

Formal Systems II: Theory

Separation Logic

SS 2018

Mattias Ulbrich
Institute of Theoretical Informatics

Motivation

Given: a program with a contract:

- 1 precondition, FOL formula pre
- 2 postcondition, FOL formula $post$
- 3 code, while program π

In program verification, one formally proves that

$$\mathbb{N} \models pre \rightarrow [\pi]post$$

If pre holds before execution of π then $post$ holds after termination.

Reminder: weakest precondition calculus for DL.

Formal Software Verification

- Prove what effects a program has.

Formal Software Verification

- Prove what effects a program has.
- Prove what effects a program does *not* have.

Formal Software Verification

- Prove what effects a program has.
- Prove what effects a program does *not* have.

You should not have to specify the latter explicitly.

Formal Software Verification

- Prove what effects a program has.
- Prove what effects a program does *not* have.

You should not have to specify the latter explicitly.

Example (after McCarthy and Hayes, 1969)

P calls operator to ask for Q 's number.



Formal Software Verification

- Prove what effects a program has.
- Prove what effects a program does *not* have.

You should not have to specify the latter explicitly.

Example (after McCarthy and Hayes, 1969)

P calls operator to ask for Q 's number.

- **Precondition:** P has a telephone.



Formal Software Verification

- Prove what effects a program has.
- Prove what effects a program does *not* have.

You should not have to specify the latter explicitly.

Example (after McCarthy and Hayes, 1969)

P calls operator to ask for Q 's number.

- **Precondition:** P has a telephone.
- **Postcondition:** P knows the number of Q



Formal Software Verification

- Prove what effects a program has.
- Prove what effects a program does *not* have.

You should not have to specify the latter explicitly.

Example (after McCarthy and Hayes, 1969)

P calls operator to ask for Q 's number.

- **Precondition:** P has a telephone.
- **Postcondition:** P knows the number of Q
- **missing postcondition?**
Postcondition: P still has a telephone.



Example in Java

```
interface Account {  
    void setBalance(int);  
    int getBalance();  
}
```

Example in Java

```
interface Account {
    void setBalance(int);
    int getBalance();
}

/*@ ensures \result == 100;
int f(Account account1, Account account2) {
    account1.setBalance(100);
    account2.setBalance(200);
    return account1.getBalance();
}
```

Example in Java

```
interface Account {  
    void setBalance(int);  
    int getBalance();  
}  
  
/*@ requires account1 != account2;  
/*@ ensures \result == 100;  
int f(Account account1, Account account2) {  
    account1.setBalance(100);  
    account2.setBalance(200);  
    return account1.getBalance();  
}
```

Specify what does *not* change

Example in Java

```
interface Account {
    void setBalance(int);
    int getBalance();
}

/*@ requires account1 !=
  *@ ensures \result == 100;
int f(Account account1, Account account2) {
    account1.setBalance(100);
    account2.setBalance(200);
    return account1.getBalance();
}
```

Specify what does *not* change

Example in Java

```
interface Account {  
    void setBalance(int);  
    int getBalance();  
}  
  
/*@ requires account1 !=  
    *@ ensures \result == 100;  
int f(Account account1, Account account2) {  
    account1.setBalance(100);  
    account2.setBalance(200);  
    return account1.getBalance();  
}
```

- setBalance does not effect other accounts

Example in Java

```
interface Account {  
    void setBalance(int);  
    int getBalance();  
}
```

```
//@ requires account1 !=  
//@ ensures \result == 100;  
int f(Account account1, Account account2) {  
    account1.setBalance(100);  
    account2.setBalance(200);  
    return account1.getBalance();  
}
```

Specify what does *not* change

- setBalance does not effect other accounts
- setBalance does not effect other customer objects

Example in Java

```
interface Account {
    void setBalance(int);
    int getBalance();
}

/*@ requires account1 !=
  *@ ensures \result == 100;
int f(Account account1, Account account2) {
    account1.setBalance(100);
    account2.setBalance(200);
    return account1.getBalance();
}
```

Specify what does *not* change

- setBalance does not effect other accounts
- setBalance does not effect other customer objects
- setBalance does not effect any object of any classes which may be added later.

Problem statement

In program verification, the framing problem is the problem to specify and verify that the effects of a program are limited to the data structure that is being operated on.

It is a challenge for specifying user (needs to think about not-effects) and for reasoning engines (increased complexity).

Problem statement

In program verification, the framing problem is the problem to specify and verify that the effects of a program are limited to the data structure that is being operated on.

It is a challenge for specifying user (needs to think about not-effects) and for reasoning engines (increased complexity).

Suggested solutions:

- Ownership (Types) (Noble, Vitek and Potter 1998)
- Separation Logic (Reynolds, 1999)
- Dynamic Frames (Kassios 2006)
- ...

Problem statement

In program verification, the framing problem is the problem to specify and verify that the effects of a program are limited to the data structure that is being operated on.

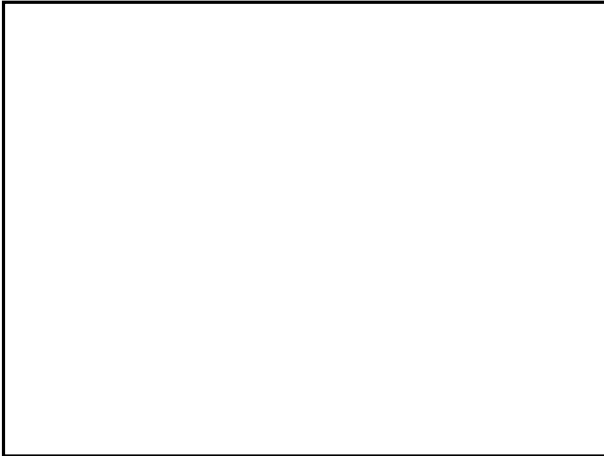
It is a challenge for specifying user (needs to think about not-effects) and for reasoning engines (increased complexity).

Suggested solutions:

- Ownership (Types) (Noble, Vitek and Potter 1998)
- Separation Logic (Reynolds, 1999)
- Dynamic Frames (Kassios 2006)
- ...

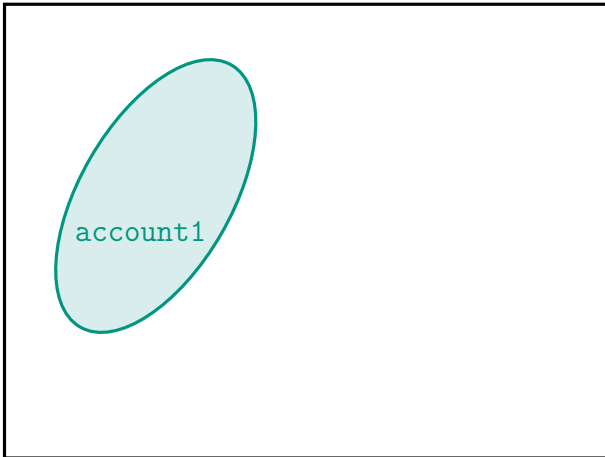
Heaps and “Footprints”

Heap



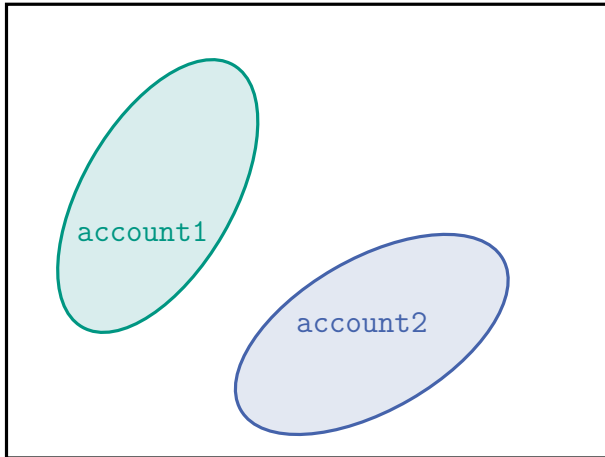
Heaps and “Footprints”

Heap



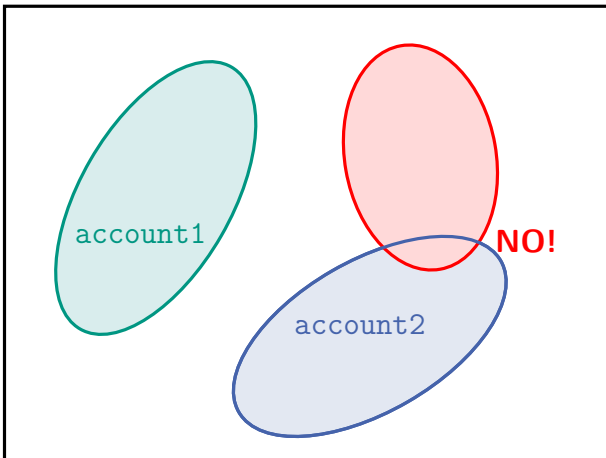
Heaps and “Footprints”

Heap



Heaps and “Footprints”

Heap



Modelling assumptions

- Every memory location holds a value in \mathbb{N} .
- There infinitely many memory locations.

Heap and Heaplet

A **heap** is a total function modelling memory:

$$\text{heap} : \mathbb{N} \rightarrow \mathbb{N}$$

A **heaplet** is a finite partial function modelling footprints:

$$\text{heaplet} : \mathbb{N} \dashrightarrow \mathbb{N}$$

Partial function:

Partial function $f : A \dashrightarrow B$ is a function $f : D \rightarrow B$ for $D \subseteq A$.

The finite set $D = \text{dom } f$ is called the domain of f .

Disjoint union of heaplets:

$$h = h_1 \uplus h_2 \quad \text{iff} \quad \text{dom } h_1 \cap \text{dom } h_2 = \emptyset \text{ and } h = h_1 \cup h_2.$$

$h_1 \uplus h_2$ is always a heaplet.

(Union \cup of heaplets does not always result in heaplets.)

Membership

For $(x, y) \in h$ write $h(x) = y$.

It means: Memory location x holds value y .

Empty Heap

The empty heaplet \emptyset is without allocated locations.

Singletons

Heaplet with exactly one allocated location x which holds value y :

write $h = \{(x, y)\}$

Separation Logic

Terms t :

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, \dots

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x.\varphi$, $\exists x.\varphi$

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x.\varphi$, $\exists x.\varphi$
- $\varphi_1 * \varphi_2$

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x.\varphi$, $\exists x.\varphi$
- $\varphi_1 * \varphi_2$
- **emp**

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x.\varphi$, $\exists x.\varphi$
- $\varphi_1 * \varphi_2$
- **emp**
- $t_1 \mapsto t_2$

new in Separation Logic

Terms t :

- FOL terms over \mathbb{N} with $+$, $-$, \cdot , 0 , 1

Formulae φ :

- $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x.\varphi$, $\exists x.\varphi$
- $\varphi_1 * \varphi_2$
- **emp**
- $t_1 \mapsto t_2$
- $\varphi_1 \multimap \varphi_2$ (later)

new in Separation Logic

Operator Precedence

How are the implicit parentheses in

$$B \rightarrow C \wedge D \vee A * x \mapsto y ?$$

How are the implicit parentheses in

$$B \multimap C \wedge D \vee A * x \mapsto y ?$$

Binding force:

- * binds like \wedge
- \multimap binds like \rightarrow, \vee
- \mapsto binds like $=$

How are the implicit parentheses in

$$B \rightarrow C \wedge D \vee A * x \mapsto y ?$$

Binding force:

- * binds like \wedge
- \rightarrow binds like \rightarrow, \vee
- \mapsto binds like $=$

Answer:

$$(B \rightarrow (C \wedge D)) \vee (A * (x \mapsto y))$$

or

$$B \rightarrow ((C \wedge D) \vee (A * (x \mapsto y)))$$

How are the implicit parentheses in

$$B \ast C \wedge D \vee A \ast x \mapsto y ?$$

Binding force:

- \ast binds like \wedge
- $\rightarrow \ast$ binds like \rightarrow, \vee
- \mapsto binds like $=$

Answer:

$$(B \ast (C \wedge D)) \vee (A \ast (x \mapsto y))$$

or $B \ast ((C \wedge D) \vee (A \ast (x \mapsto y)))$

Add explicit parentheses when combining $\vee / \rightarrow / \rightarrow \ast$ or \wedge / \ast

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
- 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
- 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$

Terms:

- $val_{\beta}(t_1 + t_2) = val_{\beta}(t_1) +_{\mathbb{N}} val_{\beta}(t_2)$, same for “.”
- $val_{\beta}(x) = \beta(x)$ for variable x

Formulas in FOL:

- Operator $\beta, h \models$ is homomorphic for $\wedge, \vee, \rightarrow, \forall, \exists, <, =$.
- Example: $\beta, h \models \varphi_1 \wedge \varphi_2$ iff $\beta, h \models \varphi_1$ and $\beta, h \models \varphi_2$

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
 - 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$
- $\beta, h \models \mathbf{emp}$ iff $\text{dom } h = \emptyset$

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
 - 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$
- $\beta, h \models \mathbf{emp}$ iff $\text{dom } h = \emptyset$
 - $\beta, h \models t_1 \mapsto t_2$ iff $h = \{(val_{\beta}(t_1), val_{\beta}(t_2))\}$

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
 - 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$
- $\beta, h \models \mathbf{emp}$ iff $\text{dom } h = \emptyset$
 - $\beta, h \models t_1 \mapsto t_2$ iff $h = \{(val_\beta(t_1), val_\beta(t_2))\}$
 - $\beta, h \models \varphi_1 * \varphi_2$ iff there exist heaplets $h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N}$ with

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
- 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$

- $\beta, h \models \mathbf{emp}$ iff $\text{dom } h = \emptyset$
- $\beta, h \models t_1 \mapsto t_2$ iff $h = \{(val_\beta(t_1), val_\beta(t_2))\}$
- $\beta, h \models \varphi_1 * \varphi_2$ iff there exist heaplets $h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N}$ with
 - 1 $h = h_1 \uplus h_2$ and

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
- 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$

- $\beta, h \models \mathbf{emp}$ iff $\text{dom } h = \emptyset$
- $\beta, h \models t_1 \mapsto t_2$ iff $h = \{(val_\beta(t_1), val_\beta(t_2))\}$
- $\beta, h \models \varphi_1 * \varphi_2$ iff there exist heaplets $h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N}$ with
 - 1 $h = h_1 \uplus h_2$ and
 - 2 $\beta, h_1 \models \varphi_1$ and

Structure

Fixed first order domain: \mathbb{N} .

Terms and formulas are evaluated over:

- 1 Variable assignment $\beta : Var \rightarrow \mathbb{N}$
- 2 Heaplet $h : \mathbb{N} \rightarrow \mathbb{N}$

- $\beta, h \models \mathbf{emp}$ iff $\text{dom } h = \emptyset$
- $\beta, h \models t_1 \mapsto t_2$ iff $h = \{(val_\beta(t_1), val_\beta(t_2))\}$
- $\beta, h \models \varphi_1 * \varphi_2$ iff there exist heaplets $h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N}$ with
 - 1 $h = h_1 \uplus h_2$ and
 - 2 $\beta, h_1 \models \varphi_1$ and
 - 3 $\beta, h_2 \models \varphi_2$

Connector $*$ is called **Separating Conjunction**

$A * B$ has the following intuitive semantics:

$A * B$ is true
 \iff
 A is true
and B is true
and A and B refer to
disjoint sets of memory locations.

Idempotence

- $\models A \leftrightarrow A \wedge A$

(idempotence for \wedge)

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\stackrel{?}{\models} A \leftrightarrow A * A$ (idempotence also for $*$?)

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\stackrel{?}{\models} A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\models^? A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\models^? A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\models^? A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \wedge B \rightarrow A$ (Weakening of conjunction)

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\models^? A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \wedge B \rightarrow A$ (Weakening of conjunction)
- $\models^? A * B \rightarrow A$ (Weakening of separating conjunction?)

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\models^? A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \wedge B \rightarrow A$ (Weakening of conjunction)
- $\models^? A * B \rightarrow A$ (Weakening of separating conjunction?)
- **NO!** Counterexample:

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for \wedge)
- $\models^? A \leftrightarrow A * A$ (idempotence also for $*$?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \wedge B \rightarrow A$ (Weakening of conjunction)
- $\models^? A * B \rightarrow A$ (Weakening of separating conjunction?)
- **NO!** Counterexample:
 $\models \neg(7 \mapsto 3 * 6 \mapsto 4 \rightarrow 7 \mapsto 3)$

Caution

$\beta, h \models A \mapsto B$ means that:

- $\{(val(A), val(B))\} = h,$
- not only $(val(A), val(B)) \in h$

$\beta, h \models A \mapsto B$ means that:

- $\{(val(A), val(B))\} = h$,
- not only $(val(A), val(B)) \in h$

On the other hand:

$$\beta, h \models ? \iff (val(A), val(B)) \in h$$

$\beta, h \models A \mapsto B$ means that:

- $\{(val(A), val(B))\} = h$,
- not only $(val(A), val(B)) \in h$

On the other hand:

$$\beta, h \models A \mapsto B * true \iff (val(A), val(B)) \in h$$

$\beta, h \models A \mapsto B$ means that:

- $\{(val(A), val(B))\} = h,$
- not only $(val(A), val(B)) \in h$

On the other hand:

$$\beta, h \models A \mapsto B * true \iff (val(A), val(B)) \in h$$

Notation sometimes: $A \hookrightarrow B \text{ :}\leftrightarrow A \mapsto B * true$

Some Valid Formulas

- $\mathbf{emp} \leftrightarrow \neg(\exists x, y. x \mapsto y * \mathit{true})$

Some Valid Formulas

- $\mathbf{emp} \leftrightarrow \neg(\exists x, y. x \mapsto y * \mathit{true})$
- $\varphi * \psi \leftrightarrow \varphi \wedge \psi$
if neither **emp** nor \mapsto occur.

Some Valid Formulas

- $\mathbf{emp} \leftrightarrow \neg(\exists x, y. x \mapsto y * \mathit{true})$

- $\varphi * \psi \leftrightarrow \varphi \wedge \psi$
if neither **emp** nor \mapsto occur.

- $x \mapsto y \wedge x \mapsto z \rightarrow y = z$

Some Valid Formulas

- $\mathbf{emp} \leftrightarrow \neg(\exists x, y. x \mapsto y * \mathbf{true})$
- $\varphi * \psi \leftrightarrow \varphi \wedge \psi$
if neither **emp** nor \mapsto occur.
- $x \mapsto y \wedge x \mapsto z \rightarrow y = z$
- $P * (Q \vee R) \leftrightarrow (P * Q) \vee (P * R)$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1 $x \mapsto y * x \mapsto z$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1 $x \mapsto y * x \mapsto z$

UNSAT

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

① $x \mapsto y * x \mapsto z$

② $x \mapsto y \wedge x \mapsto z$

UNSAT

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

① $x \mapsto y * x \mapsto z$

UNSAT

② $x \mapsto y \wedge x \mapsto z$

SAT ; true if $y = z$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

① $x \mapsto y * x \mapsto z$

UNSAT

② $x \mapsto y \wedge x \mapsto z$

SAT ; true if $y = z$

③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

① $x \mapsto y * x \mapsto z$

UNSAT

② $x \mapsto y \wedge x \mapsto z$

SAT ; true if $y = z$

③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$

VALID

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

① $x \mapsto y * x \mapsto z$

UNSAT

② $x \mapsto y \wedge x \mapsto z$

SAT ; true if $y = z$

③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$

VALID

④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|--|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

① $x \mapsto y * x \mapsto z$

UNSAT

② $x \mapsto y \wedge x \mapsto z$

SAT ; true if $y = z$

③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$

VALID

④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$

SAT

⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | |

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ $\varphi * \mathbf{emp} \rightarrow \varphi$ | |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ $\varphi * \mathbf{emp} \rightarrow \varphi$ | VALID |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|------------------------------|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ $\varphi * \mathbf{emp} \rightarrow \varphi$ | VALID |
| ⑧ $\varphi * \neg\varphi$ | |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|--|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ $\varphi * \mathbf{emp} \rightarrow \varphi$ | VALID |
| ⑧ $\varphi * \neg\varphi$ | |
| a. $\psi * \neg\psi$ | for ψ without \mapsto, \mathbf{emp} |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|---|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ $\varphi * \mathbf{emp} \rightarrow \varphi$ | VALID |
| ⑧ $\varphi * \neg\varphi$ | |
| a. $\psi * \neg\psi$ | UNSAT for ψ without \mapsto, \mathbf{emp} |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | |
|---|---|
| ① $x \mapsto y * x \mapsto z$ | UNSAT |
| ② $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ $\varphi * \mathbf{emp} \rightarrow \varphi$ | VALID |
| ⑧ $\varphi * \neg\varphi$ | |
| a. $\psi * \neg\psi$ | UNSAT for ψ without \mapsto, \mathbf{emp} |
| b. $x \mapsto y * \neg(x \mapsto y)$ | |

Are the following formulas valid/satisfiable/unsatisfiable?

- | | | |
|----|---|--|
| ① | $x \mapsto y * x \mapsto z$ | UNSAT |
| ② | $x \mapsto y \wedge x \mapsto z$ | SAT ; true if $y = z$ |
| ③ | $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$ | VALID |
| ④ | $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$ | SAT |
| ⑤ | $(x \mapsto 0 * y \mapsto 0) \rightarrow \neg(x = y)$ | VALID |
| ⑥ | $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$ | VALID |
| ⑦ | $\varphi * \mathbf{emp} \rightarrow \varphi$ | VALID |
| ⑧ | $\varphi * \neg\varphi$ | |
| a. | $\psi * \neg\psi$ | UNSAT for ψ without \mapsto , emp |
| b. | $x \mapsto y * \neg(x \mapsto y)$ | SAT , equivalent to $x \mapsto y * \mathit{true}$ |

Modus Ponens for classical logic

$$\frac{A \wedge (A \rightarrow B)}{B}$$

Modus Ponens for classical logic

$$\frac{A \wedge (A \rightarrow B)}{B}$$

Corresponding rule for separating conjunction *?

Modus Ponens for classical logic

$$\frac{A \wedge (A \rightarrow B)}{B}$$

Corresponding rule for separating conjunction $*$?

Modus Ponens for separation logic

$$\frac{A * (A \multimap B)}{B}$$

The **magic wand operator** $A \multimap B$, aka **separating implication**:

$$\beta, h \models A \multimap B$$



for all $h', h^+ : \mathbb{N} \mapsto \mathbb{N}$: If $h^+ = h \uplus h'$ and $h' \models A$, then $h^+ \models B$

Separating Operators

- $\models_{\text{SL}} f * g$ when there are ◐ and ◑ such that ● = ◐ ◑, as well as ◐ $\models_{\text{SL}} f$ and ◑ $\models g$.
- ▷ $\models_{\text{SL}} f \multimap g$ when any ◐ such that ◐ $\models_{\text{SL}} f$ is also such that ◐ $\models g$.

Figure 1.5: Visual representation of the semantics of separation operators

Taken from:

Separation Logic: Expressiveness, Complexity, Temporal Extension

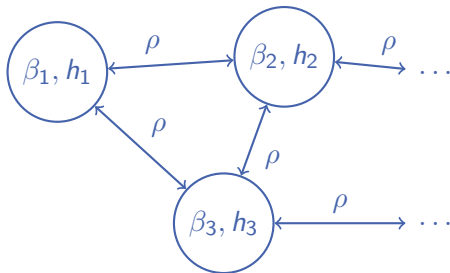
Rémi Brochenin, PhD Thesis. 2013

Programs and Separation Logic

```
statement ::= while formula do statement
           |   if formula then statement else statement
           |   statement ; statement
           |   var := term
           |   [term] := term
           |   var := [term]
(later)   |   var := cons(term, ..., term)
(later)   |   dispose(var)
```

Kripke Frames with Heaps

- Every state is a pair (β, h) with $\beta : \text{Var} \rightarrow \mathbb{N}$ and $h : \mathbb{N} \rightarrow \mathbb{N}$
- Kripke state transition the program semantics $\rho(st) \in S \times S$ for any statement st .



Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\rho(\pi_1 ; \pi_2) = \rho(\pi_1) ; \rho(\pi_2) \quad ; \text{ is forward composition}$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\begin{aligned} \rho(\pi_1 ; \pi_2) &= \rho(\pi_1) ; \rho(\pi_2) \quad ; \text{ is forward composition} \\ &= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \end{aligned}$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\begin{aligned} \rho(\pi_1 ; \pi_2) &= \rho(\pi_1) ; \rho(\pi_2) && \text{; is forward composition} \\ &= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \end{aligned}$$

$$\rho(\pi^*) = \rho(\pi)^* \quad * \text{ is refl. transitive closure}$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\begin{aligned} \rho(\pi_1 ; \pi_2) &= \rho(\pi_1) ; \rho(\pi_2) \quad ; \text{ is forward composition} \\ &= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \end{aligned}$$

$$\begin{aligned} \rho(\pi^*) &= \rho(\pi)^* \quad * \text{ is refl. transitive closure} \\ &= \{(s_0, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\} \end{aligned}$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\begin{aligned} \rho(\pi_1 ; \pi_2) &= \rho(\pi_1) ; \rho(\pi_2) && \text{; is forward composition} \\ &= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \end{aligned}$$

$$\begin{aligned} \rho(\pi^*) &= \rho(\pi)^* && \text{* is refl. transitive closure} \\ &= \{(s_0, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\} \end{aligned}$$

$$\rho(? \varphi) = \{(s, s) \mid s \models \varphi\}$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\begin{aligned} \rho(\pi_1 ; \pi_2) &= \rho(\pi_1) ; \rho(\pi_2) \quad ; \text{ is forward composition} \\ &= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \end{aligned}$$

$$\begin{aligned} \rho(\pi^*) &= \rho(\pi)^* \quad * \text{ is refl. transitive closure} \\ &= \{(s_0, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\} \end{aligned}$$

$$\rho(? \varphi) = \{(s, s) \mid s \models \varphi\}$$

Reminder: IF and WHILE

$$\begin{aligned} \text{if } \varphi \text{ then } \alpha \text{ else } \beta &= (? \varphi ; \alpha) \cup (? \neg \varphi ; \beta) \\ \text{while } \varphi \text{ do } \alpha &= (? \varphi ; \alpha)^* ; ? \neg \varphi \end{aligned}$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

A state $s \in S$ is a pair (β, h) with $\beta : \text{Var} \rightarrow \mathbb{N}$ and $h : \mathbb{N} \rightarrow \mathbb{N}$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

A state $s \in S$ is a pair (β, h) with $\beta : \text{Var} \rightarrow \mathbb{N}$ and $h : \mathbb{N} \rightarrow \mathbb{N}$

$$((\beta, h), (\beta', h')) \in \rho(v := t) \iff \beta' = \beta[v/\text{val}_\beta(t)] \text{ and } h' = h$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

A state $s \in S$ is a pair (β, h) with $\beta : \text{Var} \rightarrow \mathbb{N}$ and $h : \mathbb{N} \rightarrow \mathbb{N}$

$$((\beta, h), (\beta', h')) \in \rho(v := t) \iff \beta' = \beta[v/\text{val}_\beta(t)] \text{ and } h' = h$$

$$((\beta, h), (\beta', h')) \in \rho(v := [t]) \iff \text{val}_\beta(t) \in \text{dom } h \text{ and } h' = h \text{ and } \beta' = \beta[v/h[\text{val}_\beta(t)]]$$

Accessibility Relation for Programs

$\rho : \text{statement} \rightarrow S \times S$

A state $s \in S$ is a pair (β, h) with $\beta : \text{Var} \rightarrow \mathbb{N}$ and $h : \mathbb{N} \rightarrow \mathbb{N}$

$$((\beta, h), (\beta', h')) \in \rho(v := t) \iff \beta' = \beta[v/\text{val}_\beta(t)] \text{ and } h' = h$$

$$((\beta, h), (\beta', h')) \in \rho(v := [t]) \iff \text{val}_\beta(t) \in \text{dom } h \text{ and } h' = h \text{ and } \beta' = \beta[v/h[\text{val}_\beta(t)]]$$

$$((\beta, h), (\beta', h')) \in \rho([t] := u) \iff \text{val}_\beta(t) \in \text{dom } h \text{ and } \beta' = \beta \text{ and } h' = h[\text{val}_\beta(t)/\text{val}_\beta(u)]$$

(Remember: $f[a/b](a) = b$ and $f[a/b](x) = f(x)$ for $x \neq a$)

Statement $x := [10]$ must not be executed if $10 \notin \text{dom } h$.

State (β, \emptyset) has no successor state in $\rho(x := [10])$.

How to distinguish between failed test $?\psi$ and memory violation?

Statement $x := [10]$ must not be executed if $10 \notin \text{dom } h$.

State (β, \emptyset) has no successor state in $\rho(x := [10])$.

How to distinguish between failed test $?\psi$ and memory violation?

Model unallowed heap access:

$fail : \text{statement} \rightarrow S$

$s \in fail(\pi)$ means: π started in s may cause memory violation.

Model unallowed heap access:

$fail : \text{statement} \rightarrow \mathcal{S}$

$s \in fail(\pi)$ means: π started in s may cause memory violation

$$fail(x := t) =$$

$$fail(? \psi) = \emptyset$$

Model unallowed heap access:

$fail : \text{statement} \rightarrow \mathcal{S}$

$s \in fail(\pi)$ means: π started in s may cause memory violation

$$fail(x := t) =$$

$$fail(?\psi) = \emptyset$$

$$fail(x := [t]) =$$

$$fail([t] := u) = \{(\beta, h) \mid val_{\beta}(t) \notin \text{dom } h\}$$

Model unallowed heap access:

$fail : \text{statement} \rightarrow \mathcal{S}$

$s \in fail(\pi)$ means: π started in s may cause memory violation

$$fail(x := t) =$$

$$fail(? \psi) = \emptyset$$

$$fail(x := [t]) =$$

$$fail([t] := u) = \{(\beta, h) \mid val_{\beta}(t) \notin \text{dom } h\}$$

$$fail(\pi_1 ; \pi_2) = fail(\pi_1) \cup (\rho(\pi_1) ; fail(\pi_2))$$

Model unallowed heap access:

$fail : \text{statement} \rightarrow \mathcal{S}$

$s \in fail(\pi)$ means: π started in s may cause memory violation

$$fail(x := t) =$$

$$fail(?\psi) = \emptyset$$

$$fail(x := [t]) =$$

$$fail([t] := u) = \{(\beta, h) \mid val_{\beta}(t) \notin \text{dom } h\}$$

$$fail(\pi_1 ; \pi_2) = fail(\pi_1) \cup (\rho(\pi_1) ; fail(\pi_2))$$

$$fail(\pi^*) = \rho(\pi^*) ; fail(\pi)$$

Model unallowed heap access:

$fail : \text{statement} \rightarrow \mathcal{S}$

$s \in fail(\pi)$ means: π started in s may cause memory violation

$$fail(x := t) =$$

$$fail(? \psi) = \emptyset$$

$$fail(x := [t]) =$$

$$fail([t] := u) = \{(\beta, h) \mid val_{\beta}(t) \notin \text{dom } h\}$$

$$fail(\pi_1 ; \pi_2) = fail(\pi_1) \cup (\rho(\pi_1) ; fail(\pi_2))$$

$$fail(\pi^*) = \rho(\pi^*) ; fail(\pi)$$

with $A ; B = \{x \mid \text{ex } y \text{ with } (x, y) \in A \text{ and } y \in B\}$

Remember:

$s \models [\pi]\varphi$ iff $s' \models \varphi$ for all $(s, s') \in \rho(\pi)$.

Problem:

$\text{emp} \rightarrow [[5] := 42] \text{false}$ is a valid formula.

New modality $[[\cdot]]$

$s \models [[\pi]]\varphi$ iff $s' \models \varphi$ for all $(s, s') \in \rho(\pi)$ and $s \notin \text{fail}(\pi)$

Now:

$\text{emp} \rightarrow [[5] := 42]]\psi$ is not valid for any ψ

Valid formulas:

- $x \mapsto 5 \rightarrow \llbracket v := [x] ; [x] := v + 1 \rrbracket x \mapsto 6$
- $(\exists y. x \mapsto y) \rightarrow \llbracket [x] := 7 \rrbracket x \mapsto 7$
- $x \mapsto 5 * y \mapsto 6 \rightarrow \llbracket [x] := 7 \rrbracket (x \mapsto 7 * y \mapsto 6)$

Hoare Calculus

Separation Logic originally formulated as rules for a *Hoare* calculus.

Hoare Calculus

Separation Logic originally formulated as rules for a *Hoare* calculus.

Hoare Calculus (1969, Hoare and Floyd)

Operates on **Hoare Triples**: $\{P\} \pi \{Q\}$

A Hoare triple is valid if program π started in a state that satisfies precondition P terminates in a state which satisfies postcondition Q (if it terminates).

Hoare Calculus

Separation Logic originally formulated as rules for a *Hoare* calculus.

Hoare Calculus (1969, Hoare and Floyd)

Operates on **Hoare Triples**: $\{P\} \pi \{Q\}$

A Hoare triple is valid if program π started in a state that satisfies precondition P terminates in a state which satisfies postcondition Q (if it terminates).

Semantically the same as $P \rightarrow \llbracket \pi \rrbracket Q$.

Hoare Calculus

Separation Logic originally formulated as rules for a *Hoare* calculus.

Hoare Calculus (1969, Hoare and Floyd)

Operates on **Hoare Triples**: $\{P\} \pi \{Q\}$

A Hoare triple is valid if program π started in a state that satisfies precondition P terminates in a state which satisfies postcondition Q (if it terminates).

Semantically the same as $P \rightarrow \llbracket \pi \rrbracket Q$.

We present the calculus using dynamic logic notation.

Reminder: Hoare Calculus (in DL notation)

$$\frac{}{P[x \leftarrow E] \rightarrow \llbracket x := E \rrbracket P} \quad [x \leftarrow E] \text{ is substitution}$$

$$\frac{P \rightarrow \llbracket \pi_1 \rrbracket Q \quad Q \rightarrow \llbracket \pi_2 \rrbracket R}{P \rightarrow \llbracket \pi_1 ; \pi_2 \rrbracket R} \quad \frac{P' \rightarrow P \quad P \rightarrow \llbracket \pi \rrbracket Q \quad Q \rightarrow Q'}{P' \rightarrow \llbracket \pi \rrbracket Q'}$$

$$\frac{P \wedge C \rightarrow \llbracket \pi_1 \rrbracket Q \quad P \wedge \neg C \rightarrow \llbracket \pi_2 \rrbracket Q}{P \rightarrow \llbracket \text{if } C \text{ then } \pi_1 \text{ else } \pi_2 \rrbracket Q}$$

$$\frac{P \wedge C \rightarrow \llbracket \pi \rrbracket P}{P \rightarrow \llbracket \text{while } C \text{ do } \pi \rrbracket (P \wedge \neg C)}$$

$$\frac{P \rightarrow \llbracket \pi \rrbracket Q}{(\exists x. P) \rightarrow \llbracket \pi \rrbracket (\exists x. Q)} \quad \text{if } x \notin \text{Free}(\pi)$$

Axioms:

$$x = m \wedge \mathbf{emp} \rightarrow \llbracket x := E \rrbracket x = E[x \leftarrow m] \wedge \mathbf{emp}$$

$$x = m \wedge E \mapsto n \rightarrow \llbracket x := [E] \rrbracket (x = n \wedge E[x \leftarrow m] \mapsto n)$$

$$(E \mapsto n) \rightarrow \llbracket [E] := F \rrbracket E \mapsto F$$

Heap location must be accessible

Recall: $s \models \llbracket \pi \rrbracket \varphi$ iff $s' \models \varphi$ for all $(s, s') \in \rho(\pi)$ and $s \notin \mathit{fail}(\pi)$.
All accessed heap locations (read or write) must be in domain.

Therefore: Precondition must ensure that.

Axioms:

$$x = m \wedge \mathbf{emp} \rightarrow \llbracket x := E \rrbracket x = E[x \leftarrow m] \wedge \mathbf{emp}$$

$$x = m \wedge E \mapsto n \rightarrow \llbracket x := [E] \rrbracket (x = n \wedge E[x \leftarrow m] \mapsto n)$$

$$(\exists n. E \mapsto n) \rightarrow \llbracket [E] := F \rrbracket E \mapsto F$$

Heap location must be accessible

Recall: $s \models \llbracket \pi \rrbracket \varphi$ iff $s' \models \varphi$ for all $(s, s') \in \rho(\pi)$ and $s \notin \text{fail}(\pi)$.
All accessed heap locations (read or write) must be in domain.

Therefore: Precondition must ensure that.

THIS IS THE KEY POINT ABOUT SEPARATION LOGIC

$$\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P * R \rightarrow \llbracket \pi \rrbracket (Q * R)}$$

$$\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$$

THIS IS THE KEY POINT ABOUT SEPARATION LOGIC

$$\frac{P \quad \rightarrow \llbracket \pi \rrbracket Q}{P * R \rightarrow \llbracket \pi \rrbracket (Q * R)}$$

$$\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$$

Separation in Proofs

Proof $P \rightarrow \llbracket \pi \rrbracket Q$ using in P, Q the memory π refers to.

Get for free: Nothing besides these memory locations has changed.

Remember: The Framing Problem

Example in Java

```
//@ requires acc1 != acc2;  
//@ ensures \result == 100;  
int f(Account acc1, Account acc2) {  
    acc1.setBalance(100);  
    acc2.setBalance(200);  
    return acc1.getBalance();  
}
```

Rule for setBalance:

$$A \mapsto x \rightarrow \llbracket A.setBalance(y) \rrbracket A \mapsto y$$

Remember: The Framing Problem

Example in Java

```
//@ requires acc1 != acc2;  
/*@ ensures \result == 100;  
int f(Account acc1, Account acc2) {  
    acc1.setBalance(100);  
    acc2.setBalance(200);  
    return acc1.getBalance();  
}
```

Rule for setBalance:

$$A \mapsto x \rightarrow \llbracket A.setBalance(y) \rrbracket A \mapsto y$$

Use Frame Rule:

$$acc2 \mapsto x \rightarrow \dots$$

$$\dots \llbracket acc2.setBalance(200); \rrbracket acc2 \mapsto 200$$

Remember: The Framing Problem

Example in Java

```
//@ requires acc1 != acc2;  
//@ ensures \result == 100;  
int f(Account acc1, Account acc2) {  
    acc1.setBalance(100);  
    acc2.setBalance(200);  
    return acc1.getBalance();  
}
```

Rule for setBalance:

$$A \mapsto x \rightarrow \llbracket A.setBalance(y) \rrbracket A \mapsto y$$

Use Frame Rule:

$$\begin{aligned} acc2 \mapsto x * acc1 \mapsto 100 &\rightarrow \dots \\ \dots \llbracket acc2.setBalance(200); \rrbracket acc2 \mapsto 200 * acc1 \mapsto 100 \end{aligned}$$

On the board ...

$$(\exists v. X \mapsto v * Y \mapsto v) \rightarrow \llbracket X := [X]; Y := [Y] \rrbracket X = Y$$

Soundness of Frame Rule

$$\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P * R \rightarrow \llbracket \pi \rrbracket (Q * R)}$$

or equivalently

$$\models (\llbracket \pi \rrbracket Q) * R \rightarrow \llbracket \pi \rrbracket (Q * R)$$

if $\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$

Soundness of Frame Rule

$$\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P * R \rightarrow \llbracket \pi \rrbracket (Q * R)}$$

or equivalently

$$\models (\llbracket \pi \rrbracket Q) * R \rightarrow \llbracket \pi \rrbracket (Q * R)$$

if $\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$



Instantiate left rule with $P := \llbracket \pi \rrbracket Q$.

Premiss: trivially true, conclusion: desired implication.

Soundness of Frame Rule

$$\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P * R \rightarrow \llbracket \pi \rrbracket (Q * R)}$$

or equivalently

$$\models (\llbracket \pi \rrbracket Q) * R \rightarrow \llbracket \pi \rrbracket (Q * R)$$

if $\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$



Instantiate left rule with $P := \llbracket \pi \rrbracket Q$.

Premiss: trivially true, conclusion: desired implication.



Let $\beta, h \models P * R$, i.e., $\beta, h_1 \models P$ and $\beta, h_2 \models R$ with $h = h_1 \uplus h_2$.

By premiss: $\beta, h_1 \models \llbracket \pi \rrbracket Q$ and $\beta, h \models (\llbracket \pi \rrbracket Q) * R$

Right rule gives: $\beta, h \models \llbracket \pi \rrbracket (Q * R)$

Soundness of Frame Rule

$$\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P * R \rightarrow \llbracket \pi \rrbracket (Q * R)}$$

or equivalently

$$\models (\llbracket \pi \rrbracket Q) * R \rightarrow \llbracket \pi \rrbracket (Q * R)$$

if $\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$



Instantiate left rule with $P := \llbracket \pi \rrbracket Q$.

Premiss: trivially true, conclusion: desired implication.



Let $\beta, h \models P * R$, i.e., $\beta, h_1 \models P$ and $\beta, h_2 \models R$ with $h = h_1 \uplus h_2$.

By premiss: $\beta, h_1 \models \llbracket \pi \rrbracket Q$ and $\beta, h \models (\llbracket \pi \rrbracket Q) * R$

Right rule gives: $\beta, h \models \llbracket \pi \rrbracket (Q * R)$



Lemma

Let $h_1, h'_1, h_2 : \mathbb{N} \rightarrow \mathbb{N}$ be heaplets, $\text{dom } h_1 \cap \text{dom } h_2 = \emptyset$
 $\beta, \beta' : \text{Var} \rightarrow \mathbb{N}$ be variable assignments. Then:

$$\beta, h_1 \xrightarrow{\pi} \beta', h'_1 \quad \Longrightarrow \quad \beta, h_1 \uplus h_2 \xrightarrow{\pi} \beta', h'_1 \uplus h_2$$

$s \xrightarrow{\pi} s'$ means $(s, s') \in \rho(\pi)$

Lemma

Let $h_1, h'_1, h_2 : \mathbb{N} \mapsto \mathbb{N}$ be heaplets, $\text{dom } h_1 \cap \text{dom } h_2 = \emptyset$
 $\beta, \beta' : \text{Var} \rightarrow \mathbb{N}$ be variable assignments. Then:

$$\beta, h_1 \xrightarrow{\pi} \beta', h'_1 \quad \Longrightarrow \quad \beta, h_1 \uplus h_2 \xrightarrow{\pi} \beta', h'_1 \uplus h_2$$

By structural induction:

- variable assignment $v := t$
- heap store $[t_1] := t_2$
- heap load $v := [t]$
- test $?\varphi$
- sequential $\pi_1 ; \pi_2$
- Kleene π^*

(hint, proof as exercise)

(heap irrelevant)

($\text{val}(t) \stackrel{!}{\in} \text{dom } h_1$)

($\text{val}(t) \stackrel{!}{\in} \text{dom } h_1$)

(heap irrelevant)

(appeal to ind. hyp)

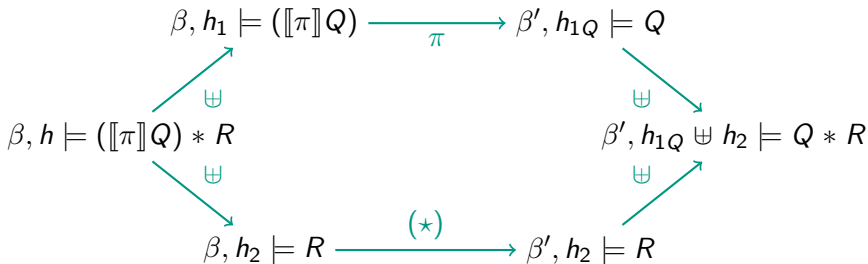
(appeal to ind. hyp)

$s \xrightarrow{\pi} s'$ means $(s, s') \in \rho(\pi)$

Soundness of Frame Rule

$$\models ([[\pi]]Q) * R \rightarrow [[\pi]](Q * R) \quad \text{if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset \quad (\star)$$

Let $\beta, h \models ([[\pi]]Q) * R$, i.e., $\beta, h_1 \models [[\pi]]Q$ and $\beta, h_2 \models R$, $h = h_1 \uplus h_2$.



Lemma, lang. is deterministic $\implies \beta, h \models [[\pi]](Q * R)$

Memory Allocation and Deallocation

Syntax: Two statements

`var := cons(term, ..., term)` and `dispose(var)`

Syntax: Two statements

$\text{var} := \text{cons}(\text{term}, \dots, \text{term})$ and $\text{dispose}(\text{var})$

Semantics: ρ and $fail$

$$((\beta, h), (\beta', h')) \in \rho(v := \text{cons}(t))$$

iff

$$\beta' = \beta[v/loc] \text{ and } h' = h \uplus \{(loc, val_{\beta}(t))\} \text{ and } loc \notin \text{dom } h$$

$$fail(v := \text{cons}(t)) = \emptyset$$

`cons` allocates n consecutive unused memory locations, stores the argument values there and returns the first memory location.

(See literature for general n -ary version)

Syntax: Two statements

$\text{var} := \text{cons}(\text{term}, \dots, \text{term})$ and $\text{dispose}(\text{var})$

Semantics: ρ and *fail*

$((\beta, h), (\beta', h')) \in \rho(\text{dispose}(v))$
iff

$\beta' = \beta$ and $\beta(v) \in \text{dom } h$ and $h' = h \setminus \{(\beta(v), h(\beta(v)))\}$

$\text{fail}(\text{dispose}(v)) = \{(\beta, h) \mid \beta(v) \notin \text{dom } h\}$

`dispose` deallocates the allocated memory location v ;
fails if an unallocated location is disposed.

$$\frac{(\llbracket \pi \rrbracket Q) * R}{\llbracket \pi \rrbracket (Q * R)} \text{ if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset$$

Proof by structural induction over π .

see Reynolds p.77ff

Decidable

Some restricted logics from Separation Logic are decidable.

- ① Restricted arithmetic
- ② No magic wand \rightarrow^*

They can be reduced to Monadic Second Order Logic over \mathbb{N} .
Equivalent to word emptiness of Büchi Automata.

The separating implication \rightarrow^* makes undecidable.

Relatively complete

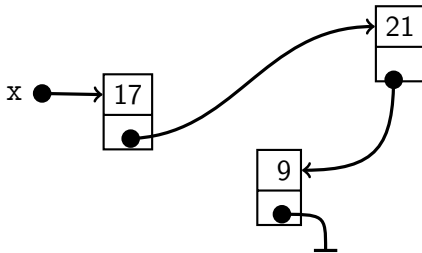
The calculus for Separation Logic is relatively complete.
Every correct program can be proved using an oracle for \mathbb{N} .

Application of Separation Logic

Abstraction Predicates

Use predicate symbols to abstract away from data structures

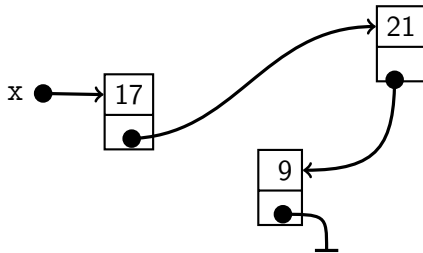
Example: Lists



Use predicate symbols to abstract away from data structures

Example: Lists

$$\text{list}(x, \langle 17, 21, 9 \rangle) \Leftrightarrow (x \mapsto 17) * (x+1 \mapsto v) * (v \mapsto 21) * \dots \\ \dots * (v+1 \mapsto w) * (w \mapsto 9) * (w+1 \mapsto 0)$$



Use predicate symbols to abstract away from data structures

Example: Lists

$$\begin{aligned} list(x, \langle 17, 21, 9 \rangle) \leftrightarrow & (x \mapsto 17) * (x+1 \mapsto v) * (v \mapsto 21) * \dots \\ & \dots * (v+1 \mapsto w) * (w \mapsto 9) * (w+1 \mapsto 0) \end{aligned}$$

General:

Recursive predicate *list*:

$$\forall x, v_1, \bar{v}. list(x, \langle v_1, \bar{v} \rangle) \leftrightarrow \exists n. ((x \mapsto v_1) * (x+1 \mapsto n) * list(n, \bar{v}))$$

- Verifast → Demo! (Bart Jacobs *et al.*, U Leuven)
<https://www.cs.kuleuven.be/~bartj/verifast/>

- Verifast → Demo! (Bart Jacobs *et al.*, U Leuven)
<https://www.cs.kuleuven.be/~bartj/verifast/>
- Infer (Peter O'Hearn *et al.*, Facebook)
<http://fbinfer.com/>

- Verifast → Demo! (Bart Jacobs *et al.*, U Leuven)
<https://www.cs.kuleuven.be/~bartj/verifast/>
- Infer (Peter O'Hearn *et al.*, Facebook)
<http://fbinfer.com/>
- jStar (M. Parkinson, now MS)

- Verifast → [Demo!](#) (Bart Jacobs *et al.*, U Leuven)
<https://www.cs.kuleuven.be/~bartj/verifast/>
- Infer (Peter O'Hearn *et al.*, Facebook)
<http://fbinfer.com/>
- jStar (M. Parkinson, now MS)
- SpacInvader, YNot, HOLFoot, . . . , . . .

Advantages of Separation Logic

- + Functional and frame specification combined – no extra consideration needed
- + Frame rule!
- + Abstraction Predicates are nice way of abstraction

Disadvantages of Separation Logic

- Functional and frame specification combined – no separation of concerns!
- All data must be hierarchically structured
- Complicated semantics of Sep Logic (c.f. \rightarrow^*)