \vee P_1 \vee P_2 \vee \dots$

In this example we have $(n = 0) \vee (n = 1) \vee (n = 2) \vee \dots$

We can compress this infinite disjunction into a finite final result:

$$wp(\texttt{while n > 0 do n := n-1}, n = 0) \equiv (n \geq 0)$$

Total correctness

Example
We want to find

$$wp(\texttt{while n} \neq \texttt{0 do n := n-1}, n = 0)$$

Step 1 - finding the P_k:

- $P_0 \equiv \neg(n \neq 0) \land (n = 0) \equiv (n = 0)$ i.e., $\neg\mathbb{B} \land Q$
- $P_1 \equiv (n \neq 0) \land wp(n := n - 1, n = 0) \equiv (n = 1)$ i.e., $\mathbb{B} \land wp(\mathbb{C}, P_0)$
- ...
- $P_k \equiv (n = k)$ (induction omitted)

Total correctness

Example
Step 2 - finding the weakest precondition:

$$\begin{aligned} \exists k \ ((k \geq 0) \wedge P_k) &\equiv \exists k \ (k \geq 0 \wedge n = k) \\ &\equiv (n \geq 0) \end{aligned}$$

Thus,

$$wp(\texttt{while n} \neq \texttt{0 do n := n-1}, n = 0) \ \equiv \ (n \geq 0)$$

This is not really any different than the previous example.

But what is the trap in this while-loop?

We have automatically found that the while-loop will not terminate for initial values of n less than 0.

WP rules

- Rule for Assignment: $wp(\texttt{x := } \mathbb{E}, Q(x)) \equiv Q(\mathbb{E})$
- Rule for Sequencing: $wp(\mathbb{C}_1 \texttt{;}\ \mathbb{C}_2, Q) \equiv wp(\mathbb{C}_1, wp(\mathbb{C}_2, Q))$
- Rule for Conditionals:
  $wp(\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}_1 \texttt{ else } \mathbb{C}_2, Q) \equiv (\mathbb{B} \rightarrow wp(\mathbb{C}_1, Q)) \wedge (\neg\mathbb{B} \rightarrow wp(\mathbb{C}_2, Q))$
- There is no algorithm to compute $wp(\texttt{while } \mathbb{B} \texttt{ do } \mathbb{C}, Q)$ in all cases!
  - But that doesn't mean there are no techniques to tackle this problem that at least work some of the time!
  - Inductive definition.

Quiz time!

https://tinyurl.com/FMI-PV2023-Quiz4

Separation Logic

Adding the heap

We extend our toy programming language with:

- Heap reads: `x := [`$\mathbb{E}$`]` *(dereferencing)*
- Heap writes: `[`$\mathbb{E}_1$`] :=` $\mathbb{E}_2$ *(update heap)*
- Heap allocation: `x := cons(`$\mathbb{E}_1$`,...`$\mathbb{E}_n$`)`
- Heap deallocation: `dispose` $\mathbb{E}$

The state is now represented by a pair of type $Store \times Heap$, denoted (σ, h), where

$$\sigma \in Store, \text{ where } Store \triangleq Var \rightarrow Val$$

$$h \in Heap, \text{ where } Heap \triangleq Loc \rightarrow Val$$

where $Loc \subseteq Val$.

Note that we consider $dom(h)$ to always be finite. By this, we ensure that `cons` commands will never fail.

Adding the heap

Heap reads: `x := [`$\mathbb{E}$`]`

- evaluate expression $\mathbb{E}$ to get location l
- fault if location l is not in the current heap
- otherwise variable `x` is assigned the content of location l

Example (`x := [y+1]`)

σ

y	0xAB

h

0xAB	1
0xAC	2

`x := [y+1]`

σ

y	0xAB
x	2

h

0xAB	1
0xAC	2

Adding the heap

Heap writes: `[`$\mathbb{E}_1$`] :=` $\mathbb{E}_2$

- evaluate expression $\mathbb{E}_1$ to get location *l*
- fault if location *l* is not in the current heap
- otherwise make the content of location *l* the value of expression $\mathbb{E}_2$

Example (`[y+1] := 5`)

σ

y	0xAB

h

0xAB	1
0xAC	2

`[y+1] := 5`

σ

y	0xAB

h

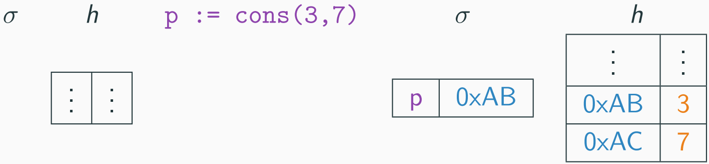
0xAB	1
0xAC	5

Adding the heap

Heap allocation: `x := cons(`$\mathbb{E}_1$`,...`$\mathbb{E}_n$`)`

- extend the heap with n consecutive new locations $l, l+1, \ldots, l+n-1$
- put values of $\mathbb{E}_1$,...,$\mathbb{E}_n$ into locations $l, l+1, \ldots, l+n-1$ respectively
- extend the stack by assigning `x` the value l
- never fault

Example (`p := cons(3,7)`)



σ h `p := cons(3,7)` σ h

⋮	⋮

p	0xAB

⋮	⋮
0xAB	3
0xAC	7

Adding the heap

Heap deallocation: `dispose` $\mathbb{E}$

- evaluate expression $\mathbb{E}$ to get location l
- fault if location l is not in the current heap
- otherwise remove location l from the heap

Example (`dispose p+1`)

σ

p	0xAB

h

0xAB	5
0xAC	6

`dispose p+1`

σ

p	0xAB

h

0xAB	5

Example

```
x := cons(3,3)
```

σ

x	0xAB

h

0xAB	3
0xAC	3

Example

```
x := cons(3,3); y := cons(4,4);
```

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	3
0xDD	4
0xDE	4

Example

```
x := cons(3,3) ; y := cons(4,4); [x+1] := y;
```

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	4

Example

```
x := cons(3,3) ; y := cons(4,4); [x+1] := y; [y+1] := x;
```

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

Example

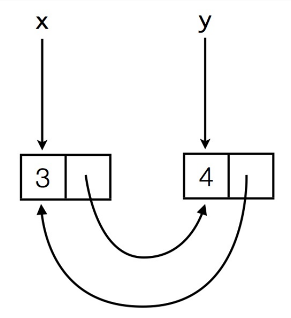
```
x := cons(3,3) ; y := cons(4,4); [x+1] := y; [y+1] := x;
```

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Why separation logic?

Can you suggest a precondition such that this triple holds?

$\{???\}$

```
    [y] := 4; [z] := 5;
```

$\{(\exists y, z)(\mathtt{y} \mapsto y \wedge \mathtt{z} \mapsto z \wedge y \neq z)\}$

Note that, for example, $\mathtt{y}$ is used to denote program variables, while y is used to denote logical variables.

Why separation logic?

Can you suggest a precondition such that this triple holds?

$$\{\mathtt{y} \neq \mathtt{z} \wedge \mathtt{y} \mapsto \_ \wedge \mathtt{z} \mapsto \_\}$$

```
      [y] := 4; [z] := 5;
```

$$\{(\exists y, z)(\mathtt{y} \mapsto y \wedge \mathtt{z} \mapsto z \wedge y \neq z)\}$$

Note that, for example, $\mathtt{y}$ is used to denote program variables, while y is used to denote logical variables.

We need to assume that the locations pointed by $\mathtt{y}$ and $\mathtt{z}$ are different (aliasing).

Why separation logic?

And now?

```
      [y] := 4; [z] := 5;
```

$$\{(\exists y, z)(\mathtt{y} \mapsto y \wedge \mathtt{z} \mapsto z \wedge y \neq z \wedge \mathtt{x} \mapsto 3)\}$$

We need to assume that the locations pointed by `y` and `z` are different (aliasing).

We also need to know when things stay the same.

Why separation logic?

And now?

$$\{\mathtt{y} \neq \mathtt{z} \wedge \mathtt{x} \neq \mathtt{y} \wedge \mathtt{x} \neq \mathtt{z} \wedge \mathtt{y} \mapsto \_ \wedge \mathtt{z} \mapsto \_ \wedge \mathtt{x} \mapsto 3\}$$

```
      [y] := 4; [z] := 5;
```

$$\{(\exists y, z)(\mathtt{y} \mapsto y \wedge \mathtt{z} \mapsto z \wedge y \neq z \wedge \mathtt{x} \mapsto 3)\}$$

We need to assume that the locations pointed by y and z are different (aliasing).

We also need to know when things stay the same.

Framing

We want a general concept of things not being affected.

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{\mathtt{x} \mapsto 3 \land P\}\ \mathbb{C}\ \{Q \land \mathtt{x} \mapsto 3\}}$$

What are the conditions on $\mathbb{C}$ and $\mathtt{x} \mapsto 3$?

These are very hard to define if reasoning about a heap and aliasing.

Framing

We want a general concept of things not being affected.

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{\mathtt{x} \mapsto 3 \land P\}\ \mathbb{C}\ \{Q \land \mathtt{x} \mapsto 3\}}$$

What are the conditions on $\mathbb{C}$ and $\mathtt{x} \mapsto 3$?

These are very hard to define if reasoning about a heap and aliasing.

This is where separation logic comes in:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{R * P\}\ \mathbb{C}\ \{Q * R\}}$$

The new connective $*$ ("sep" operator) is used to separate the heap.

From Hoare logic to separation logic

- Robert W. Floyd 1967: gave some rules to reason about programs.
- Sometimes, our Hoare Logic is called Floyd-Hoare Logic in recognition.
- Many attempts made to extend Floyd-Hoare Logic to handle pointers.

- Only really solved around 2000 by Reynolds, O'Hearn and Yang using a connective $*$ called separating conjunction.

Extra connectives in separation logic

$\mathtt{emp}$	empty heap
$\mathbb{E}_1 \mapsto \mathbb{E}_2$	points to
$P * Q$	separating conjunction

Evaluating expressions in the store of a state

Strictly speaking, the store gives values to variables only.

But we need a way to say ”value of an expression in a store” so we will abuse notation and use $\sigma(\mathbb{E})$ for this as below:

- $\sigma(n) = n$ where n is a number is just its usual value
- $\sigma(x + n) = \sigma(x) + \sigma(n)$ where n is a number and x is a variable

Semantics of separation logic

$\sigma \triangleq \mathit{Var} \rightarrow \mathit{Val}$

$h \triangleq \mathit{Loc} \rightarrow \mathit{Val}$

$(\sigma, h) \models \texttt{emp}$ if $\mathit{dom}(h) = \emptyset$

- `emp` is an atomic formula for checking if the heap is empty
- a state (σ, h) makes the formula `emp` true if the heap is empty

Semantics of separation logic

$\sigma \triangleq \mathit{Var} \to \mathit{Val}$

$h \triangleq \mathit{Loc} \to \mathit{Val}$

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $\mathit{dom}(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

- a state (σ, h) makes the formula $\mathbb{E}_1 \mapsto \mathbb{E}_2$ true if the heap is a singleton and maps the location $\sigma(\mathbb{E}_1)$ to the value $\sigma(\mathbb{E}_2)$
- $\sigma(\mathbb{E})$ is the value of an expression in a store as explained before

Semantics of separation logic

$\sigma \triangleq Var \to Val$

$h \triangleq Loc \to Val$

$(\sigma, h) \models P * Q$ if h can be partitioned into two disjoint heaps h_1 and h_2,

and $(\sigma, h_1) \models P$ and $(\sigma, h_2) \models Q$

Note that two heaps are disjoint if the intersection of their domains is empty.

Semantics of separation logic

$\sigma \triangleq Var \rightarrow Val$

$h \triangleq Loc \rightarrow Val$

$(\sigma, h) \models P * Q$ if h can be partitioned into two disjoint heaps h_1 and h_2,

and $(\sigma, h_1) \models P$ and $(\sigma, h_2) \models Q$

Note that two heaps are disjoint if the intersection of their domains is empty.

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n disjoint heaps

$h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Semantics of separation logic

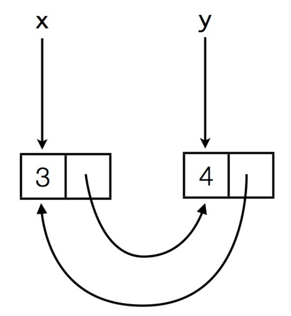
Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Semantics of separation logic

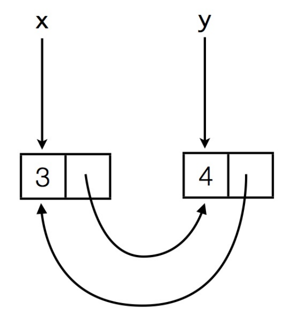
Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Satisfies the statement:

$$(\mathrm{x} \mapsto 3) * (\mathrm{x} + 1 \mapsto \mathrm{y}) * (\mathrm{y} \mapsto 4) * (\mathrm{y} + 1 \mapsto \mathrm{x})$$

Semantics of separation logic

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n distinct heaps $h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

We want to show that

$$(\sigma, h) \models (\mathtt{x} \mapsto 3) * (\mathtt{x} + 1 \mapsto \mathtt{y}) * (\mathtt{y} \mapsto 4) * (\mathtt{y} + 1 \mapsto \mathtt{x})$$

Semantics of separation logic

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n distinct heaps $h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

We want to show that

$$(\sigma, h) \models (\mathtt{x} \mapsto 3) * (\mathtt{x} + 1 \mapsto \mathtt{y}) * (\mathtt{y} \mapsto 4) * (\mathtt{y} + 1 \mapsto \mathtt{x})$$

We can partition h into 4 distinct heaps:

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

Semantics of separation logic

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n distinct heaps $h_1, h_2, \ldots, h_n$

and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

We want to show that

$$(\sigma, h) \models (\mathtt{x} \mapsto 3) * (\mathtt{x} + 1 \mapsto \mathtt{y}) * (\mathtt{y} \mapsto 4) * (\mathtt{y} + 1 \mapsto \mathtt{x})$$

We can partition h into 4 distinct heaps:

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

We must show that

$(\sigma, h_1) \models \mathtt{x} \mapsto 3$

$(\sigma, h_2) \models \mathtt{x} + 1 \mapsto \mathtt{y}$

$(\sigma, h_3) \models \mathtt{y} \mapsto 4$

$(\sigma, h_4) \models \mathtt{y} + 1 \mapsto \mathtt{x}$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = 0\text{xAB} = \sigma(\mathtt{x})$
- $h_1(0\text{xAB}) = 3$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = 0\text{xAB} = \sigma(\mathtt{x})$
- $h_1(0\text{xAB}) = 3$

$(\sigma, h_2) \models x + 1 \mapsto y$

- $dom(h_2) = 0\text{xAC} = \sigma(\mathtt{x} + 1)$
- $h_2(0\text{xAC}) = 0\text{xDD} = \sigma(\mathtt{y})$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = \text{0xAB} = \sigma(\mathtt{x})$
- $h_1(\text{0xAB}) = 3$

$(\sigma, h_2) \models x + 1 \mapsto y$

- $dom(h_2) = \text{0xAC} = \sigma(\mathtt{x} + 1)$
- $h_2(\text{0xAC}) = \text{0xDD} = \sigma(\mathtt{y})$

$(\sigma, h_3) \models \mathtt{y} \mapsto 4$

- $dom(h_3) = \text{0xDD} = \sigma(\mathtt{y})$
- $h_3(\text{0xDD}) = 4$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $\mathit{dom}(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $\mathit{dom}(h_1) = \text{0xAB} = \sigma(\mathtt{x})$
- $h_1(\text{0xAB}) = 3$

$(\sigma, h_2) \models x + 1 \mapsto y$

- $\mathit{dom}(h_2) = \text{0xAC} = \sigma(\mathtt{x} + 1)$
- $h_2(\text{0xAC}) = \text{0xDD} = \sigma(\mathtt{y})$

$(\sigma, h_3) \models \mathtt{y} \mapsto 4$

- $\mathit{dom}(h_3) = \text{0xDD} = \sigma(\mathtt{y})$
- $h_3(\text{0xDD}) = 4$

$(\sigma, h_4) \models \mathtt{y} + 1 \mapsto \mathtt{x}$

- $\mathit{dom}(h_4) = \text{0xDE} = \sigma(\mathtt{y} + 1)$
- $h_4(\text{0xDE}) = \text{0xAB} = \sigma(\mathtt{x})$

Semantics of separation logic

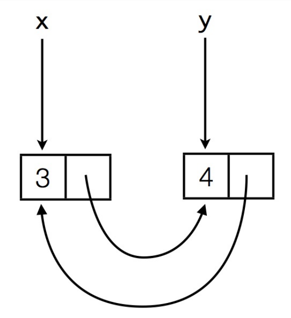
Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Does not satisfy the statement $\mathtt{x} \mapsto 3$

Semantics of separation logic

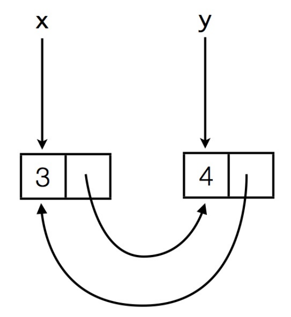
Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Does not satisfy the statement $\mathtt{x} \mapsto 3$

- $(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $\mathit{dom}(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$
- $\mathit{dom}(h) = \{0\text{xAB}, 0\text{xAC}, 0\text{xDD}, 0\text{xDE}\}$
- $\sigma(\mathtt{x}) = 0\text{xAB}$
- $h(\sigma(\mathtt{x})) = 3$

Store assignment axiom of Floyd

Hoare axiom: $\{Q[x/\mathbb{E}]\}$ `x := ` $\mathbb{E}$ $\{Q\}$ (backward driven)

Floyd axiom: $\{x = v\}$ `x := ` $\mathbb{E}$ $\{x = \mathbb{E}[x/v]\}$ (forward driven)

- equivalent to Hoare axiom
- v is an auxiliary variable which does not occur in $\mathbb{E}$
- $\mathbb{E}[x/v]$ means replace all occurrences of x in $\mathbb{E}$ by v

Example

Hoare instance: $\{x + 1 = 5\}$ `x := x+1` $\{x = 5\}$

Floyd instance: $\{x = v\}$ `x := x+1` $\{x = v + 1\}$

- If we want the postcondition $x = 5$ then instantiate v to be 4
  $\{x = 4\}$ `x := x+1` $\{x = 5\}$

Note: does not solve the problem with pointers!

Store assignment axiom for separation logic

Hoare axiom: $\{Q[\mathrm{x}/\mathbb{E}]\}\ \texttt{x := }\mathbb{E}\ \{Q\}$

Floyd axiom: $\{\mathrm{x} = v\}\ \texttt{x := }\mathbb{E}\ \{\mathrm{x} = \mathbb{E}[\mathrm{x}/v])\}$

where v is an auxiliary variable which does not occur in $\mathbb{E}$.

Store assignment axiom for Separation logic:

$$\{\mathrm{x} = v \wedge \texttt{emp}\}\ \texttt{x := }\mathbb{E}\ \{\mathrm{x} = \mathbb{E}[\mathrm{x}/v] \wedge \texttt{emp}\}$$

where v is an auxiliary variable which does not occur in $\mathbb{E}$

New:

- atomic formula `emp` to say that the "heap is empty"
- we want to track the smallest amount of heap information

Store assignment axiom for separation logic

Store assignment axiom for Separation logic:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ }\mathbb{E}\ \{\mathtt{x} = \mathbb{E}[\mathtt{x}/v] \wedge \mathtt{emp}\}$$

where v is an auxiliary variable which does not occur in $\mathbb{E}$

Example

$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ 1}\ \{\mathtt{x} = 1 \wedge \mathtt{emp}\}$

If we want the precondition $1 = 1$ (i.e. $\top$) then instantiate v to $\mathtt{x}$
$\{\mathtt{x} = \mathtt{x} \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ 1}\ \{\mathtt{x} = 1 \wedge \mathtt{emp}\}$

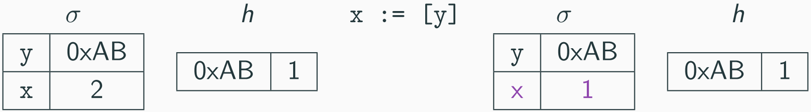
Heap reads axiom

Heap reads axiom:

$$\{\mathtt{x} = v_1 \land \mathbb{E} \mapsto v_2\}\ \mathtt{x\ :=\ [}\mathbb{E}\mathtt{]}\ \{\mathtt{x} = v_2 \land \mathbb{E}[\mathtt{x}/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Example (`x := [y]`)



σ

y	0xAB
x	2

h

0xAB	1

`x := [y]`

σ

y	0xAB
x	1

h

0xAB	1

Heap read axiom instance:

$$\{\mathtt{x} = 2 \land \mathtt{y} \mapsto 1\}\ \mathtt{x\ :=\ [y]}\ \{\mathtt{x} = 1 \land \mathtt{y} \mapsto 1\}$$

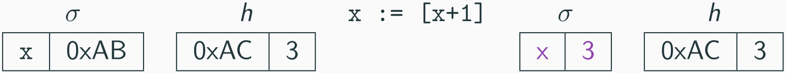
Heap reads axiom

Heap reads axiom:

$$\{\mathtt{x} = v_1 \land \mathbb{E} \mapsto v_2\}\ \mathtt{x\ \ :=\ \ [}\mathbb{E}\mathtt{]}\ \{\mathtt{x} = v_2 \land \mathbb{E}[\mathtt{x}/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Example (`x := [x+1]`)



σ	
x	0xAB

h	
0xAC	3

`x := [x+1]`

σ	
x	3

h	
0xAC	3

Heap read axiom instance:

$$\{\mathtt{x} = \mathsf{0xAB} \land \mathtt{x+1} \mapsto 3\}\ \mathtt{x\ \ :=\ \ [x+1]}\ \{\mathtt{x} = 3 \land \mathsf{0xAC} \mapsto 3\}$$

Heap writes axiom

Heap writes axiom:

$$\{\mathbb{E}_1 \mapsto -\}\ [\mathbb{E}_1] := \mathbb{E}_2\ \{\mathbb{E}_1 \mapsto \mathbb{E}_2\}$$

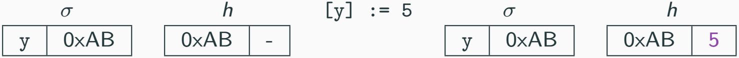
where $(\mathbb{E}_1 \mapsto -)$ abbreviates $(\exists z. \mathbb{E}_1 \mapsto z)$ and z does no occur in $\mathbb{E}_1$

Heap assignment semantics:

- evaluate expression $\mathbb{E}_1$ to get location l
- fault if location l is not in the current heap
- otherwise make the contents of location l the value of expression $\mathbb{E}_2$

Example

$\{y \mapsto -\}$ `[y] := 5` $\{y \mapsto 5\}$



σ

y	0xAB

h

0xAB	-

`[y] := 5`

σ

y	0xAB

h

0xAB	5

Heap allocation axiom

Heap allocation axiom:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)\ \{\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]\}$$

where v is a variable diff. from $\mathtt{x}$ and not appearing in $\mathbb{E}_1, \ldots, \mathbb{E}_n$

Heap allocation assignment axiom means: if $\sigma(\mathtt{x}) = v$ and the heap is empty then executing $\mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)$ gives a heap consisting of n new consecutive locations, where location $\sigma(\mathtt{x}) + i$ contains $\sigma(\mathbb{E}_{i+1}[\mathtt{x}/v])$

σ

$\mathtt{x}$	–
$\mathtt{y}$	7

h

$\mathtt{x} := \mathtt{cons}(5, \mathtt{y} + 1)$

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	7

h

0xAB	5
0xAC	8

$\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]$ abbreviates

$\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v]\ *\ (\mathtt{x} + 1) \mapsto \mathbb{E}_2[\mathtt{x}/v]\ * \ldots *\ (\mathtt{x} + n - 1) \mapsto \mathbb{E}_n[\mathtt{x}/v]$

Heap deallocation axiom

Heap deallocation axiom: $\{\mathbb{E} \mapsto -\}$ `dispose` $\mathbb{E}$ $\{\texttt{emp}\}$

where $(\mathbb{E} \mapsto -)$ abbreviates $(\exists z.\mathbb{E} \mapsto z)$ and z does no occur in $\mathbb{E}$

Heap deallocation: `dispose` $\mathbb{E}$

- evaluate $\mathbb{E}$ to get location l
- fault if location l is not in the current heap
- otherwise remove location l from the heap

Heap deallocation axiom means: if the heap is a singleton with domain $\sigma(\mathbb{E})$ then executing `dispose` $\mathbb{E}$ results in the empty heap.

Separation logic axioms - recap

Store assignment axiom:

$$\{\mathtt{x} = v \land \mathtt{emp}\}\ \mathtt{x\ :=}\ \mathbb{E}\ \{\mathtt{x} = \mathbb{E}[\mathtt{x}/v] \land \mathtt{emp}\}$$

where v is an auxiliary variable which does not occur in $\mathbb{E}$

Heap reads axiom:

$$\{\mathtt{x} = v_1 \land \mathbb{E} \mapsto v_2\}\ \mathtt{x\ :=\ [}\mathbb{E}\mathtt{]}\ \{\mathtt{x} = v_2 \land \mathbb{E}[\mathtt{x}/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Heap writes axiom: $\{\mathbb{E}_1 \mapsto -\}[\mathbb{E}_1] := \mathbb{E}_2\{\mathbb{E}_1 \mapsto \mathbb{E}_2\}$

where $(\mathbb{E}_1 \mapsto -)$ abbreviates $(\exists z.\mathbb{E}_1 \mapsto z)$ and z does no occur in $\mathbb{E}_1$

Heap allocation axiom:

$$\{\mathtt{x} = v \land \mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)\ \{\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]\}$$

where v is a variable diff. from $\mathtt{x}$ and not appearing in $\mathbb{E}_1, \ldots, \mathbb{E}_n$

Heap deallocation axiom: $\{\mathbb{E} \mapsto -\}\ \mathtt{dispose}\ \ \mathbb{E}\ \{\mathtt{emp}\}$

where $(\mathbb{E} \mapsto -)$ abbreviates $(\exists z.\mathbb{E} \mapsto z)$ and z does no occur in $\mathbb{E}$

The frame rule

Frame rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{P * R\}\ \mathbb{C}\ \{Q * R\}}$$

where no variables modified by $\mathbb{C}$ appears free in R.

The Frame rule means that $\{P\}\ \mathbb{C}\ \{Q\}$ is restricted to the variables and parts of the heap that are actually used by $\mathbb{C}$.

The frame rule

Frame rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{P * R\}\ \mathbb{C}\ \{Q * R\}}$$

where no variables modified by $\mathbb{C}$ appears in R.

Example

Is the following instance a legal instance of the Frame rule?
If so, why and if not, why not?

$$\frac{\{\mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(1)\ \{\mathtt{x} \mapsto 1\}}{\{\mathtt{emp}\ *\ \mathtt{x} \mapsto 1\}\ \mathtt{x} := \mathtt{cons}(1)\ \{\mathtt{x} \mapsto 1\ *\ \mathtt{x} \mapsto 1\}}$$

The frame rule

Frame rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{P * R\}\ \mathbb{C}\ \{Q * R\}}$$

where no variables modified by $\mathbb{C}$ appears in R.

Example

Is the following instance a legal instance of the Frame rule?
If so, why and if not, why not?

$$\frac{\{\texttt{emp}\}\ \texttt{x} := \texttt{cons}(1)\ \{\texttt{x} \mapsto 1\}}{\{\texttt{emp}\ *\ \texttt{x} \mapsto 1\}\ \texttt{x} := \texttt{cons}(1)\ \{\texttt{x} \mapsto 1\ *\ \texttt{x} \mapsto 1\}}$$

No, the command modifies `x` and R contains an occurrence of `x`.

Quiz time!

`https://tinyurl.com/FMI-PV2023-Quiz5`

References

- Lecture Notes on ”Formal Methods for Software Engineering”, Australian National University, Rajeev Goré.
- Mike Gordon, ”Specification and Verification I”, chapters 1 and 2.
- Michael Huth, Mark Ryan, ”Logic in Computer Science: Modeling and Reasoning about Systems”, 2nd edition, Cambridge University Press, 2004.
- Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog, ”Verification of Sequential and Concurrent Programs”, 3rd edition, Springer.