Formale Systeme II: Theorie
Separation Logic

SS 2016

Prof. Dr. Bernhard Beckert · Dr. Mattias Ulbrich
Motivation
Given: a program with a contract:

1. precondition, FOL formula \( pre \)
2. postcondition, FOL formula \( post \)
3. code, while program \( \pi \)

In program verification, one formally proves that

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

If \( pre \) holds before execution of \( \pi \) then \( post \) holds after termination.

Reminder: weakest precondition calculus for DL.
The Framing Problem

Formal Software Verification

- Prove what effects a program has.

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

Formal Software Verification

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

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

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 still 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.
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);
    void 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

```java
interface Account {
    void setBalance(int);
    void 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

```java
interface Account {
    void setBalance(int);
    void 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

```java
interface Account {
    void setBalance(int);
    void 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:

- `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

Example in Java

```java
interface Account {
    void setBalance(int);
    void 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

- `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

## Example in Java

```java
interface Account {
    void setBalance(int);
    void 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:

- `setBalance` does not effect other accounts
- `setBalance` does not effect other customer objects
The Framing Problem

Example in Java

```java
interface Account {
    void setBalance(int);
    void 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

- `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 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 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)
- ...
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 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

account1
Heaps and “Footprints”

Heap

account1

account2
Heaps and “Footprints”

Heap

account1

account2

NO!
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:

\[
heap : \mathbb{N} \to \mathbb{N}
\]

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

\[
heaplet : \mathbb{N} \mapsto \mathbb{N}
\]

Partial function:

Partial function \( f : A \mapsto B \) is a function \( f : D \to B \) for \( D \subseteq A \). The finite set \( D = \text{dom } f \) is called the domain of \( f \).
Operations and Heaps

Disjoint union of heaplets:

\[ h = h_1 \cup h_2 \text{ iff } \text{dom}\ h_1 \cap \text{dom}\ h_2 = \emptyset \text{ and } h = h_1 \cup h_2. \]

\( h_1 \cup h_2 \) 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 +, −, ⋅, 0, 1

new in Separation Logic
Separation Logic – Syntax

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

**Formulae $\varphi$:**

new in Separation Logic
Separation Logic – Syntax

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

Formulae $\varphi$:
- $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \rightarrow \varphi_2$

new in Separation Logic
Separation Logic – Syntax

Terms \( t \):
- FOL terms over \( \mathbb{N} \) with \( +, –, \cdot, 0, 1 \)

Formulae \( \varphi \):
- \( \varphi_1 \land \varphi_2, \varphi_1 \lor \varphi_2, \varphi_1 \rightarrow \varphi_2 \)
- \( t_1 = t_2, t_1 < t_2, \ldots \)
Separation Logic – Syntax

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

Formulae $\varphi$:
- $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ... 
- $\forall x. \varphi$, $\exists x. \varphi$

new in Separation Logic
Separation Logic – Syntax

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

Formulae $\varphi$:
- $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x. \varphi$, $\exists x. \varphi$
- $\varphi_1 \ast \varphi_2$

new in Separation Logic
Separation Logic – Syntax

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

Formulae $\varphi$:
- $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x. \varphi$, $\exists x. \varphi$
- $\varphi_1 \ast \varphi_2$
- $\text{emp}$

new in Separation Logic
Separation Logic – Syntax

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

Formulae $\varphi$:
- $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, \ldots
- $\forall x. \varphi$, $\exists x. \varphi$
- $\varphi_1 \ast \varphi_2$
- 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 $\varphi$:
- $\varphi_1 \land \varphi_2$, $\varphi_1 \lor \varphi_2$, $\varphi_1 \rightarrow \varphi_2$
- $t_1 = t_2$, $t_1 < t_2$, ...
- $\forall x. \varphi$, $\exists x. \varphi$
- $\varphi_1 \ast \varphi_2$
- $\text{emp}$
- $t_1 \mapsto t_2$
- $\varphi_1 \ast \varphi_2$ (later)

new in Separation Logic
Operator Precedence

How are the implicit parentheses in

\[ B \to C \land D \lor A \times x \mapsto y \]
How are the implicit parentheses in

\[ B \rightarrow C \land D \lor A \ast x \mapsto y \] ?

**Binding force:**

* binds like \( \land \)

\(-\ast\) binds like \( \rightarrow, \lor \)

\( \mapsto \) binds like \( = \)
Operator Precedence

How are the implicit parentheses in
\[ B \rightarrow (C \land D) \lor (A \times (x \mapsto y)) \]

**Binding force:**

- \(*\) binds like \(\land\)
- \(\rightarrow\) binds like \(\rightarrow, \lor\)
- \(\mapsto\) binds like \(=\)

**Answer:**

\[
(B \rightarrow (C \land D)) \lor (A \times (x \mapsto y))
\]

or

\[
B \rightarrow ((C \land D) \lor (A \times (x \mapsto y)))
\]
Operator Precedence

How are the implicit parentheses in
\[ B \rightarrow^* C \land D \lor A \ast x \mapsto^\downarrow y \]?

**Binding force:**

- \( \ast \) binds like \( \land \)
- \( \rightarrow^* \) binds like \( \rightarrow, \lor \)
- \( \mapsto^\downarrow \) binds like \( = \)

**Answer:**

\[
\Big( B \rightarrow^* (C \land D) \Big) \lor \Big( A \ast (x \mapsto^\downarrow y) \Big)
\]

or

\[
B \rightarrow^* \Big( (C \land D) \lor (A \ast (x \mapsto^\downarrow y)) \Big)
\]

Add explicit parentheses when combining \( \lor/ \rightarrow/ \rightarrow^* \) or \( \land/ \ast \)
Separation Logic – Semantics

Structure

Fixed first order domain: \( \mathbb{N} \).
Terms and formulas are evaluated over:

1. Variable assignment \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \rightarrow \mathbb{N} \)
Separation Logic – Semantics

Structure

Fixed first order domain: \( \mathbb{N} \).
Terms and formulas are evaluated over:

1. Variable assignment \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \rightarrow \mathbb{N} \)

Terms:
- \( \text{val}_\beta(t_1 + t_2) = \text{val}_\beta(t_1) +_{\mathbb{N}} \text{val}_\beta(t_2) \), same for “·”
- \( \text{val}_\beta(x) = \beta(x) \) for variable \( x \)

Formulas in FOL:
- Operator \( \beta, h \models \) is homomorphic for \( \land, \lor, \rightarrow, \forall, \exists, <, = \).
- Example: \( \beta, h \models \varphi_1 \land \varphi_2 \) 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 \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \rightarrow \mathbb{N} \)

- \( \beta, h \models \text{emp} \) iff \( \text{dom } h = \emptyset \)
**Separation Logic – Semantics**

**Structure**

Fixed first order domain: \( \mathbb{N} \).

Terms and formulas are evaluated over:

1. Variable assignment \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \rightarrow \mathbb{N} \)

- \( \beta, h \models \text{emp} \) iff \( \text{dom } h = \emptyset \)
- \( \beta, h \models t_1 \mapsto t_2 \) iff \( h = \{(\text{val}_\beta(t_1), \text{val}_\beta(t_2))\} \)
Separation Logic – Semantics

Structure

Fixed first order domain: \( \mathbb{N} \).
Terms and formulas are evaluated over:

1. Variable assignment \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \mapsto \mathbb{N} \)

- \( \beta, h \models \text{emp} \) iff \( \text{dom} \ h = \emptyset \)
- \( \beta, h \models t_1 \mapsto t_2 \) iff \( h = \{(\text{val}_\beta(t_1), \text{val}_\beta(t_2))\} \)
- \( \beta, h \models \varphi_1 \ast \varphi_2 \) iff there exist heaplets \( h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N} \) with
Separation Logic – Semantics

Structure

Fixed first order domain: \( \mathbb{N} \).

Terms and formulas are evaluated over:

1. Variable assignment \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \rightarrow \mathbb{N} \)

- \( \beta, h \models \text{emp} \)  if \( \text{dom} \ h = \emptyset \)
- \( \beta, h \models t_1 \mapsto t_2 \)  if \( h = \{(\text{val}_\beta(t_1), \text{val}_\beta(t_2))\} \)
- \( \beta, h \models \varphi_1 \ast \varphi_2 \)  if there exist heaplets \( h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N} \) 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 \( \beta : \text{Var} \to \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \to \mathbb{N} \)

- \( \beta, h \models \text{emp} \) iff \( \text{dom } h = \emptyset \)

- \( \beta, h \models t_1 \mapsto t_2 \) iff \( h = \{ (\text{val}_\beta(t_1), \text{val}_\beta(t_2)) \} \)

- \( \beta, h \models \varphi_1 \ast \varphi_2 \) iff there exist heaplets \( h_1, h_2 : \mathbb{N} \to \mathbb{N} \) with
  1. \( h = h_1 \cup h_2 \) 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 \( \beta : \text{Var} \rightarrow \mathbb{N} \)
2. Heaplet \( h : \mathbb{N} \rightarrow \mathbb{N} \)

\[ \begin{align*}
\beta, h \models \text{emp} & \iff \text{dom } h = \emptyset \\
\beta, h \models t_1 \mapsto t_2 & \iff h = \{ (\text{val}_\beta(t_1), \text{val}_\beta(t_2)) \} \\
\beta, h \models \varphi_1 \ast \varphi_2 & \iff \text{there exist heaplets } h_1, h_2 : \mathbb{N} \rightarrow \mathbb{N} \text{ with } \\
& \quad 1. h = h_1 \cup h_2 \text{ and } \\
& \quad 2. \beta, h_1 \models \varphi_1 \text{ and } \\
& \quad 3. \beta, h_2 \models \varphi_2
\end{align*} \]
Connector $\ast$ is called **Separating Conjunction**

$A \ast B$ has the following intuitive semantics:

$A \ast B$ 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

| $\models A \iff A \land A$ | (idempotence for $\land$) | NO! Counterexample: $\not\models 7 \mapsto 3 \land 7 \mapsto 3 \ast 6 \mapsto 4 \rightarrow 7 \mapsto 3$ |
### Properties of Separation Logic

#### Idempotence

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\models A \iff A \land A)</td>
<td>(idempotence for (\land))</td>
</tr>
<tr>
<td></td>
<td>(\not\models A \iff A \ast A)</td>
<td>(idempotence also for (\ast))</td>
</tr>
</tbody>
</table>

---

**NO! Counterexample:**

\[\not\models (7 \mapsto \{\{3\}\} \ast (7 \mapsto \{\{3\}\})\]
## Properties of Separation Logic

### Idempotence

- $\models A \iff A \land A$  
  (idempotence for $\land$)
- $\mathrel{?} A \iff A \ast A$  
  (idempotence also for $\ast$)

### NO! Counterexample:

\[
\neg(7 \mapsto 3 \ast 6 \mapsto 4 \mapsto 7 \mapsto 3)
\]
## Properties of Separation Logic

<table>
<thead>
<tr>
<th>Idempotence</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\models A \Leftrightarrow A \land A$</td>
<td>(idempotence for $\land$)</td>
</tr>
<tr>
<td>$\not\models A \Leftrightarrow A \ast A$</td>
<td>(idempotence also for $\ast$)</td>
</tr>
<tr>
<td><strong>NO! Counterexample:</strong></td>
<td></td>
</tr>
<tr>
<td>$\models \neg(7 \leftrightarrow 3 \rightarrow 7 \leftrightarrow 3 \ast 7 \leftrightarrow 3)$</td>
<td></td>
</tr>
</tbody>
</table>
Properties of Separation Logic

Idempotence

- $\models A \leftrightarrow A \land A$ (idempotence for $\land$)
- $\not\models A \leftrightarrow A \ast A$ (idempotence also for $\ast$?)
- NO! Counterexample:
  $\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 \ast 7 \mapsto 3)$

Weakening

- $\models A \land B \rightarrow A$ (Weakening of conjunction)
- $\not\models A \ast B \rightarrow A$ (Weakening of separating conjunction?)
- NO! Counterexample:
  $\models \neg(7 \mapsto 3 \rightarrow 6 \mapsto 4 \rightarrow 7 \mapsto 3)$
## Properties of Separation Logic

### Idempotence

| ▪ | $\models A \iff A \land A$ | (idempotence for $\land$) |
| ▪ | $\not\models A \iff A \ast A$ | (idempotence also for $\ast$?) |
| ▪ | **NO!** Counterexample: |
| ▪ | $\not\models \neg(7 \mapsto 3 \rightarrow 7 \mapsto 3 \ast 7 \mapsto 3)$ |

### Weakening

| ▪ | $\models A \land B \rightarrow A$ | (Weakening of conjunction) |
Properties of Separation Logic

Idempotence

- $\models A \leftrightarrow A \land A$  
  (idempotence for $\land$)
- $\not\models A \leftrightarrow A \ast A$  
  (idempotence also for $\ast$?)
- **NO!** Counterexample:
  $\models \neg (7 \mapsto 3 \rightarrow 7 \mapsto 3 \ast 7 \mapsto 3)$

Weakening

- $\models A \land B \rightarrow A$  
  (Weakening of conjunction)
- $\not\models A \ast B \rightarrow A$  
  (Weakening of separating conjunction?)
Properties of Separation Logic

### Idempotence

- \( \models A \leftrightarrow A \land A \)  
  (idempotence for \( \land \))
- \( \not\models A \leftrightarrow A \ast A \)  
  (idempotence also for \( \ast \))
- **NO!** Counterexample:
  \[ \models \neg(7 \leftrightarrow 3 \rightarrow 7 \leftrightarrow 3 \ast 7 \rightarrow 3) \]

### Weakening

- \( \models A \land B \rightarrow A \)  
  (Weakening of conjunction)
- \( \not\models A \ast B \rightarrow A \)  
  (Weakening of separating conjunction?)
- **NO!** Counterexample:
Properties of Separation Logic

### Idempotence

- \[\models A \iff A \land A\]  
  (idempotence for \(\land\))

- \[? A \iff A \ast A\]  
  (idempotence also for \(\ast\)?)

- **NO!** Counterexample:
  \[\models \neg (7 \leftrightarrow 3 \rightarrow 7 \leftrightarrow 3 \ast 7 \rightarrow 3)\]

### Weakening

- \[\models A \land B \rightarrow A\]  
  (Weakening of conjunction)

- \[? A \ast B \rightarrow A\]  
  (Weakening of separating conjunction?)

- **NO!** Counterexample:
  \[\models \neg (7 \leftrightarrow 3 \ast 6 \leftrightarrow 4 \rightarrow 7 \leftrightarrow 3)\]
\( \beta, h \models A \leftrightarrow B \) means that:

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

\( \beta, h \models A \leftrightarrow B \) means that:

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

On the other hand:

\( \beta, h \models ? \iff (\text{val}(A), \text{val}(B)) \in h \)
Caution

\( \beta, h \models A \leftrightarrow B \) means that:

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

On the other hand:

\( \beta, h \models A \leftrightarrow B \ast \text{true} \iff (\text{val}(A), \text{val}(B)) \in h \)
Caution

\( \beta, h \models A \leftrightarrow B \) means that:

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

On the other hand:

\[ \beta, h \models A \leftrightarrow B \ast \text{true} \iff (\text{val}(A), \text{val}(B)) \in h \]

Notation sometimes: \( A \hookrightarrow B \iff A \leftrightarrow B \ast \text{true} \)
Some Valid Formulas

- \( \text{emp} \iff \neg (\exists x, y. x \mapsto y \mapsto true) \)
Some Valid Formulas

- \( \text{emp} \iff \neg(\exists x, y. x \to y \ast \text{true}) \)

- \( \varphi \ast \psi \iff \varphi \land \psi \)
  
  if neither \( \text{emp} \) nor \( \to \) occur.
Some Valid Formulas

- \(\text{emp} \iff \neg (\exists x, y. x \mapsto y \ast \text{true})\)

- \(\varphi \ast \psi \iff \varphi \land \psi\)
  
  if neither \text{emp} nor \mapsto occur.

- \(x \mapsto y \land x \mapsto z \rightarrow y = z\)
Some Valid Formulas

\begin{itemize}
  \item \textbf{emp} \iff \neg(\exists x, y. x \mapsto y \neq true)
  
  \item \varphi \neq \psi \iff \varphi \land \psi
    
    if neither \textbf{emp} nor \mapsto occur.
  
  \item \(x \mapsto y \land x \mapsto z \rightarrow y = z\)
  
  \item \(P \neq (Q \lor R) \iff (P \neq Q) \lor (P \neq R)\)
\end{itemize}
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \ast x \mapsto z \)
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

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

Beckert, Ulbrich – Formale Systeme II: Theorie
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

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

2. $x \mapsto y \land x \mapsto z$
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  
   - **UNSAT**

2. $x \mapsto y \land x \mapsto z$  
   - **SAT**; true if $y = z$
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \leftrightarrow y \ast x \leftrightarrow z$  \hspace{1cm} **UNSAT**
2. $x \leftrightarrow y \land x \leftrightarrow z$  \hspace{1cm} **SAT**; true if $y = z$
3. $(x \leftrightarrow 0 \land y \leftrightarrow 0) \rightarrow x = y$
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \ast x \mapsto z \)  \hspace{1cm} \text{UNSAT}
2. \( x \mapsto y \land x \mapsto z \)  \hspace{1cm} \text{SAT}; \text{true if } y = z
3. \( (x \mapsto 0 \land y \mapsto 0) \rightarrow x = y \)  \hspace{1cm} \text{VALID}
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$ \hspace{1cm} UNSAT
2. $x \mapsto y \land x \mapsto z$ \hspace{1cm} SAT; true if $y = z$
3. $(x \mapsto 0 \land y \mapsto 0) \rightarrow x = y$ \hspace{1cm} VALID
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \ast x \mapsto z \)  
   - **UNSAT**

2. \( x \mapsto y \land x \mapsto z \)  
   - **SAT**; true if \( y = z \)

3. \((x \mapsto 0 \land y \mapsto 0) \rightarrow x = y\)  
   - **VALID**

4. \((x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y\)  
   - **UNSAT**
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \ast x \mapsto z \)  
   - UNSAT

2. \( x \mapsto y \land x \mapsto z \)  
   - SAT; true if \( y = z \)

3. \( (x \mapsto 0 \land y \mapsto 0) \rightarrow x = y \)  
   - VALID

4. \( (x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y \)  
   - UNSAT

5. \( (x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y) \)
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \ast x \mapsto z \)  \( \text{UNSAT} \)
2. \( x \mapsto y \wedge x \mapsto z \)  \( \text{SAT} \); true if \( y = z \)
3. \( (x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y \)  \( \text{VALID} \)
4. \( (x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y \)  \( \text{UNSAT} \)
5. \( (x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y) \)  \( \text{VALID} \)
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  \hspace{2cm} \text{UNSAT}
2. $x \mapsto y \land x \mapsto z$  \hspace{2cm} \text{SAT}; \text{ true if } y = z
3. $(x \mapsto 0 \land y \mapsto 0) \rightarrow x = y$  \hspace{2cm} \text{VALID}
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$  \hspace{2cm} \text{UNSAT}
5. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y)$  \hspace{2cm} \text{VALID}
6. $(x \mapsto a \land y \mapsto b) \rightarrow a = b$
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  \hspace{1cm} \text{UNSAT}
2. $x \mapsto y \wedge x \mapsto z$  \hspace{1cm} \text{SAT}; true if $y = z$
3. $(x \mapsto 0 \wedge y \mapsto 0) \rightarrow x = y$  \hspace{1cm} \text{VALID}
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$  \hspace{1cm} \text{UNSAT}
5. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y)$  \hspace{1cm} \text{VALID}
6. $(x \mapsto a \wedge y \mapsto b) \rightarrow a = b$  \hspace{1cm} \text{VALID}
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  \hspace{1cm} UNSAT
2. $x \mapsto y \land x \mapsto z$  \hspace{1cm} SAT; true if $y = z$
3. $(x \mapsto 0 \land y \mapsto 0) \rightarrow x = y$  \hspace{1cm} VALID
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$  \hspace{1cm} UNSAT
5. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg (x = y)$  \hspace{1cm} VALID
6. $(x \mapsto a \land y \mapsto b) \rightarrow a = b$  \hspace{1cm} VALID
7. $\varphi \ast \text{emp} \rightarrow \varphi$
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \cdot x \mapsto z \) \hspace{5cm} \text{UNSAT}
2. \( x \mapsto y \land x \mapsto z \) \hspace{5cm} \text{SAT}; \text{true if } y = z
3. \((x \mapsto 0 \land y \mapsto 0) \rightarrow x = y\) \hspace{5cm} \text{VALID}
4. \((x \mapsto 0 \cdot y \mapsto 0) \rightarrow x = y\) \hspace{5cm} \text{UNSAT}
5. \((x \mapsto 0 \cdot y \mapsto 0) \rightarrow \neg(x = y)\) \hspace{5cm} \text{VALID}
6. \((x \mapsto a \land y \mapsto b) \rightarrow a = b\) \hspace{5cm} \text{VALID}
7. \(\varphi \cdot \text{emp} \rightarrow \varphi\) \hspace{5cm} \text{VALID}
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \ast x \mapsto z \)  
   - UNSAT
2. \( x \mapsto y \land x \mapsto z \)  
   - SAT; true if \( y = z \)
3. \( (x \mapsto 0 \land y \mapsto 0) \rightarrow x = y \)  
   - VALID
4. \( (x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y \)  
   - UNSAT
5. \( (x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y) \)  
   - VALID
6. \( (x \mapsto a \land y \mapsto b) \rightarrow a = b \)  
   - VALID
7. \( \varphi \ast \text{emp} \rightarrow \varphi \)  
   - VALID
8. \( \varphi \ast \neg \varphi \)
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  \hspace{1cm} UNSAT
2. $x \mapsto y \land x \mapsto z$  \hspace{1cm} SAT; true if $y = z$
3. $(x \mapsto 0 \land y \mapsto 0) \rightarrow x = y$  \hspace{1cm} VALID
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$  \hspace{1cm} UNSAT
5. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y)$  \hspace{1cm} VALID
6. $(x \mapsto a \land y \mapsto b) \rightarrow a = b$  \hspace{1cm} VALID
7. $\varphi \ast \text{emp} \rightarrow \varphi$  \hspace{1cm} VALID
8. $\varphi \ast \neg\varphi$

a. $\psi \ast \neg\psi$  \hspace{1cm} for $\psi$ without $\mapsto$, emp
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  \hspace{1cm} \text{UNSAT}
2. $x \mapsto y \land x \mapsto z$  \hspace{1cm} \text{SAT}; \text{true if } y = z
3. $(x \mapsto 0 \land y \mapsto 0) \rightarrow x = y$  \hspace{1cm} \text{VALID}
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$  \hspace{1cm} \text{UNSAT}
5. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y)$  \hspace{1cm} \text{VALID}
6. $(x \mapsto a \land y \mapsto b) \rightarrow a = b$  \hspace{1cm} \text{VALID}
7. $\varphi \ast \text{emp} \rightarrow \varphi$  \hspace{1cm} \text{VALID}
8. $\varphi \ast \neg \varphi$  \hspace{1cm} \text{UNSAT} for $\psi$ without $\mapsto$, emp

a. $\psi \ast \neg \psi$  \hspace{1cm} \text{UNSAT} for $\psi$ without $\mapsto$, emp
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. \( x \mapsto y \neq x \mapsto z \)  
   **UNSAT**

2. \( x \mapsto y \land x \mapsto z \)  
   **SAT**; true if \( y = z \)

3. \((x \mapsto 0 \land y \mapsto 0) \rightarrow x = y\)  
   **VALID**

4. \((x \mapsto 0 \neq y \mapsto 0) \rightarrow x = y \)  
   **UNSAT**

5. \((x \mapsto 0 \neq y \mapsto 0) \rightarrow \neg(x = y)\)  
   **VALID**

6. \((x \mapsto a \land y \mapsto b) \rightarrow a = b\)  
   **VALID**

7. \( \varphi \neq \text{emp} \rightarrow \varphi \)  
   **VALID**

8. \( \varphi \neq \neg \varphi \)
   a. \( \psi \neq \neg \psi \)  
   **UNSAT** for \( \psi \) without \( \mapsto \), \text{emp}
   b. \( x \mapsto y \neq (x \mapsto y) \)
Quiz!

Are the following formulas valid/satisfiable/unsatisfiable?

1. $x \mapsto y \ast x \mapsto z$  \hspace{1cm} \text{UNSAT}
2. $x \mapsto y \land x \mapsto z$  \hspace{1cm} \text{SAT}; true if $y = z$
3. $(x \mapsto 0 \land y \mapsto 0) \rightarrow x = y$  \hspace{1cm} \text{VALID}
4. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow x = y$  \hspace{1cm} \text{UNSAT}
5. $(x \mapsto 0 \ast y \mapsto 0) \rightarrow \neg(x = y)$  \hspace{1cm} \text{VALID}
6. $(x \mapsto a \land y \mapsto b) \rightarrow a = b$  \hspace{1cm} \text{VALID}
7. $\varphi \ast \text{emp} \rightarrow \varphi$  \hspace{1cm} \text{VALID}
8. $\varphi \ast \neg \varphi$
   a. $\psi \ast \neg \psi$  \hspace{1cm} \text{UNSAT} for $\psi$ without $\mapsto$, $\text{emp}$
   b. $x \mapsto y \ast \neg(x \mapsto y)$  \hspace{1cm} \text{SAT}, equivalent to $x \mapsto y \ast \text{true}$
The Magic Wand

Modus Ponens for classical logic

\[
A \land (A \rightarrow B) \\
\hline
B
\]

Corresponding rule for separating conjunction

Modus Ponens for separation logic

\[
A \ast (A \rightarrow^* B) \\
\hline
B
\]

The magic wand operator \(A \rightarrow^* B\), aka separating implication

\[
\beta, h | = A \rightarrow^* B \\
\iff \\
\text{for all } h', h + : N \mapsto \rightarrow N: \text{If } h + = h \cup h' \text{ and } h' | = A, \text{ then } h' + | = B
\]
The Magic Wand

Modus Ponens for classical logic

\[
A \land (A \rightarrow B) \quad \Rightarrow \quad B
\]

Corresponding rule for separating conjunction \(\ast\)?
The Magic Wand

Modus Ponens for classical logic

\[ A \land (A \rightarrow B) \quad \Rightarrow \quad B \]

Corresponding rule for separating conjunction \( \ast \)?

Modus Ponens for separation logic

\[ A \ast (A \rightarrow \ast B) \quad \Rightarrow \quad B \]

The **magic wand operator** \( A \rightarrow \ast B \), aka **separating implication**:

\[ \beta, h \models A \rightarrow \ast B \]

\[ \iff \]

for all \( h', h^+ : \mathbb{N} \to \mathbb{N} : \) If \( h^+ = h \cup h' \) and \( h' \models A \), then \( h^+ \models B \)
Separating Operators

\[ \bullet \models_{\text{SL}} f \star g \quad \text{when there are } \bullet \text{ and } \mathbb{D} \text{ such that } \bullet = \mathbb{D}, \text{ as well as } \bullet \models_{\text{SL}} f \text{ and } \mathbb{D} \models g. \]

\[ \mathbb{D} \models_{\text{SL}} f \rightarrow g \quad \text{when any } \bullet \text{ such that } \bullet \models_{\text{SL}} f \text{ is also such that } \bullet \models g. \]

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
Programming Language

\[
\text{statement} ::= \text{while formula do statement} \\
  \quad \text{if formula then statement else statement} \\
  \quad \text{statement ; statement} \\
  \quad \text{var := term} \\
  \quad [\text{term}] := \text{term} \\
  \quad \text{var := [term]} \\
  \quad \text{var := cons(term, ..., term)} \\
  \quad \text{dispose(var)}
\]
Kripke Frames with Heaps

- Every state is a pair \((\beta, h)\) 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 \(st\).
**Program semantics (repetition from FODL)**

**Accessibility Relation for Programs**

\[ \rho : \text{statement} \rightarrow S \times S \]
Accessiblity Relation for Programs

\[ \rho : \text{statement} \rightarrow S \times S \]

\[ \rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2) \]
Program semantics (repetition from FODL)

Accessibility Relation for Programs
\( \rho : \text{statement} \rightarrow S \times S \)

\[ \rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2) \]

\[ \rho(\pi_1 ; \pi_2) = \rho(\pi_1) ; \rho(\pi_2) \quad ; \text{is forward composition} \]
Program semantics (repetition from FODL)

Accessiblity Relation for Programs

\[ \rho : \text{statement} \rightarrow S \times S \]

\[ \rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2) \]

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

\[ = \{(s, t) \mid \exists u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \]
Accessibility Relation for Programs

\( \rho : \text{statement} \rightarrow S \times S \)

\[
\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)
\]

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

\[
= \{(s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\}
\]

\[
\rho(\pi^*) = \rho(\pi)^* \quad \text{* is refl. transitive closure}
\]
Program semantics (repetition from FODL)

Accessibility Relation for Programs

\[ \rho: \text{statement} \rightarrow S \times S \]

\[ \rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2) \]

\[ \rho(\pi_1 ; \pi_2) = \rho(\pi_1) ; \rho(\pi_2) \quad \text{; is forward composition} \]
\[ = \{(s, t) | \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2)\} \]

\[ \rho(\pi^*) = \rho(\pi)^* \quad \text{* is refl. transitive closure} \]
\[ = \{(s_0, s_n) | \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\} \]
**Accessiblity Relation for Programs**

\[ \rho : \text{statement} \rightarrow S \times S \]

\[
\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)
\]

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

\[
\quad = \{ (s, t) \mid \text{ex. } u \in S \text{ with } (s, u) \in \rho(\pi_1), (u, t) \in \rho(\pi_2) \}
\]

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

\[
\quad = \{ (s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n \}
\]

\[
\rho(\Diamond \varphi) = \{ (s, s) \mid s \models \varphi \}
\]
Program semantics (repetition from FODL)

Accessibility Relation for Programs

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

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

$\rho(\pi_1 ; \pi_2) = \rho(\pi_1) ; \rho(\pi_2)$ ; is forward composition

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

$\rho(\pi^*) = \rho(\pi)^*$ * is refl. transitive closure

$= \{(s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\}$

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

Reminder: IF and WHILE

if $\varphi$ then $\alpha$ else $\beta = (?\varphi ; \alpha) \cup (?\neg\varphi ; \beta)$
while $\varphi$ do $\alpha = (?\varphi ; \alpha)^* ; ?\neg\varphi$
Program semantics (with heap)

Accessibility Relation for Programs

\[ \rho: \text{statement} \rightarrow S \times S \]

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

Accessibility Relation for Programs

\[ \rho : \text{statement} \rightarrow S \times S \]

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

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

Accessibility Relation for Programs

\( \rho : \text{statement} \to S \times S \)

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

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

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

Accessiblity Relation for Programs

\[ \rho : \text{statement} \rightarrow S \times S \]

A state \( s \in S \) is a pair \( (\beta, h) \) 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 \( x \neq a \))
Failing executions

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

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

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

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

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

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

Model unallowed heap access:

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

$s \in \text{fail(}\pi\text{)}$ means: $\pi$ started in $s$ may cause memory violation.
Failing executions

Model unallowed heap access:

\[ \text{fail : statement } \rightarrow S \]

\[ s \in \text{fail}(\pi) \text{ means: } \pi \text{ started in } s \text{ may cause memory violation} \]

\[
\begin{align*}
\text{fail}(x := t) &= \\
\text{fail}(?\psi) &= \emptyset
\end{align*}
\]
Failing executions

Model unallowed heap access:

\[ \text{fail} : \text{statement} \rightarrow S \]

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

\[
\begin{align*}
\text{fail}(x := t) &= \\
\text{fail}(?\psi) &= \emptyset \\
\text{fail}(x := [t]) &= \\
\text{fail}([t] := u) &= \{(\beta, h) \mid \text{val}_\beta(t) \notin \text{dom } h\}
\end{align*}
\]
Failing executions

Model unallowed heap access:

\[ \text{fail : statement} \rightarrow S \]

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

\[
\begin{align*}
\text{fail}(x := t) &= \\
\text{fail}(?\psi) &= \emptyset \\
\text{fail}(x := [t]) &= \\
\text{fail}([t] := u) &= \{(\beta, h) \mid \text{val}_\beta(t) \notin \text{dom } h\} \\
\text{fail}(\pi_1 ; \pi_2) &= \text{fail}(\pi_1) \cup (\rho(\pi_1) ; \text{fail}(\pi_2))
\end{align*}
\]
Failing executions

Model unallowed heap access:

\[ \text{fail} : \text{statement} \rightarrow S \]

\[ s \in \text{fail}(\pi) \text{ means: } \pi \text{ started in } s \text{ may cause memory violation} \]

\[
\begin{align*}
\text{fail}(x := t) & = \\
\text{fail}(?\psi) & = \emptyset \\
\text{fail}(x := [t]) & = \\
\text{fail}([t] := u) & = \{ (\beta, h) \mid \text{val}_\beta(t) \notin \text{dom } h \} \\
\text{fail}(\pi_1 ; \pi_2) & = \text{fail}(\pi_1) \cup (\rho(\pi_1) ; \text{fail}(\pi_2)) \\
\text{fail}(\pi^*) & = \rho(\pi^*) ; \text{fail}(\pi)
\end{align*}
\]
Failing executions

Model unallowed heap access:

\[
\text{fail : statement } \rightarrow S \\
s \in \text{fail}(\pi) \text{ means: } \pi \text{ started in } s \text{ may cause memory violation}
\]

\[
\begin{align*}
\text{fail}(x := t) &= \\
\text{fail}(?\psi) &= \emptyset \\
\text{fail}(x := [t]) &= \\
\text{fail}([t] := u) &= \{(\beta, h) | \text{val}_\beta(t) \not\in \text{dom } h\} \\
\text{fail}(\pi_1 ; \pi_2) &= \text{fail}(\pi_1) \cup (\rho(\pi_1) ; \text{fail}(\pi_2)) \\
\text{fail}(\pi^*) &= \rho(\pi^*) ; \text{fail}(\pi)
\end{align*}
\]

with \( A ; B = \{x | \text{ex } y \text{ with } (x, y) \in A \text{ and } y \in B\} \)
Fail-aware modality

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

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

New modality \([\cdot]\):
\[ s \models [\pi] \varphi \iff s' \models \varphi \text{ for all } (s, s') \in \rho(\pi) \text{ and } s \not\in \text{fail}(\pi). \]

Now:
\[ \text{emp} \rightarrow [[5 := 42]] \psi \] is not valid for any \( \psi \).
Valid formulas:

- \( x \mapsto 5 \rightarrow [v := [x]; [x] := v + 1]x \mapsto 6 \)
- \( (\exists y. x \mapsto y) \rightarrow [[x] := 7]x \mapsto 7 \)
- \( x \mapsto 5 \ast y \mapsto 6 \rightarrow [[x] := 7](x \mapsto 7 \ast y \mapsto 6) \)
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 $\pi$ 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 \( \pi \) started in a state that satisfies precondition \( P \) terminates in a state which satisfies postcondition \( Q \) (it it terminates).

Semantically the same as \( P \rightarrow \left[\pi\right]Q \).
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 \( \pi \) started in a state that satisfies precondition \( P \) terminates in a state which satisfies postcondition \( Q \) (it it terminates).

Semantically the same as \( P \rightarrow [\pi] Q \).

We present the calculus using dynamic logic notation.
Calculus Rules for Sep Logic

\[ x = m \rightarrow [x := E]x = E[x \leftarrow m] \]

\[ x = m \land E \leftrightarrow n \rightarrow [x := [E]](x = n \land E[x \leftarrow m] \leftrightarrow n) \]

\[ (\exists x. E \leftrightarrow x) \rightarrow [[E] := F] E \leftrightarrow F \]

\(x, m, n\) distinct variables; \(E, F\) terms of \(\mathbb{N}\).
\(E[x \leftarrow F]\) is substitution: replaces all free occurrences of \(x\) in \(E\) by \(F\).
Calculus Rules for Sep Logic

\[
\begin{align*}
P & \rightarrow [\pi_1]Q & Q & \rightarrow [\pi_2]R  \\
\hline
P & \rightarrow [\pi_1; \pi_2]R
\end{align*}
\]

\[
\begin{align*}
P & \land C \rightarrow [\pi_1]Q & P & \land \neg C \rightarrow [\pi_2]Q  \\
\hline
P & \rightarrow [\text{if } C \text{ then } \pi_1 \text{ else } \pi_2]Q
\end{align*}
\]

\[
\begin{align*}
P & \land C \rightarrow [\pi]P  \\
\hline
P & \rightarrow [\text{while } C \text{ do } \pi](P \land \neg C)
\end{align*}
\]

\[
\begin{align*}
P & \rightarrow [\pi]Q  \\
\hline
(\exists x. P) & \rightarrow [\pi](\exists x. Q) \quad \text{if } x \not\in \text{Free}(\pi)
\end{align*}
\]

(Normal rules of Hoare Calculus – nothing special for Sep Logic)
The Frame Rule

This is the key point about Separation Logic:

\[
P \rightarrow \llbracket \pi \rrbracket \ Q
\]

\[
P \ast R \rightarrow \llbracket \pi \rrbracket (Q \ast R)
\]

\[
\text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset
\]
The Frame Rule

\[ P \rightarrow [\pi] Q \]
\[ P \ast R \rightarrow [\pi](Q \ast R) \]

\( Modifies(\pi) \cap Free(R) = \emptyset \)

Separation in Proofs

Proof: \( P \rightarrow [\pi] Q \) using in \( P, Q \) the memory \( \pi \) refers to.

Get for free: Nothing besides these memory locations has changed.
Remember: The Framing Problem

Example in Java

```java
//@ requires account1 != account2;
//@ ensures \result == 100;
int f(Account acc1, Account acc2) {
    acc1.setBalance(100);
    acc2.setBalance(200);
    return acc1.getBalance();
}
```

Rule for setBalance:

\[ A \mapsto x \mapsto [\text{setBalance}(A, y)] A \mapsto y \]
Remember: The Framing Problem

Example in Java

```java
//@ requires account1 != account2;
//@ ensures \result == 100;
int f(Account acc1, Account acc2) {
    acc1.setBalance(100);
    acc2.setBalance(200);
    return acc1.getBalance();
}
```

Rule for setBalance:

\[ A \mapsto x \rightarrow \left[\text{setBalance}(A, y)\right] A \mapsto y \]

Use Frame Rule:

\[ acc2 \mapsto x \rightarrow \ldots \]

\[ \ldots \left[\text{setBalance}(acc2, 200); \right] acc2 \mapsto 200 \]
Remember: The Framing Problem

Example in Java

```java
//@ requires account1 != account2;
//@ ensures result == 100;
int f(Account acc1, Account acc2) {
    acc1.setBalance(100);
    acc2.setBalance(200);
    return acc1.getBalance();
}
```

Rule for setBalance:

\[ A \leftrightarrow x \rightarrow [setBalance(A, y)] A \leftrightarrow y \]

Use Frame Rule:

\[ acc2 \leftrightarrow x \ast acc1 \leftrightarrow 100 \rightarrow \ldots \]

\[ \ldots [setBalance(acc2, 200); \] acc2 \leftrightarrow 200 \ast acc1 \leftrightarrow 100 \]
On the board ...

\[(\exists v. X \mapsto v \ast Y \mapsto v) \rightarrow [X := [X] ; Y := [Y]] \quad X = Y\]
Soundness of Frame Rule

\[
P \rightarrow \llbracket \pi \rrbracket Q \\
P \ast R \rightarrow \llbracket \pi \rrbracket (Q \ast R)
\]

or equivalently

\[
(\llbracket \pi \rrbracket Q) \ast R \\
\llbracket \pi \rrbracket (Q \ast R)
\]

if \( \text{Modifies} (\pi) \cap \text{Free}(R) = \emptyset \)
Soundness of Frame Rule

\[
\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P \ast R \rightarrow \llbracket \pi \rrbracket (Q \ast R)} \quad \text{or equivalently} \quad \frac{(\llbracket \pi \rrbracket Q) \ast R}{\llbracket \pi \rrbracket (Q \ast R)}
\]

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

\(\implies\)

Instantiate left rule with \( P := \llbracket \pi \rrbracket Q \).
Premiss: trivially true, conclusion: desired implication.
Soundness of Frame Rule

\[
P \rightarrow \lfloor \pi \rfloor Q \\
P \ast R \rightarrow \lfloor \pi \rfloor (Q \ast R)
\]

or equivalently

\[
\lfloor \pi \rfloor Q \ast R \\
\lfloor \pi \rfloor (Q \ast R)
\]

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

\[\implies\]

Instantiate left rule with \( P := \lfloor \pi \rfloor Q \).

Premiss: trivially true, conclusion: desired implication.

\[\iff\]

Let \( \beta, h \models P \ast R \), i.e., \( \beta, h_1 \models P \) and \( \beta, h_2 \models R \) with \( h = h_1 \cup h_2 \).

By premiss: \( \beta, h_1 \models \lfloor \pi \rfloor Q \) and \( \beta, h \models (\lfloor \pi \rfloor Q) \ast R \)

Right rule gives: \( \beta, h \models \lfloor \pi \rfloor (Q \ast R) \)
Soundness of Frame Rule

\[
\frac{P \rightarrow \llbracket \pi \rrbracket Q}{P \ast R \rightarrow \llbracket \pi \rrbracket (Q \ast R)} \quad \text{or equivalently} \quad \frac{(\llbracket \pi \rrbracket Q) \ast R}{\llbracket \pi \rrbracket (Q \ast R)}
\]

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

\[\Rightarrow\]

Instantiate left rule with \(P := \llbracket \pi \rrbracket Q\).
Premiss: trivially true, conclusion: desired implication.

\[\Leftarrow\]

Let \(\beta, h \models P \ast R\), i.e., \(\beta, h_1 \models P\) and \(\beta, h_2 \models R\) with \(h = h_1 \cup h_2\).
By premiss: \(\beta, h_1 \models \llbracket \pi \rrbracket Q\) and \(\beta, h \models (\llbracket \pi \rrbracket Q) \ast R\)
Right rule gives: \(\beta, h \models \llbracket \pi \rrbracket (Q \ast R)\)
Soundness of Frame Rule

\[
\frac{([\pi] Q) \ast R}{[\pi](Q \ast R)} \text{ if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset
\]

Proof by structural induction over \( \pi \).

**Case** \( x := t \)

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

\( x \) does not occur in \( R \) (by side condition):

\[\models R \leftrightarrow [\pi] R\]

Therefore: \( \beta, h_1 \models [\pi] Q \) and \( \beta, h_2 \models [\pi] R \)

After assignment: \( \beta[x/\text{val}_\beta(t)], h_1 \models Q \) and \( \beta[x/\text{val}_\beta(t)], h_2 \models R \)

and \( \beta, h \models [\pi](Q \ast R) \)
Soundness of Frame Rule

\[
\frac{([\pi]Q) \ast R}{[\pi](Q \ast R)} \quad \text{if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset
\]

Proof by structural induction over \( \pi \).

**Case** \([t] := u\)

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

Together: \( h \models [\pi](Q \ast R) \)

\( h_Q = h[val(t)/val(u)] \) after executing \( Q \)
Soundness of Frame Rule

\[
\frac{([\pi]Q) \ast R}{[\pi](Q \ast R)} \quad \text{if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset
\]

Proof by structural induction over \(\pi\).

**Case** \(\pi_1 ; \pi_2\)

Assume: \(([[\pi_1 ; \pi_2]Q) \ast R\)

\(([[\pi_1]([[\pi_2]Q)) \ast R\)

by ind. hyp.: \([\pi_1]([[\pi_2]Q) \ast R\)

by ind. hyp.: \([\pi_1][\pi_2](Q \ast R)\) using \(\square A \quad A \rightarrow B \quad \square B\)
Soundness of Frame Rule

\[
\frac{([\pi] Q) \ast R}{[\pi](Q \ast R)} \quad \text{if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset
\]

Proof by structural induction over \(\pi\).

Remaining Cases: \(\times := [t], \ ?\phi, \ \pi^*\)

similar, left as exercise
Memory Allocation and Deallocation

Syntax: Two statements

\[
\text{var} := \text{cons}(\text{term}, ..., \text{term}) \quad \text{and} \quad \text{dispose}(\text{var})
\]
Memory Allocation and Deallocation

Syntax: Two statements

\[ \text{var} := \text{cons}(\text{term}, \ldots, \text{term}) \quad \text{and} \quad \text{dispose(\text{var})} \]

Semantics: \( \rho \) and \( \text{fail} \)

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

\[
\text{fail}(\text{v} := \text{cons}(t_1, \ldots t_n)) = \emptyset
\]

\text{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

\[
\text{var} := \text{cons}(\text{term}, \ldots, \text{term}) \quad \text{and} \quad \text{dispose}(	ext{var})
\]

Semantics: \(\rho\) and \(\text{fail}\)

\[
((\beta, h), (\beta', h')) \in \rho(\text{dispose}(v))
\]

iff

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

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

dispose deallocates the allocated memory location \(v\);

fails if an unallocated location is disposed.
Soundness of Frame Rule

\[
\frac{([\pi]Q) \ast R}{[\pi](Q \ast R)} \quad \text{if } \text{Modifies}(\pi) \cap \text{Free}(R) = \emptyset
\]

Proof by structural induction over \( \pi \).

Case \( x := \text{cons}(e) \)

By assumption: For all \( \text{loc} \not\in \text{dom} \ h_1 \): \( Q \) holds after allocating \( \text{loc} \) in \( h_1 \).

Need to show: For all \( \text{loc} \not\in \text{dom} \ h_1 \cup h_2 \): \( Q \ast R \) holds after allocating \( \text{loc} \) in \( h \).

This is a subset of the set in the assumption.
Some restricted logics from Separation Logic are decidable.

1. Restricted arithmetic
2. No magic wand $\rightarrow\star$

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

The separating implication $\rightarrow\star$ makes undecidable.

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

```
X \rightarrow 17 \rightarrow 21 \rightarrow 9
```

Beckert, Ulbrich – Formale Systeme II: Theorie
Abstraction Predicates

Use predicate symbols to abstract away from data structures

**Example: Lists**

\[
\text{list}(x, \langle 17, 21, 9 \rangle) \iff (x \mapsto 17) * (x + 1 \mapsto v) * (v \mapsto 21) * \ldots
\]

\[
\ldots * (v + 1 \mapsto w) * (w \mapsto 9) * (w + 1 \mapsto 0)
\]
Abstraction Predicates

Use predicate symbols to abstract away from data structures

**Example:** Lists

\[
\text{list}(x, \langle 17, 21, 9 \rangle) \iff (x \mapsto 17) \ast (x+1 \mapsto v) \ast (v \mapsto 21) \ast \ldots \\
\ldots \ast (v + 1 \mapsto w) \ast (w \mapsto 9) \ast (w + 1 \mapsto 0)
\]

**General:**

Recursive predicate \textit{list}:

\[
\forall x, \nu_1, \bar{\nu}. \text{list}(x, \langle \nu_1, \bar{\nu} \rangle) \iff \exists n. ((x \mapsto \nu_1) \ast (x+1 \mapsto n) \ast \text{list}(n, \bar{\nu}))
\]
- Verifast → Demo! (Bart Jacobs et al., U Leuven)
- **Verifast → Demo!** (Bart Jacobs et al., U Leuven)

- **Infer** (Peter O’Hearn et al., Facebook)
  http://fbinfer.com/
- **Verifast → Demo!** (Bart Jacobs et al., U Leuven)

- **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)

- Infer (Peter O’Hearn et al., Facebook)
  http://fbinfer.com/

- jStar (M. Parkinson, now MS)

- 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. *)