Formal Systems II: Theory

Separation Logic

SS 2022

Mattias Ulbrich
Institute of Theoretical Informatics

Motivation

Reminder . . . Dynamic Logic

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.

The Framing Problem

Formal Software Verification

- Prove what effects a program has.

The Framing Problem

Formal Software Verification

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

The Framing Problem

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.

The Framing Problem

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.

The Framing Problem

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.

The Framing Problem

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

The Framing Problem

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.

The Framing Problem

Example in Java

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

The Framing Problem

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();
}
```

The Framing Problem

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();
}
```

The Framing Problem

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

The Framing Problem

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

The Framing Problem

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

The Framing Problem

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.

The Framing Problem

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 the specifying user (needs to think about not-effects) and for reasoning engines (increased complexity).

The Framing Problem

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 the 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/Region Logic (Kassios 2006)
- . . .

The Framing Problem

Karlsruhe Institute of Technology

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 the specifying user (needs to think about not-effects) and for reasoning engines (increased complexity).

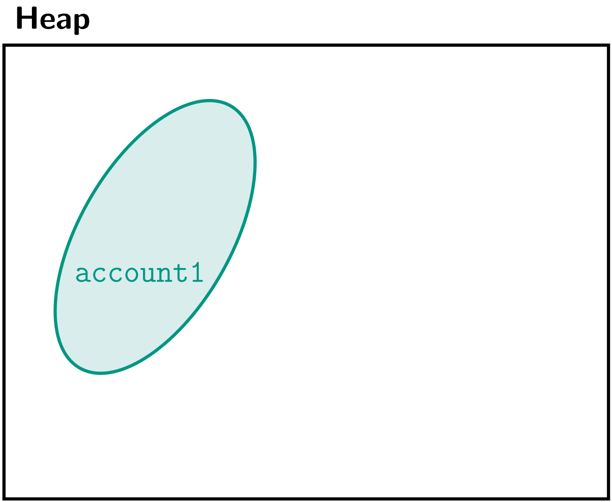
Suggested solutions:

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

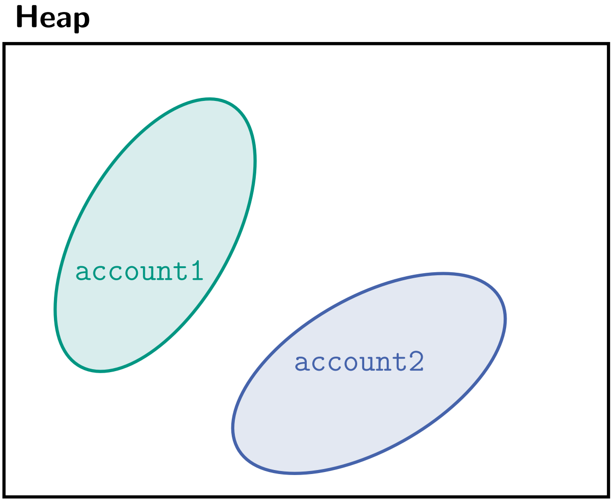
Heaps and “Footprints”

Heap

Heaps and “Footprints”

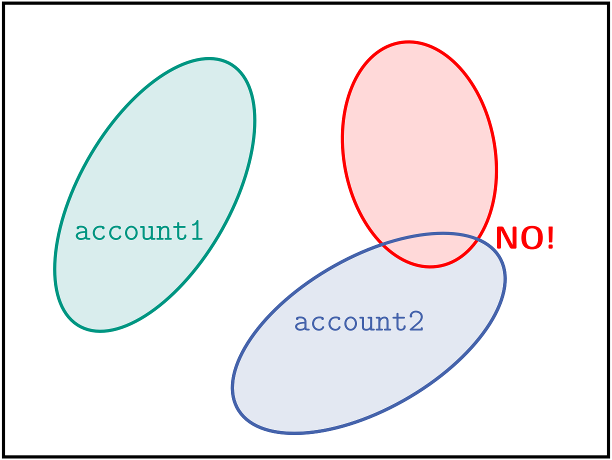


Heaps and “Footprints”



Heaps and “Footprints”

Heap



Heaps and Heaplets

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:

$$\mathit{heap} : \mathbb{N} \to \mathbb{N}$$

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

$$\mathit{heaplet} : \mathbb{N} \nrightarrow \mathbb{N}$$

Partial function:

Partial function $f : A \nrightarrow B$ is a function $f : D \to B$ for $D \subseteq A$.
The finite set $D = \operatorname{dom} f$ is called the domain of f.

Operations and Heaps

Disjoint union of heaplets:

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

$h_1 \uplus h_2$ 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

Separation Logic – Syntax

Terms t:

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

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

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

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

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

- $\varphi_1 \wedge \varphi_2, \quad \varphi_1 \vee \varphi_2, \quad \varphi_1 \to \varphi_2$
- $t_1 = t_2, \quad t_1 < t_2, \; \ldots$
- $\forall x.\varphi, \quad \exists x.\varphi$

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

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

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

- $\varphi_1 \wedge \varphi_2, \quad \varphi_1 \vee \varphi_2, \quad \varphi_1 \to \varphi_2$
- $t_1 = t_2, \quad t_1 < t_2, \; \ldots$
- $\forall x.\varphi, \quad \exists x.\varphi$
- $\varphi_1 * \varphi_2$
- emp

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

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

new in Separation Logic

Separation Logic – Syntax

Terms t:

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

Formulae φ:

- $\varphi_1 \wedge \varphi_2, \quad \varphi_1 \vee \varphi_2, \quad \varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2, \quad t_1 < t_2, \ \ldots$
- $\forall x.\varphi, \quad \exists x.\varphi$
- $\varphi_1 * \varphi_2$
- emp
- $t_1 \mapsto t_2$
- $\varphi_1 \mathrel{-\!\!*} \varphi_2$ (later)

new in Separation Logic

Operator Precedence

How are the implicit parentheses in
$B \mathrel{-\!\!*} C \wedge D \vee A * x \mapsto y$?

Operator Precedence

How are the implicit parentheses in
$B \mathrel{-\!\!*} C \wedge D \vee A * x \mapsto y$?

Binding force:

$*$	binds like $\wedge$
$\mathrel{-\!\!*}$	binds like $\rightarrow$, $\vee$
$\mapsto$	binds like $=$

Operator Precedence

How are the implicit parentheses in
$B \mathrel{-\!\!*} C \wedge D \vee A * x \mapsto y$?

Binding force:

$*$	binds like $\wedge$
$\mathrel{-\!\!*}$	binds like $\rightarrow$, $\vee$
$\mapsto$	binds like $=$

Answer:

$$\big(B \mathrel{-\!\!*} (C \wedge D)\big) \vee \big(A * (x \mapsto y)\big)$$

or $$B \mathrel{-\!\!*} \Big((C \wedge D) \vee \big(A * (x \mapsto y)\big)\Big)$$

Operator Precedence

How are the implicit parentheses in
$B \mathrel{-\!\!*} C \wedge D \vee A * x \mapsto y$?

Binding force:

$*$	binds like $\wedge$
$\mathrel{-\!\!*}$	binds like $\rightarrow$, $\vee$
$\mapsto$	binds like $=$

Answer:

$$\big(B \mathrel{-\!\!*} (C \wedge D)\big) \vee \big(A * (x \mapsto y)\big)$$

or $$B \mathrel{-\!\!*} \Big((C \wedge D) \vee \big(A * (x \mapsto y)\big)\Big)$$

Add explicit parentheses when combining $\vee / \rightarrow / \mathrel{-\!\!*}$ or $\wedge / *$

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

Terms:

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

Formulas in FOL:

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

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

- $\beta, h \models \mathrm{emp}$ iff $\mathrm{dom}\, h = \emptyset$

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

- $\beta, h \models \text{emp} \quad \text{iff} \quad \text{dom}\, h = \emptyset$
- $\beta, h \models t_1 \mapsto t_2 \quad \text{iff} \quad h = \{(val_\beta(t_1), val_\beta(t_2))\}$

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

- $\beta, h \models \mathrm{emp}$ iff $\mathrm{dom}\, h = \emptyset$
- $\beta, h \models t_1 \mapsto t_2$ iff $h = \{(\mathit{val}_\beta(t_1), \mathit{val}_\beta(t_2))\}$
- $\beta, h \models \varphi_1 * \varphi_2$ iff there exist heaplets $h_1, h_2 : \mathbb{N} \nrightarrow \mathbb{N}$ with

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \rightharpoonup \mathbb{N}$

- $\beta, h \models \text{emp}$ 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 $h_1, h_2 : \mathbb{N} \rightharpoonup \mathbb{N}$ with
  1. $h = h_1 \uplus h_2$ and

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : Var \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

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

Separation Logic – Semantics

Structure

Fixed first order domain: $\mathbb{N}$.

Terms and formulas are evaluated over:

1. Variable assignment $\beta : \mathit{Var} \to \mathbb{N}$
2. Heaplet $h : \mathbb{N} \nrightarrow \mathbb{N}$

- $\beta, h \models \text{emp}$ 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 $h_1, h_2 : \mathbb{N} \nrightarrow \mathbb{N}$ with
  1. $h = h_1 \uplus h_2$ and
  2. $\beta, h_1 \models \varphi_1$ and
  3. $\beta, h_2 \models \varphi_2$

Separation Logic

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.

Properties of Separation Logic

Idempotence

- $\models A \leftrightarrow A \wedge A$ (idempotence for $\wedge$)

Properties of Separation Logic

Idempotence

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

Properties of Separation Logic

Idempotence

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

Properties of Separation Logic

Idempotence

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

  $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Properties of Separation Logic

Idempotence

- $\models A \;\leftrightarrow\; A \land A$ (idempotence for $\land$)
- $\overset{?}{\models} A \;\leftrightarrow\; A * A$ (idempotence also for $*$?)
- NO! Counterexample:

$$\models \neg(7 \mapsto 3 \;\rightarrow\; 7 \mapsto 3 * 7 \mapsto 3)$$

Weakening

Properties of Separation Logic

Idempotence

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

  $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \land B \rightarrow A$ (Weakining of conjunction)

Properties of Separation Logic

Idempotence

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

  $\models \neg(7 \mapsto 3 \;\rightarrow\; 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \wedge B \;\rightarrow\; A$ (Weakining of conjunction)
- $\overset{?}{\models} A * B \;\rightarrow\; A$ (Weakining of separating conjunction?)

Properties of Separation Logic

Idempotence

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

  $\models \neg(7 \mapsto 3 \;\rightarrow\; 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \wedge B \;\rightarrow\; A$ (Weakining of conjunction)
- $\overset{?}{\models} A * B \;\rightarrow\; A$ (Weakining of separating conjunction?)
- NO! Counterexample:

Properties of Separation Logic

Idempotence

- $\models A \;\leftrightarrow\; A \land A$ (idempotence for $\land$)
- $\overset{?}{\models} A \;\leftrightarrow\; A * A$ (idempotence also for $*$?)
- NO! Counterexample:

  $\models \neg(7 \mapsto 3 \;\rightarrow\; 7 \mapsto 3 * 7 \mapsto 3)$

Weakening

- $\models A \land B \;\rightarrow\; A$ (Weakining of conjunction)
- $\overset{?}{\models} A * B \;\rightarrow\; A$ (Weakining 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 $(val(A), val(B)) \in h$

Caution

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

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

On the other hand:

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

Caution

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

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

On the other hand:

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

Caution

$\beta, h \models A \mapsto B$ 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: $A \hookrightarrow B \;:\leftrightarrow\; A \mapsto B * true$

Some Valid Formulas

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

Some Valid Formulas

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

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

Some Valid Formulas

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

Some Valid Formulas

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

Quiz!

KIT
Karlsruhe Institute of Technology

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

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

Quiz!

Karlsruhe Institute of Technology

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \wedge x \mapsto z$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \land x \mapsto z$
3. $(x \mapsto 0 \land y \mapsto 0) \to x = y$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \wedge x \mapsto z$
3. $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$
4. $(x \mapsto 0 * y \mapsto 0) \rightarrow x = y$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \land x \mapsto z$
3. $(x \mapsto 0 \land y \mapsto 0) \to x = y$
4. $(x \mapsto 0 * y \mapsto 0) \to x = y$
5. $(x \mapsto 0 * y \mapsto 0) \to \neg(x = y)$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \wedge x \mapsto z$
3. $(x \mapsto 0 \wedge y \mapsto 0) \to x = y$
4. $(x \mapsto 0 * y \mapsto 0) \to x = y$
5. $(x \mapsto 0 * y \mapsto 0) \to \neg(x = y)$
6. $(x \mapsto a \wedge y \mapsto b) \to a = b$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \land x \mapsto z$
3. $(x \mapsto 0 \land y \mapsto 0) \to x = y$
4. $(x \mapsto 0 * y \mapsto 0) \to x = y$
5. $(x \mapsto 0 * y \mapsto 0) \to \neg(x = y)$
6. $(x \mapsto a \land y \mapsto b) \to a = b$
7. $\varphi * \mathrm{emp} \to \varphi$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \wedge x \mapsto z$
3. $(x \mapsto 0 \wedge y \mapsto 0) \to x = y$
4. $(x \mapsto 0 * y \mapsto 0) \to x = y$
5. $(x \mapsto 0 * y \mapsto 0) \to \neg(x = y)$
6. $(x \mapsto a \wedge y \mapsto b) \to a = b$
7. $\varphi * \mathrm{emp} \to \varphi$
8. $\varphi * \neg\varphi$

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \land x \mapsto z$
3. $(x \mapsto 0 \land y \mapsto 0) \to x = y$
4. $(x \mapsto 0 * y \mapsto 0) \to x = y$
5. $(x \mapsto 0 * y \mapsto 0) \to \neg(x = y)$
6. $(x \mapsto a \land y \mapsto b) \to a = b$
7. $\varphi * \text{emp} \to \varphi$
8. $\varphi * \neg\varphi$

a. $\psi * \neg\psi$ for ψ without $\mapsto$, emp

Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

(here: A formula φ is satisfiable iff are there β and h such that $\beta, h \models \varphi$.)

1. $x \mapsto y * x \mapsto z$
2. $x \mapsto y \land x \mapsto z$
3. $(x \mapsto 0 \land y \mapsto 0) \to x = y$
4. $(x \mapsto 0 * y \mapsto 0) \to x = y$
5. $(x \mapsto 0 * y \mapsto 0) \to \neg(x = y)$
6. $(x \mapsto a \land y \mapsto b) \to a = b$
7. $\varphi * \mathrm{emp} \to \varphi$
8. $\varphi * \neg\varphi$
   a. $\psi * \neg\psi$ for ψ without $\mapsto$, emp
   b. $x \mapsto y * \neg(x \mapsto y)$

The Magic Wand

Modus Ponens for classical logic

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

The Magic Wand

Modus Ponens for classical logic

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

Corresponding rule for separating conjunction $*$?

The Magic Wand

Modus Ponens for classical logic

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

Corresponding rule for separating conjunction $*$?

Modus Ponens for separation logic

$$\frac{A * (A \mathrel{-\!\!*} B)}{B}$$

The **magic wand operator** $A \mathrel{-\!\!*} B$, aka **separating implication**:

$$\beta, h \models A \mathrel{-\!\!*} B$$

$$\iff$$

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

Separating Operators

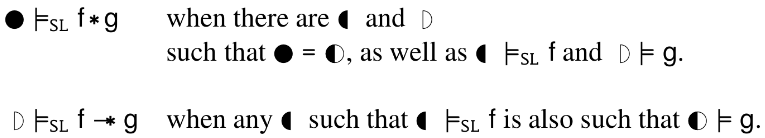


Figure 1.5: Visual representation of the semantics of separation operators

Taken from:
Separation Logic: Expressiveness, Complexity, Temporal Extension
Rémi Brochenin, PhD Thesis. 2013

Programs and Separation Logic

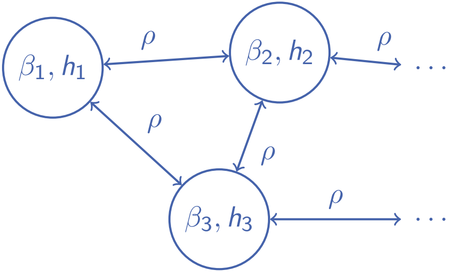
Programming Language

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`)`

Restriction: formula are the arithmetic formulas that do not contain $\mapsto$ or emp.

Kripke Frames with Heaps

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



Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

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

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

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

$$\rho(\pi_1 \,;\, \pi_2) = \rho(\pi_1) \,;\, \rho(\pi_2)$$

; is forward composition

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

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

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

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

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

; is forward composition

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

* is refl. transitive closure

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

$$
\begin{aligned}
\rho(\pi_1 \cup \pi_2) &= \rho(\pi_1) \cup \rho(\pi_2) \\
\rho(\pi_1 \,;\, \pi_2) &= \rho(\pi_1) \,;\, \rho(\pi_2) \qquad \text{; is forward composition} \\
&= \{(s,t) \mid \text{ex. } u \in S \text{ with } (s,u) \in \rho(\pi_1), (u,t) \in \rho(\pi_2)\} \\
\rho(\pi^*) &= \rho(\pi)^* \qquad \text{* is refl. transitive closure} \\
&= \{(s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\}
\end{aligned}
$$

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

$$
\begin{aligned}
\rho(\pi_1 \cup \pi_2) &= \rho(\pi_1) \cup \rho(\pi_2) \\
\rho(\pi_1 \,;\, \pi_2) &= \rho(\pi_1) \,;\, \rho(\pi_2) \qquad \text{; is forward composition} \\
&= \{(s,t) \mid \text{ex. } u \in S \text{ with } (s,u) \in \rho(\pi_1), (u,t) \in \rho(\pi_2)\} \\
\rho(\pi^*) &= \rho(\pi)^* \qquad \text{* is refl. transitive closure} \\
&= \{(s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\} \\
\rho(\mathbf{?}\varphi) &= \{(s,s) \mid s \models \varphi\}
\end{aligned}
$$

Program semantics (repetition from FODL)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

$$
\begin{aligned}
\rho(\pi_1 \cup \pi_2) &= \rho(\pi_1) \cup \rho(\pi_2) \\
\rho(\pi_1 \,;\, \pi_2) &= \rho(\pi_1) \,;\, \rho(\pi_2) \qquad \text{; is forward composition} \\
&= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \\
\rho(\pi^*) &= \rho(\pi)^* \qquad \text{* is refl. transitive closure} \\
&= \{(s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\} \\
\rho(\mathbf{?}\varphi) &= \{(s, s) \mid s \models \varphi\}
\end{aligned}
$$

Reminder: IF and WHILE

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

Program semantics (with heap)

Accessiblity Relation for Programs

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

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

Program semantics (with heap)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$
A state $s \in S$ is a pair (β, h) with $\beta : Var \to \mathbb{N}$ and $h : \mathbb{N} \nrightarrow \mathbb{N}$

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

Program semantics (with heap)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

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

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

$$\big((\beta, h), (\beta', h')\big) \in \rho(v := [t]) \iff \begin{array}{l} val_\beta(t) \in \mathrm{dom}\, h \text{ and } h' = h \text{ and} \\ \beta' = \beta\big[v/h[val_\beta(t)]\big] \end{array}$$

Program semantics (with heap)

Accessiblity Relation for Programs

$\rho : \texttt{statement} \to S \times S$

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

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

$$\big((\beta, h), (\beta', h')\big) \in \rho(v := [t]) \iff \begin{array}{l} val_\beta(t) \in \operatorname{dom} h \text{ and } h' = h \text{ and} \\ \beta' = \beta\big[v/h[val_\beta(t)]\big] \end{array}$$

$$\big((\beta, h), (\beta', h')\big) \in \rho([t] := u) \iff \begin{array}{l} val_\beta(t) \in \operatorname{dom} h \text{ and } \beta' = \beta \text{ and} \\ h' = h[val_\beta(t)/val_\beta(u)] \end{array}$$

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

Failing executions

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

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

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

Failing executions

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

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

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

Model unallowed heap access:

$\mathit{fail} : \texttt{statement} \to 2^S$

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

Failing executions

Model unallowed heap access:

$fail : \texttt{statement} \to 2^S$

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

$$\begin{aligned} fail(x := t) &= \\ fail(\mathbf{?}\psi) &= \emptyset \end{aligned}$$

Failing executions

Model unallowed heap access:

$\mathit{fail} : \texttt{statement} \to 2^S$

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

$$
\begin{aligned}
\mathit{fail}(x := t) &= \\
\mathit{fail}(\mathbf{?}\psi) &= \emptyset \\[1em]
\mathit{fail}(x := [t]) &= \\
\mathit{fail}([t] := u) &= \{(\beta, h) \mid \mathit{val}_\beta(t) \notin \mathrm{dom}\, h\}
\end{aligned}
$$

Failing executions

Model unallowed heap access:

$\mathit{fail} : \texttt{statement} \to 2^S$

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

$$
\begin{aligned}
\mathit{fail}(x := t) &= \\
\mathit{fail}(?\psi) &= \emptyset \\
\\
\mathit{fail}(x := [t]) &= \\
\mathit{fail}([t] := u) &= \{(\beta, h) \mid \mathit{val}_\beta(t) \notin \mathrm{dom}\, h\} \\
\\
\mathit{fail}(\pi_1 \,;\, \pi_2) &= \mathit{fail}(\pi_1) \cup \big(\rho(\pi_1) \,;\, \mathit{fail}(\pi_2)\big) \\
\\
\mathit{fail}(\pi_1 \cup \pi_2) &= \mathit{fail}(\pi_1) \cup \mathit{fail}(\pi_2)
\end{aligned}
$$

Failing executions

Model unallowed heap access:

$\mathit{fail} : \texttt{statement} \to 2^S$

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

$$
\begin{aligned}
\mathit{fail}(x := t) &= \\
\mathit{fail}(\mathbf{?}\psi) &= \emptyset \\
\\
\mathit{fail}(x := [t]) &= \\
\mathit{fail}([t] := u) &= \{(\beta, h) \mid \mathit{val}_\beta(t) \notin \operatorname{dom} h\} \\
\\
\mathit{fail}(\pi_1 \,;\, \pi_2) &= \mathit{fail}(\pi_1) \cup \big(\rho(\pi_1) \,;\, \mathit{fail}(\pi_2)\big) \\
\\
\mathit{fail}(\pi_1 \cup \pi_2) &= \mathit{fail}(\pi_1) \cup \mathit{fail}(\pi_2) \\
\\
\mathit{fail}(\pi^*) &= \rho(\pi^*) \,;\, \mathit{fail}(\pi)
\end{aligned}
$$

Failing executions

Model unallowed heap access:

$\mathit{fail} : \mathtt{statement} \to 2^S$

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

$$
\begin{array}{rcl}
\mathit{fail}(x := t) & = & \\
\mathit{fail}(\mathbf{?}\psi) & = & \emptyset \\
\\
\mathit{fail}(x := [t]) & = & \\
\mathit{fail}([t] := u) & = & \{(\beta, h) \mid \mathit{val}_\beta(t) \notin \mathrm{dom}\, h\} \\
\\
\mathit{fail}(\pi_1 \,;\, \pi_2) & = & \mathit{fail}(\pi_1) \cup \big(\rho(\pi_1) \,;\, \mathit{fail}(\pi_2)\big) \\
\\
\mathit{fail}(\pi_1 \cup \pi_2) & = & \mathit{fail}(\pi_1) \cup \mathit{fail}(\pi_2) \\
\\
\mathit{fail}(\pi^*) & = & \rho(\pi^*) \,;\, \mathit{fail}(\pi)
\end{array}
$$

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

Fail-aware modality

Remember:

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

Problem:

$\text{emp} \to \big[[5] := 42\big]\,\mathit{false} \qquad$ is a valid formula.

New modality $[\![\cdot]\!]$

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

Now:

$\text{emp} \to [\![[5] := 42]\!]\,\psi \qquad$ is not valid for any ψ

Dynamic Separation Logic

Valid formulas:

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

A Calculus for Separation Logic

Hoare Calculus

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

A Calculus for Separation Logic

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 (it it terminates).

A Calculus for Separation Logic

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 (it it terminates).

Semantically the same as $P \to [\![\pi]\!]Q$.

A Calculus for Separation Logic

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 (it it terminates).

Semantically the same as $P \to [\![\pi]\!]Q$.

We present the calculus using dynamic logic notation.

Reminder: Hoare Calculus (in DL notation)

$$\frac{}{P[x\leftarrow E] \to [\![x := E]\!]P} \quad [x\leftarrow E] \text{ is substitution}$$

$$\frac{P \to [\![\pi_1]\!]Q \qquad Q \to [\![\pi_2]\!]R}{P \to [\![\pi_1\,;\,\pi_2]\!]R} \qquad \frac{P' \to P \qquad P \to [\![\pi]\!]Q \qquad Q \to Q'}{P' \to [\![\pi]\!]Q'}$$

$$\frac{P \land C \to [\![\pi_1]\!]Q \qquad P \land \neg C \to [\![\pi_2]\!]Q}{P \to [\![\text{if } C \text{ then } \pi_1 \text{ else } \pi_2]\!]Q}$$

$$\frac{P \land C \to [\![\pi]\!]P}{P \to [\![\text{while } C \text{ do } \pi]\!](P \land \neg C)}$$

$$\frac{P \to [\![\pi]\!]Q}{(\exists x.P) \to [\![\pi]\!](\exists x.Q)} \text{if } x \notin \mathit{Free}(\pi)$$

Separation Logic Rules for Assignments

Axioms:

$$x = m \land \mathsf{emp} \;\rightarrow\; [\![x := E]\!] x = E[x \leftarrow m] \land \mathsf{emp}$$

$$x = m \land E \mapsto n \;\rightarrow\; [\![x := [E]]\!](x = n \land E[x \leftarrow m] \mapsto n)$$

$$(E \mapsto n) \;\rightarrow\; [\![[E] := F]\!]\; E \mapsto F$$

Heap location must be accessible

Recall: $s \models [\![\pi]\!]\varphi$ 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.

Separation Logic Rules for Assignments

Axioms:

$$x = m \land \mathsf{emp} \;\to\; [\![x := E]\!] x = E[x \leftarrow m] \land \mathsf{emp}$$

$$x = m \land E \mapsto n \;\to\; [\![x := [E]]\!] (x = n \land E[x \leftarrow m] \mapsto n)$$

$$(\exists n. E \mapsto n) \;\to\; [\![[E] := F]\!]\; E \mapsto F$$

Heap location must be accessible

Recall: $s \models [\![\pi]\!]\varphi$ 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.

The Frame Rule

THIS IS THE KEY POINT ABOUT SEPARATION LOGIC

$$\frac{P \quad\quad \to [\![\pi]\!]\; Q}{P * R \to [\![\pi]\!](Q * R)}$$

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

The Frame Rule

THIS IS THE KEY POINT ABOUT SEPARATION LOGIC

$$\frac{P \quad \to [\![\pi]\!]\ Q}{P * R \to [\![\pi]\!](Q * R)}$$

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

Separation in Proofs

Proof $P \to [\![\pi]\!]Q$ 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:

$$A \mapsto x \;\rightarrow\; [\![A.setBalance(y)]\!] A \mapsto y$$

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:

$$A \mapsto x \;\rightarrow\; [\![A.setBalance(y)]\!] A \mapsto y$$

Use Frame Rule:

$$acc2 \mapsto x \;\rightarrow\; \ldots$$

$$\ldots [\![acc2.setBalance(200);]\!] acc2 \mapsto 200$$

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:

$$A \mapsto x \;\rightarrow\; [\![A.setBalance(y)]\!] A \mapsto y$$

Use Frame Rule:

$$acc2 \mapsto x * acc1 \mapsto 100 \;\rightarrow\; \dots$$

$$\dots [\![acc2.setBalance(200);]\!] acc2 \mapsto 200 * acc1 \mapsto 100$$

On the board ...

$$(\exists v.\ \mathtt{X} \mapsto v * \mathtt{Y} \mapsto v) \;\to\; [\![\mathtt{X} := [\mathtt{X}]\,;\mathtt{Y} := [\mathtt{Y}]]\!]\ \mathtt{X} = \mathtt{Y}$$

Soundness of Frame Rule

$$\frac{P \to [\![\pi]\!]Q}{P * R \to [\![\pi]\!](Q * R)}$$

or equivalently

$$\models ([\![\pi]\!]Q) * R \;\to\; [\![\pi]\!](Q * R)$$

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

Soundness of Frame Rule

$$\frac{P \to [\![\pi]\!]Q}{P * R \to [\![\pi]\!](Q * R)}$$

or equivalently

$$\models ([\![\pi]\!]Q) * R \ \to\ [\![\pi]\!](Q * R)$$

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

$\Longrightarrow$

Instantiate left rule with $P := [\![\pi]\!]Q$.

Premiss: trivially true, conclusion: desired implication.

Soundness of Frame Rule

$$\frac{P \to [\![\pi]\!]Q}{P * R \to [\![\pi]\!](Q * R)}$$

or equivalently

$$\models ([\![\pi]\!]Q) * R \ \to \ [\![\pi]\!](Q * R)$$

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

$\Longrightarrow$

Instantiate left rule with $P := [\![\pi]\!]Q$.

Premiss: trivially true, conclusion: desired implication.

$\Longleftarrow$

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: $\beta, h_1 \models [\![\pi]\!]Q$ and $\beta, h \models ([\![\pi]\!]Q) * R$

Right rule gives: $\beta, h \models [\![\pi]\!](Q * R)$

Soundness of Frame Rule

$$\frac{P \to [\![\pi]\!]Q}{P * R \to [\![\pi]\!](Q * R)}$$

or equivalently

$$\models ([\![\pi]\!]Q) * R \ \to \ [\![\pi]\!](Q * R)$$

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

$\Longrightarrow$

Instantiate left rule with $P := [\![\pi]\!]Q$.

Premiss: trivially true, conclusion: desired implication.

$\Longleftarrow$

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: $\beta, h_1 \models [\![\pi]\!]Q$ and $\beta, h \models ([\![\pi]\!]Q) * R$

Right rule gives: $\beta, h \models [\![\pi]\!](Q * R)$

$\square$

Lemma

Let $h_1, h'_1, h_2, h'_2 : \mathbb{N} \nrightarrow \mathbb{N}$ be heaplets, $\operatorname{dom} h_1 \cap \operatorname{dom} h_2 = \emptyset$
$\beta, \beta' : \mathit{Var} \to \mathbb{N}$ be variable assignments. Then:

$$\beta, h_1 \xrightarrow{\pi} \beta', h'_1 \quad \Longrightarrow \quad \left(\beta, h_1 \uplus h_2 \xrightarrow{\pi} \beta', h'_1 \uplus h'_2 \iff h_2 = h'_2\right)$$

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

Lemma

Let $h_1, h'_1, h_2, h'_2 : \mathbb{N} \nrightarrow \mathbb{N}$ be heaplets, $\mathrm{dom}\, h_1 \cap \mathrm{dom}\, h_2 = \emptyset$
$\beta, \beta' : \mathit{Var} \to \mathbb{N}$ be variable assignments. Then:

$$\beta, h_1 \xrightarrow{\pi} \beta', h'_1 \quad \Longrightarrow \quad \left(\beta, h_1 \uplus h_2 \xrightarrow{\pi} \beta', h'_1 \uplus h'_2 \iff h_2 = h'_2\right)$$

By structural induction:

- variable assignment $v := t$ (heap irrelevant)
- heap store $[t_1] := t_2$ $(\mathit{val}(t) \overset{!}{\in} \mathrm{dom}\, h_1)$
- heap load $v := [t]$ $(\mathit{val}(t) \overset{!}{\in} \mathrm{dom}\, h_1)$
- first-order test $\mathbf{?}\varphi$ (heap irrelevant)
- $\pi_1 \cup \pi_2$, $\pi_1 \,;\, \pi_2$, π^* (appeal to ind. hyp)

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

Soundness of Frame Rule

$$\models (\llbracket \pi \rrbracket Q) * R \;\rightarrow\; \llbracket \pi \rrbracket (Q * R) \qquad \text{if } \mathit{Modifies}(\pi) \cap \mathit{Free}(R) = \emptyset \; (\star)$$

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

$$\begin{tikzcd}
 & \beta, h_1 \models (\llbracket \pi \rrbracket Q) \arrow[r, "\pi"'] & \beta', h_{1Q} \models Q \arrow[dr, "\uplus"'] & \\
\beta, h \models (\llbracket \pi \rrbracket Q) * R \arrow[ur, "\uplus"'] \arrow[dr, "\uplus"] & & & \beta', h_{1Q} \uplus h_2 \models Q * R \\
 & \beta, h_2 \models R \arrow[r, "(\star)"] & \beta', h_2 \models R \arrow[ur, "\uplus"] &
\end{tikzcd}$$

Lemma, lang. is deterministic $\Longrightarrow \beta, h \models \llbracket \pi \rrbracket (Q * R)$

Memory Allocation and Deallocation

Syntax: Two statements

$\mathsf{var} := \mathtt{cons}(\mathsf{term}, \ldots, \mathsf{term})$ and $\mathtt{dispose}(\mathsf{var})$

Memory Allocation and Deallocation

Syntax: Two statements

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

Semantics: ρ and *fail*

$$\big((\beta, h), (\beta', h')\big) \in \rho(v := cons(t))$$
$$\text{iff}$$
$$\beta' = \beta[v/loc] \text{ and } h' = h \uplus \{(loc, val_\beta(t))\} \text{ and } loc \notin \operatorname{dom} h$$

$$fail(v := cons(t)) = \emptyset$$

`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)

Memory Allocation and Deallocation

Syntax: Two statements

var := $\mathtt{cons}$(term, ..., term) and $\mathtt{dispose}$(var)

Semantics: ρ and *fail*

$$((\beta, h), (\beta', h')) \in \rho(\mathtt{dispose}(v))$$
$$\text{iff}$$
$$\beta' = \beta \text{ and } \beta(v) \in \operatorname{dom} h \text{ and } h' = h \setminus \{(\beta(v), h(\beta(v))\}$$

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

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

Soundness of Frame Rule

$$\frac{([\![\pi]\!]Q) * R}{[\![\pi]\!](Q * R)} \text{ if } \mathit{Modifies}(\pi) \cap \mathit{Free}(R) = \emptyset$$

Proof by structural induction over π.

see Reynolds p.77ff

Decidability of Separation Logic

Karlsruhe Institute of Technology

Decidable

Some restricted logics from Separation Logic are decidable.

1. Restricted arithmetic
2. No magic wand $-\!\!*$

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

The separating implication $-\!\!*$ makes undecidable.

Relatively complete

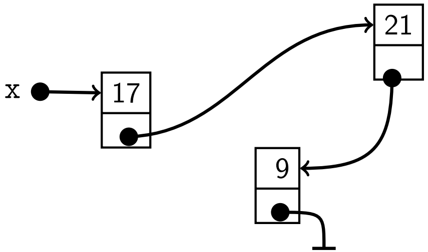
The calculus for Separation Logic is relatively complete.
Every correct program can be proved using an oracle for $\mathbb{N}$.

Application of Separation Logic

Abstraction Predicates

Use predicate symbols to abstract away from data structures

Example: Lists

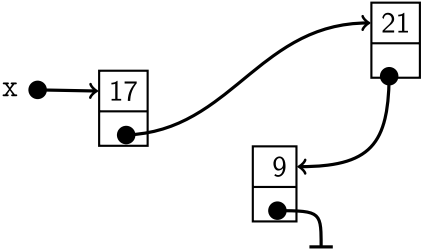


Abstraction Predicates

Use predicate symbols to abstract away from data structures

Example: Lists

$$\mathit{list}(x, \langle 17, 21, 9 \rangle) \quad \leftrightarrow \quad (x \mapsto 17) * (x{+}1 \mapsto v) * (v \mapsto 21) * \ldots$$
$$\ldots * (v + 1 \mapsto w) * (w \mapsto 9) * (w + 1 \mapsto 0)$$



Abstraction Predicates

Use predicate symbols to abstract away from data structures

Example: Lists

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

General:

Recursive predicate *list*:

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

Program Verification Using Separation Logic

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

Program Verification Using Separation Logic

- 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/`

Program Verification Using Separation Logic

- 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)

Program Verification Using Separation Logic

- 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)
- Viper (P. Müller, ETH Zurich)
  concurrency

Program Verification Using Separation Logic

- 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)
- Viper (P. Müller, ETH Zurich)
  concurrency
- SpaceInvader, YNot, HOLFoot, . . . , . . .

Discussion

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. $-\!\!*$)