Separation Logic

Part 1

Arthur Charguéraud

Febuary 7th, 2014

Content

Content

Example program

Assume `s` to be an existing reference. Consider the following program:

```
let n = !s in
s := 2*n;
let r = ref (3*n) in
(4*n, r)
```

Remark: OCaml syntax for reference is used:

`ref v`	to create a reference
`!r`	to read a reference
`r := v`	to update a reference

Example specification

```
let n = !s in
s := 3*n;
let r = ref (4*n) in
(2*n, r)
```

If there exists an integer n such that the initial state is described by:

$$(s \hookrightarrow n)$$

then the program returns a pair (a, r) such that the final state satisfies:

$$[a = 2n] * (s \hookrightarrow 3n) * (r \hookrightarrow 4n)$$

→ this description asserts that s and r are disjoint locations
→ all the rest of memory is implicitly assumed to be unchanged

Example of a Separation Logic triple

Specification of the example program:

$$\forall n. \quad \{s \hookrightarrow n\} \text{ program } \{\lambda(a, r).\, [a = 2n] * (s \hookrightarrow 3n) * (r \hookrightarrow 4n)\}$$

expressed using a Separation Logic *triple* of the form $\{H\}\ t\ \{\lambda x.\, H'\}$

- H describes the initial heap
- t is the term being specified
- x is a name for the output result
- H' describes the final heap and the output result

Remark: irrelevance of names

```
let n = !s in
s := 3*n;
let r = ref (4*n) in
(2*n, r)
```

We named logical variables like in the program:

$$\forall n. \quad \{s \hookrightarrow n\} \text{ program } \{\lambda(a,r).\,[a = 2n] * (s \hookrightarrow 3n) * (r \hookrightarrow 4n)\}$$

but we could have picked other names:

$$\forall m. \quad \{s \hookrightarrow m\} \text{ program } \{\lambda(a,p).\,[a = 2m] * (s \hookrightarrow 3m) * (p \hookrightarrow 4m)\}$$

$\rightarrow$ Note, however, that there is a confusion between `s` as a program variable appearing in the code and s as a logical variable appearing in the specification. This confusion will be resolved later in the course.

Content

Representation of heaps

A heap m, of type Heap, is a finite map from location to values.

$\varnothing$	the empty heap
$\langle l \mapsto v \rangle$	a singleton heap
$m[l]$	read in a heap
$m[l \mapsto v]$	update of a heap
$m_1 \uplus m_2$	merge of two heaps

Merge is only defined for heaps with disjoint domains:

$$m_1 \perp m_2 \qquad \equiv \qquad \mathrm{dom}(m_1) \cap \mathrm{dom}(m_2) = \varnothing$$

General form of Separation Logic triples

A heap predicate H characterizes memory stores of a particular shape.
A heap predicate has type: Heap $\rightarrow$ Prop, abbreviated as Hprop.

A triple has the form $\{H\}\ t\ \{\lambda x.\, H'\}$ where

- H has type: Heap $\rightarrow$ Prop
- $\lambda x.H'$ has type: $A \rightarrow$ Heap $\rightarrow$ Prop, where A is the type of t

More generally, a triple has the form $\{H\}\ t\ \{Q\}$ where

- the pre-condition H has type: Hprop
- the post-condition Q has type: $A \rightarrow$ Hprop

Towards an interpretation of triples

Assume in this slide that triples describe the entire state.

A triple $\{H\}\, t\, \{\lambda x.\, H'\}$ is interpreted in total correctness as:

$$\forall m. \quad H\, m \quad \Rightarrow \quad \exists v.\, \exists m'. \quad t_{/m} \Downarrow v_{/m'} \;\; \wedge \;\; (\langle v/x \rangle\, H')\, m'.$$

Let $Q = \lambda x.\, H'$. We have $Q\, v = \langle v/x \rangle\, H'$.

So, a triple $\{H\}\, t\, \{Q\}$ is interpreted as:

$$\forall m. \quad H\, m \quad \Rightarrow \quad \exists v.\, \exists m'. \quad t_{/m} \Downarrow v_{/m'} \;\; \wedge \;\; Q\, v\, m'.$$

Interpretation of Separation Logic triples

In Separation Logic, a triple describes only a part m_1 of the heap. The rest of the heap, call it m_2, is assumed to remain unchanged.

A triple $\{H\}\ t\ \{Q\}$ is thus interpreted as:

$$\forall m_1\, m_2.\ \begin{cases} H\, m_1 \\ m_1 \perp m_2 \end{cases} \Rightarrow\ \exists v.\, \exists m_1'.\ \begin{cases} t_{/m_1 \uplus m_2} \Downarrow v_{/m_1' \uplus m_2} \\ Q\, v\, m_1' \\ m_1' \perp m_2 \end{cases}$$

Content

Primitive heap predicates

We start by defining the basic heap predicates:

$[\,]$	empty heap
$[P]$	pure fact
$l \hookrightarrow v$	singleton heap
$H * H'$	separating conjunction
$\exists x.\, H$	existential quantification

Note that these predicates all have type Hprop (that is, Heap $\rightarrow$ Prop).

Empty heap and pure facts

Definition:

$$
\begin{aligned}
[\,] &\equiv \lambda m.\ m = \varnothing \\
[P] &\equiv \lambda m.\ m = \varnothing \ \wedge\ P
\end{aligned}
$$

Example: specification of the value 3:

$$\{\,[\,]\,\}\ 3\ \{\lambda x.\,[x = 3]\}$$

Specification using pure facts

Specify the recursive factorial function:

```
let rec facto n =
  if n = 0 then 1 else n*facto(n-1)
```

by filling in the dots below, using $n!$ to denote the mathematical factorial.

$$\forall n.\ \{...\}\ (\mathsf{facto}\, n)\ \{...\}$$

Solution:

$$\forall n. \quad \{[n \geqslant 0]\}\ (\mathsf{facto}\, n)\ \{\lambda x.\, [x = n!]\}$$

Singleton heap

Definition:

$$l \hookrightarrow v \qquad \equiv \qquad \lambda m.\ m = \langle l \mapsto v \rangle$$

Exercise. complete the specification: $\{...\}$ (`ref 3`) $\{...\}$.
Solution:

$$\{\,[\,]\,\}\ (\texttt{ref 3})\ \{\lambda r.\ \ r \hookrightarrow 3\}$$

Exercise: specify "`incr r`", defined as "`let x = !r in r := x+1`".
Remark: tt is the unit value and "$\lambda tt.$" is like "fun() $\rightarrow$" in Caml.
Solution:

$$\forall rn.\quad \{r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ \ r \hookrightarrow n+1\}$$

Separation Logic's star

The heap predicate $H_1 * H_2$ characterizes a heap made of two disjoint parts, one that satisfies H_1 and one that satisfies H_2.

Definition:

$$H_1 * H_2 \quad \equiv \quad \lambda m.\ \exists m_1 m_2. \begin{cases} m_1 \perp m_2 \\ m = m_1 \uplus m_2 \\ H_1\, m_1 \\ H_2\, m_2 \end{cases}$$

Example: $(r \hookrightarrow 3) * (s \hookrightarrow 4)$ describes two disjoint memory cells.

Working with separating conjunction

Exercise:

Specify the term "`incr r; !s`". (You may assume $\mathtt{r} \neq \mathtt{s}$.)

Solution:

$$\begin{array}{ll} \forall nm. & \{(r \hookrightarrow n) * (s \hookrightarrow m)\} \\ & (\mathtt{incr\ r;\ !s}) \\ & \{\lambda x.\, [x = m] * (r \hookrightarrow n + 1) * (s \hookrightarrow m)\} \end{array}$$

Separating conjunction and aliasing

Separation Logic supports reasoning about aliased references.

```
let r = ref 3 in
let s = ref 4 in
let t = id s in
incr s;

incr t;

!s
```

$$\{r \hookrightarrow 3\}$$
$$\{r \hookrightarrow 3 * s \hookrightarrow 4\}$$
$$\{r \hookrightarrow 3 * s \hookrightarrow 4 * [t = s]\}$$
$$\{r \hookrightarrow 3 * s \hookrightarrow 5 * [t = s]\}$$
$$\{r \hookrightarrow 3 * t \hookrightarrow 5 * [t = s]\}$$
$$\{r \hookrightarrow 3 * t \hookrightarrow 6 * [t = s]\}$$
$$\{r \hookrightarrow 3 * s \hookrightarrow 6 * [t = s]\}$$

where the identity function has specification: $\{[\,]\}$ (`id a`) $\{\lambda x.\,[x = a]\}$.

Towards Separation Logic derivations

Specification of "`incr r`":

$$\frac{\{(r \hookrightarrow n)\}\ (\texttt{incr r})\ \{\lambda tt.\,(r \hookrightarrow n+1)\}}{\{(s \hookrightarrow m) * (r \hookrightarrow n)\}\ (\texttt{incr r})\ \{\lambda tt.\,(r \hookrightarrow n+1) * (s \hookrightarrow m)\}}$$

Specification of "`!s`":

$$\frac{\{(s \hookrightarrow m)\}\ (\texttt{!s})\ \{\lambda x.\,[x = m] * (s \hookrightarrow m)\}}{\{(r \hookrightarrow n+1) * (s \hookrightarrow m)\}\ (\texttt{!s})\ \{\lambda x.\,[x = m] * (s \hookrightarrow m) * (r \hookrightarrow n+1)\}}$$

Specification of the sequence "`incr r; !s`":

$$\{(s \hookrightarrow m) * (r \hookrightarrow n)\}\ (\texttt{incr r; !s})\ \{\lambda x.\,[x = m] * (s \hookrightarrow m) * (r \hookrightarrow n+1)\}$$

A reference into another reference

Suppose we store the location of a reference into another reference:

```
let r = ref 3 in ref r
```

What is the problem with the following specification?

$$\{[\,]\}\ (\texttt{let r = ref 3 in ref r})\ \{\lambda s.\ (s \hookrightarrow r) * (r \hookrightarrow 3)\}$$

The location r needs to be quantified existentially in the post-condition:

$$\{[\,]\}\ (\texttt{let r = ref 3 in ref r})\ \{\lambda s.\ \exists r.\ (s \hookrightarrow r) * (r \hookrightarrow 3)\}$$

Existential quantification

Definition of the existential quantifier for heap predicates:

$$\exists x.\, H \quad \equiv \quad \lambda m.\ \exists x.\ H\, m$$

Existentials are introduced by allocation but also by abstraction, which is the key ingredient to the construction of modular proofs.

For example, the *precise* specification:

$$\{\,[\,]\,\}\ (\texttt{ref}\ \ 6)\ \{\lambda r.\ (r \hookrightarrow 6)\}$$

can be *weakened* into:

$$\{\,[\,]\,\}\ (\texttt{ref}\ \ 6)\ \{\lambda r.\ \exists n.\, (r \hookrightarrow n) * [\mathsf{even}\, n]\}.$$

Summary

Definition of the core Separation Logic heap predicates:

$$
\begin{array}{lcl}
[\,] & \equiv & \lambda m.\ m = \varnothing \\
[P] & \equiv & \lambda m.\ m = \varnothing \ \wedge\ P \\
l \hookrightarrow v & \equiv & \lambda m.\ m = \langle l \mapsto v \rangle \\
H_1 * H_2 & \equiv & \lambda m.\ \exists m_1 m_2. \begin{cases} m_1 \perp m_2 \\ m = m_1 \uplus m_2 \\ H_1\, m_1 \\ H_2\, m_2 \end{cases} \\
\exists x.\, H & \equiv & \lambda m.\ \exists x.\ H\, m
\end{array}
$$

Content

The separation algebra

The operator $*$ has type Hprop $\rightarrow$ Hprop $\rightarrow$ Hprop.
This binary operation satisfies the following properties.

associativity:	$(H_1 * H_2) * H_3 = H_1 * (H_2 * H_3)$	
commutativity:	$H_1 * H_2 = H_2 * H_1$	
neutral element:	$H * [\,] = H$	
scope extrusion:	$(\exists x.\, H_1) * H_2 = \exists x.\, (H_1 * H_2)$	(if $x \notin H_2$)

$\rightarrow$ Remark: these proof relies on predicate extensionality

$$(\forall x.\, P\, x \Leftrightarrow Q\, x) \Rightarrow (P = Q)$$

.

Definition of heap implication

We should be able to prove that:

$$\begin{array}{ll} \text{if} & \{\exists n.\,(r \hookrightarrow n) * [\text{even}\, n]\}\ t\ \{Q\} \\ \text{then} & \{(r \hookrightarrow 6)\}\ t\ \{Q\}. \end{array}$$

since the new pre-condition considered is stronger than the one required.

To formalize this, we introduce the heap implication relation:

$$H_1 \rhd H_2 \quad \equiv \quad \forall m.\, H_1\, m \Rightarrow H_2\, m$$

For example, we have $(r \hookrightarrow 6) \rhd \exists n.\,(r \hookrightarrow n) * [\text{even}\, n]$.

Implication between post-conditions

We should be able to prove that:

$$\begin{array}{ll} \text{if} & \{H\}\ t\ \{\lambda x.\,[x = 4] * (r \hookrightarrow 6)\} \\ \text{then} & \{H\}\ t\ \{\lambda x.\,[\mathsf{even}\,x] * \exists n.\,(r \hookrightarrow n) * [\mathsf{even}\,n]\}. \end{array}$$

To formalize this, we introduce implication between post-conditions:

$$Q_1 \blacktriangleright Q_2 \quad \equiv \quad \forall x.\, Q_1\, x \rhd Q_2\, x$$

Remark: $Q_1 \blacktriangleright Q_2$ is equivalent to: $\forall x. \forall m.\, Q_1\, x\, m \Rightarrow Q_2\, x\, m$.

Remark: $(\lambda tt.\, H_1) \blacktriangleright (\lambda tt.\, H_2)$ is equivalent to: $H_1 \rhd H_2$.

Heap implication as a partial order

The heap implication relation ($\rhd$) has type: Hprop $\rightarrow$ Hprop $\rightarrow$ Prop.

$$H_1 \rhd H_2 \quad \equiv \quad \forall m.\, H_1\, m \Rightarrow H_2\, m$$

$\rightarrow$ Our goal is to never unfold this definition in the proofs

This binary relation defines a partial order on the type Hprop:

reflexivity

$$\frac{}{H \rhd H}$$

transitivity

$$\frac{H_1 \rhd H_2 \qquad H_2 \rhd H_3}{H_1 \rhd H_3}$$

antisymmetry

$$\frac{H_1 \rhd H_2 \qquad H_2 \rhd H_1}{H_1 = H_2}$$

Moreover, heap implication is regular w.r.t. the star operator:

$$\frac{H_1 \rhd H_2 \qquad H_1' \rhd H_2'}{H_1 * H_1' \rhd H_2 * H_2'}$$

Simplifications in heap implications

Regularity can be reformulated into a cancellation rule:

$$\frac{H_1 \quad \rhd H_1'}{H_1 * H_2 \rhd H_1' * H_2}$$

For example, to prove:

$$(r \hookrightarrow 6) * [\,] * (s \hookrightarrow 3) \quad \rhd \quad \exists n.\, (s \hookrightarrow 3) * (r \hookrightarrow n) * [\mathsf{even}\, n]$$

using the algebra laws and the cancellation rule, it suffices to prove:

$$(r \hookrightarrow 6) \quad \rhd \quad \exists n.\, (r \hookrightarrow n) * [\mathsf{even}\, n]$$

Heap implications: true or false?

1.	$(r \hookrightarrow 3) * (s \hookrightarrow 4) \rhd (s \hookrightarrow 4) * (r \hookrightarrow 3)$	true
2.	$(r \hookrightarrow 3) \rhd (s \hookrightarrow 4) * (r \hookrightarrow 3)$	false
3.	$(s \hookrightarrow 4) * (r \hookrightarrow 3) \rhd (r \hookrightarrow 4)$	false
4.	$(s \hookrightarrow 4) * (r \hookrightarrow 3) \rhd (r \hookrightarrow 3)$	false
5.	$[\mathsf{False}] * (r \hookrightarrow 3) \rhd (s \hookrightarrow 4) * (r \hookrightarrow 4)$	true
6.	$(r \hookrightarrow 4) * (s \hookrightarrow 3) \rhd [\mathsf{False}]$	false
7.	$(r \hookrightarrow 4) * (r \hookrightarrow 3) \rhd [\mathsf{False}]$	true
8.	$(r \hookrightarrow 3) * (r \hookrightarrow 3) \rhd [\mathsf{False}]$	true
9.	$\exists n. (r \hookrightarrow n) \rhd (r \hookrightarrow 3)$	false
10.	$\exists n. (r \hookrightarrow n) \rhd (r \hookrightarrow n)$	false
11.	$(r \hookrightarrow 3) \rhd \exists n. (r \hookrightarrow n)$	true
12.	$(r \hookrightarrow 3) * (s \hookrightarrow 3) \rhd \exists n. (r \hookrightarrow n) * (s \hookrightarrow n)$	true
13.	$\exists n. (r \hookrightarrow n) * [n > 0] * [n < 0] \rhd [\mathsf{False}]$	true

Instantiation in RHS

To prove $(r \hookrightarrow 6) \rhd (\exists n.\, (r \hookrightarrow n) * [\mathsf{even}\, n])$, it suffices to exhibit an even number n such that $(r \hookrightarrow n)$.

Corresponding reasoning rules:

$$\frac{H_1 \rhd (\langle v/x \rangle\, H_2)}{H_1 \rhd (\exists x.\, H_2)} \qquad\qquad \frac{(H_1 \rhd H_2) \qquad P}{H_1 \rhd (H_2 * [P])}$$

Example:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{(r \hookrightarrow 6) \rhd (r \hookrightarrow 6)} \qquad \cfrac{}{\mathsf{even}\, 6}}{(r \hookrightarrow 6) \rhd (r \hookrightarrow 6) * [\mathsf{even}\, 6]}}{(r \hookrightarrow 6) \rhd \langle 6/n \rangle\, ((r \hookrightarrow n) * [\mathsf{even}\, n])}}{(r \hookrightarrow 6) \rhd \exists n.\, (r \hookrightarrow n) * [\mathsf{even}\, n]}$$

Extraction from LHS

To prove $(\exists n.\, [\mathsf{even}\, n] * (r \hookrightarrow n)) \rhd H$, it suffices to prove that, for any even number n, we have $(r \hookrightarrow n) \rhd H$.

Corresponding reasoning rules:

$$\frac{\Gamma,\ x : A \ \vdash\ H_1 \rhd H_2}{\Gamma \ \vdash\ (\exists (x : A).\, H_1) \rhd H_2}\ (x \notin H_2) \qquad \frac{\Gamma,\ P \ \vdash\ H_1 \rhd H_2}{\Gamma \ \vdash\ ([P] * H_1) \rhd H_2}$$

Same with implicit proof contexts:

$$\frac{\forall x.\ (H_1 \rhd H_2)}{(\exists x.\, H_1) \rhd H_2} \qquad \frac{P \ \Rightarrow\ (H_1 \rhd H_2)}{([P] * H_1) \rhd H_2}$$

Extraction and instantiation: an example

Exercise: prove the following heap implication.

$$\exists n.\, [n \neq 2 \wedge \mathsf{prime}\, n] * (r \hookrightarrow n) \quad \rhd \quad \exists k.\, (r \hookrightarrow (2k+1))$$

Solution:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{n : \mathsf{int} \;\vdash\; (r \hookrightarrow n) \;\rhd\; (r \hookrightarrow n)}}{n : \mathsf{int},\; p : \mathsf{int},\; n = 2p+1 \;\vdash\; (r \hookrightarrow n) \;\rhd\; (r \hookrightarrow 2p+1)}}{n : \mathsf{int},\; p : \mathsf{int},\; n = 2p+1 \;\vdash\; (r \hookrightarrow n) \;\rhd\; \langle p/k \rangle (r \hookrightarrow 2k+1)}}{n : \mathsf{int},\; p : \mathsf{int},\; n = 2p+1 \;\vdash\; (r \hookrightarrow n) \;\rhd\; \exists k.\, (r \hookrightarrow 2k+1)}}{n : \mathsf{int},\; n \neq 2,\; \mathsf{prime}\, n \;\vdash\; (r \hookrightarrow n) \;\rhd\; \exists k.\, (r \hookrightarrow 2k+1)}}{\varnothing \;\vdash\; \exists n.\, [n \neq 2 \wedge \mathsf{prime}\, n] * (r \hookrightarrow n) \;\rhd\; \exists k.\, (r \hookrightarrow 2k+1)}$$

Content

Towards a frame rule

Recall the reasoning that applied to the specification of "`incr r`":

$$\frac{\{r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ r \hookrightarrow (n+1)\}}{\{s \hookrightarrow m * r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ r \hookrightarrow (n+1) * s \hookrightarrow m\}}$$

More generally, we can frame over any heap predicate H:

$$\frac{\{r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ r \hookrightarrow (n+1)\}}{\{H * r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ r \hookrightarrow (n+1) * H\}}$$

The frame rule

The frame rule asserts that any triple remains valid when both the pre- and the post-condition are extended with a same heap predicate.

$$\frac{\{H_1\}\ t\ \{\lambda x.\, H_1'\}}{\{H_1 * H_2\}\ t\ \{\lambda x.\, H_1' * H_2\}}$$

The frame rule can be reformulated more concisely:

$$\frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 * H_2\}\ t\ \{Q_1 \star H_2\}}$$

where $Q \star H \ \equiv\ \lambda x.\ (Q\, x * H)$.

Interpretation of the frame rule

Reformulation of the frame rule:

$$\frac{H = H_1 * H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 = Q}{\{H\}\ t\ \{Q\}}$$

The execution of t cannot affect any data that is not described by H_1. In particular, H_2 is completely hidden from the reasoning about t.

Frame rule and allocation

The specification of "`ref 3`" ensures that the result is a fresh location.

$$\{\,[\,]\,\}\ (\texttt{ref}\ \ 3)\ \{\lambda r.\ (r \hookrightarrow 3)\}$$

For example, applying the frame rule with $s \hookrightarrow 5$ gives:

$$\{s \hookrightarrow 5\}\ (\texttt{ref}\ \ 3)\ \{\lambda r.\ (r \hookrightarrow 3) * (s \hookrightarrow 5)\}$$

The post-condition ensures $r \neq s$.

Strengthening of the pre-condition

We should be able to prove that

$$\begin{array}{ll} \text{if} & \{\exists n.\,(r \hookrightarrow n) * [\mathsf{even}\, n]\}\ t\ \{Q\} \\ \text{then} & \{r \hookrightarrow 6\}\ t\ \{Q\}. \end{array}$$

Rule for strengthening pre-conditions:

$$\frac{H \rhd H' \qquad \{H'\}\ t\ \{Q\}}{\{H\}\ t\ \{Q\}}$$

Weakening of the post-condition

We should be able to prove that

$$\begin{array}{ll} \text{if} & \{H\}\ t\ \{\lambda x.\,[x = 4] * (r \hookrightarrow 6)\} \\ \text{then} & \{H\}\ t\ \{\lambda x.\,[\mathsf{even}\ x] * \exists n.\,(r \hookrightarrow n) * [\mathsf{even}\ n]\}. \end{array}$$

Rule for weakening post-conditions:

$$\frac{\{H\}\ t\ \{Q'\} \qquad Q' \blacktriangleright Q}{\{H\}\ t\ \{Q\}}$$

Recall that $Q' \blacktriangleright Q$ is defined as: $\forall v.\ Q'\ v \rhd Q\ v$.

The rule of consequence

The two rules

$$\frac{H \rhd H' \qquad \{H'\}\ t\ \{Q\}}{\{H\}\ t\ \{Q\}} \quad \text{and} \quad \frac{\{H\}\ t\ \{Q'\} \qquad Q' \blacktriangleright Q}{\{H\}\ t\ \{Q\}}$$

can be combined into the rule of consequence:

$$\frac{H \rhd H' \qquad \{H'\}\ t\ \{Q'\} \qquad Q' \blacktriangleright Q}{\{H\}\ t\ \{Q\}}$$

$\rightarrow$ H and H' must cover the same set of memory cells. Same for Q and Q'.

Need for garbage collection

We would like the following triple to hold:

$$\{\,[\,]\,\}\ (\texttt{let r = ref 3 in !r})\ \{\lambda x.\,[x = 3]\}$$

Yet, the post-condition we have is "$\lambda x.\,[x = 3] * \exists r.\,(r \hookrightarrow 3)$".

We need to be able to discard $\exists r.\, r \hookrightarrow 3$ from the post-condition.

Some form of garbage collection in the program logic is needed to reflect the garbage collection performed in the programming language.

The garbage collection rules

Garbage collection for discarding a piece of the post-condition:

$$\frac{\{H\}\ t\ \{Q \star H'\}}{\{H\}\ t\ \{Q\}}\ \text{gc-post}$$

Garbage collection for discarding a piece of the pre-condition:

$$\frac{\{H\}\ t\ \{Q\}}{\{H * H'\}\ t\ \{Q\}}\ \text{gc-pre}$$

Exercise: show that gc-pre is derivable from gc-post and frame.

$$\frac{\dfrac{\{H\}\ t\ \{Q\}}{\{H * H'\}\ t\ \{Q \star H'\}}\ \text{frame}}{\{H * H'\}\ t\ \{Q\}}\ \text{gc-post}$$

The combined rule

Consider the three structural rules:

$$\frac{H = H_1 * H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 = Q}{\{H\}\ t\ \{Q\}}\ \text{frame}$$

$$\frac{H \rhd H' \qquad \{H'\}\ t\ \{Q'\} \qquad Q' \blacktriangleright Q}{\{H\}\ t\ \{Q\}}\ \text{conseq} \qquad \frac{\{H\}\ t\ \{Q \star H'\}}{\{H\}\ t\ \{Q\}}\ \text{gc}$$

They can be combined into a single rule:

$$\frac{H \rhd H_1 * H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 \blacktriangleright Q \star H_3}{\{H\}\ t\ \{Q\}}$$

Extraction of existentials and propositions

Consider the triple

$$\{\exists n.\, (r \hookrightarrow n) * [\mathsf{even}\, n]\}\ (\texttt{!r})\ \{Q\}.$$

To prove it, we need to show that, for any even number n, we have

$$\{r \hookrightarrow n\}\ (\texttt{!r})\ \{Q\}.$$

Corresponding reasoning rules:

$$\frac{\forall x.\, \{H\}\ t\ \{Q\}}{\{\exists x.\, H\}\ t\ \{Q\}} \qquad \frac{P \Rightarrow \{H\}\ t\ \{Q\}}{\{[P] * H\}\ t\ \{Q\}}$$

Same, with explicit proof contexts:

$$\frac{\Gamma,\, x \ \vdash\ \{H\}\ t\ \{Q\}}{\Gamma\ \vdash\ \{\exists x.\, H\}\ t\ \{Q\}} \qquad \frac{\Gamma,\, P \ \vdash\ \{H\}\ t\ \{Q\}}{\Gamma\ \vdash\ \{[P] * H\}\ t\ \{Q\}}$$

Content

Grammar of values and terms

$$
\begin{array}{rcl}
v & := & \mid\ x \mid tt \mid b \mid n \mid (v_1, v_2) \mid v_1 \,\mathsf{op}\, v_2 \quad \text{(if op total)} \\
t & := & \mid\ v \\
 & & \mid\ t_1 \,;\, t_2 \\
 & & \mid\ \mathsf{let}\, x = t_1 \,\mathsf{in}\, t_2 \\
 & & \mid\ \mathsf{if}\, v \,\mathsf{then}\, t_1 \,\mathsf{else}\, t_2 \\
 & & \mid\ \mathsf{let\,rec}\, f = \lambda x.\, t_1 \,\mathsf{in}\, t_2 \\
 & & \mid\ (v_1\, v_2)
\end{array}
$$

Remarks:

- Terms are put in A-normal form:

$$
\begin{array}{lcl}
\mathsf{if}\, t_0 \,\mathsf{then}\, t_1 \,\mathsf{else}\, t_2 & \equiv & \mathsf{let}\, v = t_0 \,\mathsf{in}\ \mathsf{if}\, v \,\mathsf{then}\, t_1 \,\mathsf{else}\, t_2 \\
(t_1\, t_2) & \equiv & \mathsf{let}\, f = t_1 \,\mathsf{in}\, \mathsf{let}\, v = t_2 \,\mathsf{in}\, (f\, v)
\end{array}
$$

- "$\mathsf{let\,rec}\, f = \lambda x.\, t_1 \,\mathsf{in}\, t_2$" is another notation for "$\mathsf{let\,rec}\, f\, x = t_1 \,\mathsf{in}\, t_2$".

Towards Separation Logic derivations

Recall the specification of the sequence "`incr r; !s`":

$$\frac{\begin{array}{c}\{r \hookrightarrow n * s \hookrightarrow m\}\ (\texttt{incr r})\ \{\lambda tt.\, r \hookrightarrow n+1 * s \hookrightarrow m\} \\ \{r \hookrightarrow n+1 * s \hookrightarrow m\}\ (\texttt{!s})\ \{\lambda x.\, [x = m] * r \hookrightarrow n+1 * s \hookrightarrow m\}\end{array}}{\{r \hookrightarrow n * s \hookrightarrow m\}\ (\texttt{incr r; !s})\ \{\lambda x.\, [x = m] * r \hookrightarrow n+1 * s \hookrightarrow m\}}$$

We need a reasoning rule to establish triples for sequences, of the form:

$$\frac{\{...\}\ t_1\ \{...\} \qquad \{...\}\ t_2\ \{...\}}{\{H\}\ (t_1\ ;\ t_2)\ \{Q\}}$$

Reasoning rule for sequences

First possibility:

$$\frac{\{H\}\ t_1\ \{\lambda tt.\, H'\} \qquad \{H'\}\ t_2\ \{Q\}}{\{H\}\ (t_1\ ;\ t_2)\ \{Q\}}$$

Second possibility:

$$\frac{\{H\}\ t_1\ \{Q'\} \qquad \{Q'\ tt\}\ t_2\ \{Q\}}{\{H\}\ (t_1\ ;\ t_2)\ \{Q\}}$$

Remark: $Q = \lambda tt.\, H'$ is equivalent to $Q\ tt = H'$.

Reasoning rule for values

Recall the specification of the term 3.

$$\{\,[\,]\,\}\ 3\ \{\lambda x.\,[x = 3]\}$$

The reasoning rule for values could take the form:

$$\frac{}{\{\,[\,]\,\}\ v\ \{\lambda x.\,[x = v]\}}$$

Exercise. State a reasoning rule for values in the form:

$$\frac{\ldots \rhd \ldots}{\{H\}\ v\ \{Q\}}$$

Solution:

$$\frac{H \rhd Q\,v}{\{H\}\ v\ \{Q\}}$$

Remark: $[\,] \rhd ((\lambda x.\,[x = 3])\,3)$.

Reasoning rule for let-bindings

Exercise: complete the reasoning rule below.

$$\frac{\{...\}\ t_1\ \{...\} \qquad \forall x.\ \big(\{...\}\ t_2\ \{...\}\big)}{\{H\}\ (\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2)\ \{Q\}}$$

Solution:

$$\frac{\{H\}\ t_1\ \{Q'\} \qquad \forall x.\ \{Q'\ x\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2)\ \{Q\}}$$

Example of let-binding

Consider the triple:

$$\{r \hookrightarrow 3\}\ (\texttt{let a = !r in a+1})\ \{Q\}$$

What should Q be? What should Q' be in the rule for let-bindings?

$$\begin{array}{rcl} Q' & \equiv & \lambda y.\ [y = 3] * (r \hookrightarrow 3) \\ Q & \equiv & \lambda x.\ [x = 4] * (r \hookrightarrow 3) \end{array}$$

$$\dfrac{\vdash \{r \hookrightarrow 3\}\ (\texttt{!r})\ \{Q'\} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\,] \rhd [\,] \qquad 4 = 4}{[\,] \rhd [4 = 4]}}{a : \mathsf{int},\ a = 3 \vdash [\,] \rhd (\lambda x.\ [x = 4])\ (a + 1)}}{a : \mathsf{int},\ a = 3 \vdash \{[\,]\}\ (\texttt{a+1})\ \{\lambda x.\ [x = 4]\}}}{a : \mathsf{int},\ a = 3 \vdash \{r \hookrightarrow 3\}\ (\texttt{a+1})\ \{Q\}}}{a : \mathsf{int} \vdash \{[a = 3] * (r \hookrightarrow 3)\}\ (\texttt{a+1})\ \{Q\}}}{a : \mathsf{int} \vdash \{Q'\ a\}\ (\texttt{a+1})\ \{Q\}}}{\vdash \{r \hookrightarrow 3\}\ (\texttt{let a = !r in a+1})\ \{Q\}}$$

Reasoning rule for conditionals

Consider the example:

$$\{r \hookrightarrow y\}\ (\texttt{if x > 0 then (r := x) else (r := -x)})\ \{\lambda tt.\ r \hookrightarrow |x|\}$$

To establish such a triple, we need the reasoning rule for conditionals:

$$\frac{(b = \mathsf{true} \Rightarrow \{H\}\ t_1\ \{Q\}) \qquad (b = \mathsf{false} \Rightarrow \{H\}\ t_2\ \{Q\})}{\{H\}\ (\mathsf{if}\, b\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2)\ \{Q\}}$$

On the example, we prove the else-branch as follows:

$$\frac{\begin{array}{c} x \leqslant 0 \ \vdash\ \{r \hookrightarrow y\}\ (\texttt{r := -x})\ \{\lambda tt.\ r \hookrightarrow (-x)\} \\ x \leqslant 0 \ \vdash\ (\lambda tt.\ r \hookrightarrow (-x)) \blacktriangleright (\lambda tt.\ r \hookrightarrow |x|) \end{array}}{(x > 0) = \mathsf{false} \ \vdash\ \{r \hookrightarrow y\}\ (\texttt{r := -x})\ \{\lambda tt.\ r \hookrightarrow |x|\}}\ \text{weaken}$$

Reasoning rule for top-level functions

Given a top-level function f defined as $\lambda x.\, t$, the behavior of the application $(f\, v)$ is the same as the behavior of the term $\langle v/x \rangle t$.

Reasoning rule for inlining of the body of the function:

$$\frac{f \equiv \lambda x.\, t \qquad \{H\}\ (\langle v/x \rangle t)\ \{Q\}}{\{H\}\ (f\, v)\ \{Q\}}$$

Verification of a function

Consider the definition of the function `incr`:

```
let incr = fun r ->
  let a = !r in
  r := a+1
```

Let us prove the following specification:

$$\forall rn. \quad \{r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ (r \hookrightarrow n+1)\}.$$

Fix r and n. By the reasoning rule for function, we have to prove

$$\{r \hookrightarrow n\}\ (\texttt{let a = !r in r := a+1})\ \{\lambda tt.\ (r \hookrightarrow n+1)\}.$$

Functions and effects

What is the specification of `f` in the following program?

```
let r = ref 3 in
let f () =
  incr r in
f();
f();
!r
```

Solution:

$$\forall n.\ \{r \hookrightarrow n\}\ (f\ tt)\ \{\lambda tt.\ r \hookrightarrow n+1\}$$

Goes from $\{r \hookrightarrow 3\}$ to $\{r \hookrightarrow 4\}$ and $\{r \hookrightarrow 5\}$, so the program prints 5.

Functions and aliasing

What are the two specifications of `swap`?

```
let swap r s =
  let a = !r in
  let b = !s in
  r := b;
  s := a
```

Solution:

$$\forall rsnm. \quad \{(r \hookrightarrow n) * (s \hookrightarrow m)\}\ (\texttt{swap r s})\ \{\lambda tt.\ (r \hookrightarrow m) * (s \hookrightarrow n)\}$$

$$\forall rsn. \quad \{[r = s] * r \hookrightarrow n\}\ (\texttt{swap r s})\ \{\lambda tt.\ r \hookrightarrow n\}$$

Functions with aliasing and effects

What are the two specifications of `f` in the following program?

```
let r = ref 3 in
let f s =
  incr r;
  incr s
```

Solution:

$$\forall snm. \quad \{(r \hookrightarrow n) * (s \hookrightarrow m)\}\ (f\ s)\ \{\lambda tt.\ (r \hookrightarrow n+1) * (s \hookrightarrow m+1)\}$$

$$\forall sn. \quad \{[r = s] * r \hookrightarrow n\}\ (f\ s)\ \{\lambda tt.\ r \hookrightarrow n+2\}$$

Verification of a recursive function

Consider the function:

```
let rec f n =
  if n = 0
    then 0
    else let y = f(n-1) in y+2
```

Exercise: what is the specification of this function? How to prove it?

$$\forall n.\ \{[n \geqslant 0]\}\ (f\ n)\ \{\lambda r.\,[r = 2n]\}.$$

By induction on n. The goal is to prove $\{[n \geqslant 0]\}\ (f\ n)\ \{\lambda r.\,[r = 2n]\}$ using the induction hypothesis:

$$\{[n - 1 \geqslant 0]\}\ (f\ (n-1))\ \{\lambda r.\,[r = 2(n-1)]\}.$$

Reasoning rule for local functions

The reasoning rule for local functions has the form:

$$\frac{\forall f.\ (\ldots) \Rightarrow \{H\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let\,rec}\ f = \lambda x.\, t_1\ \mathsf{in}\ t_2)\ \{Q\}}$$

To see what should fill the dots, recall the rule for top-level functions:

$$\frac{f\ \equiv\ \lambda x.\, t_1 \qquad \{H'\}\ (\langle v/x\rangle t_1)\ \{Q'\}}{\{H'\}\ (f\ v)\ \{Q'\}}$$

Therefore, we can assume about f the following:

$$\forall v H' Q'.\quad \{H'\}\ (\langle v/x\rangle t_1)\ \{Q'\} \quad\Rightarrow\quad \{H'\}\ (f\ v)\ \{Q'\}$$

or simply: $$\forall x H' Q'.\quad \{H'\}\ t_1\ \{Q'\} \quad\Rightarrow\quad \{H'\}\ (f\ x)\ \{Q'\}$$

Reasoning rule for local functions

The reasoning rule for local functions is therefore:

$$\frac{\forall f.\ \big(\forall x H' Q'.\ \{H'\}\ t_1\ \{Q'\} \Rightarrow \{H'\}\ (f\ x)\ \{Q'\}\big) \ \Rightarrow\ \{H\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let\,rec}\ f = \lambda x.\, t_1\ \mathsf{in}\ t_2)\ \{Q\}}$$

Note that we need no specific support for recursive functions, whose specification is simply established by induction.

Verification of a local function

Suppose our goal is to prove $\{H\}\ t\ \{Q\}$, where t is the program below:

```
let incr r =
  let a = !r in
  r := a+1 in
let s = ref 3 in
incr s;
incr s
```

By the rule for local functions, we have to prove:

$$\forall r H' Q'.\ \{H'\}\ (\texttt{let a = !r in r := a+1})\ \{Q'\} \Rightarrow \{H'\}\ (\texttt{incr r})\ \{Q'\}$$
$$\vdash \{H\}\ (\texttt{let s = ref 3 in incr s; incr s})\ \{Q\}$$

From the hypothesis, we can derive and keep the specification of `incr`:

$$\forall r n.\quad \{r \hookrightarrow n\}\ (\texttt{incr r})\ \{\lambda tt.\ (r \hookrightarrow n+1)\}.$$

Summary: rules for terms

$$\frac{H \rhd Q\, v}{\{H\}\ v\ \{Q\}} \qquad \frac{\{H\}\ t_1\ \{Q'\} \qquad \{Q'\ \mathit{tt}\}\ t_2\ \{Q\}}{\{H\}\ (t_1\,;\,t_2)\ \{Q\}}$$

$$\frac{\{H\}\ t_1\ \{Q'\} \qquad \forall x.\ \{Q'\ x\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2)\ \{Q\}}$$

$$\frac{b = \mathsf{true} \Rightarrow \{H\}\ t_1\ \{Q\} \qquad b = \mathsf{false} \Rightarrow \{H\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{if}\, b\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2)\ \{Q\}}$$

$$\frac{\forall f.\ \big(\forall x H' Q'.\ \{H'\}\ t_1\ \{Q'\} \Rightarrow \{H'\}\ (f\ x)\ \{Q'\}\big) \Rightarrow \{H\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let\,rec}\, f = \lambda x.\, t_1\, \mathsf{in}\, t_2)\ \{Q\}}$$

Remark: there is no rule for function calls; the rule for function definitions provides hypotheses for reasoning about function calls.

Summary: structural rules and primitive functions

Combined structural rule:

$$\frac{H \rhd H_1 * H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 \blacktriangleright Q \star H_3}{\{H\}\ t\ \{Q\}}$$

Extraction rules:

$$\frac{\forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\, H\}\ t\ \{Q\}} \qquad \frac{P \ \Rightarrow\ \{H\}\ t\ \{Q\}}{\{[P] * H\}\ t\ \{Q\}}$$

Specification of operations on references:

$$\begin{array}{ll} \forall v. & \{[\,]\}\ (\texttt{ref v})\ \{\lambda r.\ (r \hookrightarrow v)\} \\ \forall r v. & \{r \hookrightarrow v\}\ (\texttt{!r})\ \{\lambda x.\ [x = v] * (r \hookrightarrow v)\} \\ \forall r v v'. & \{r \hookrightarrow v'\}\ (\texttt{r := v})\ \{\lambda tt.\, (r \hookrightarrow v)\} \end{array}$$

Conclusion

In this course:

- interpretation of Separation Logic triples,
- properties of the star and of heap implication,
- structural rules of separation logic,
- reasoning rules for terms,
- support for aliasing and function with effects.

Exercise: proof of commutativity

Prove that the separating conjunction is commutative:

$$H_1 * H_2 = H_2 * H_1$$

You will need to use the definition of $*$ and predicate extensionality:

$$(\forall x.\, P\, x \Leftrightarrow Q\, x) \Rightarrow (P = Q).$$

Exercise: a heap implication

Prove the following heap implication:

$$\exists k.\, (r \hookrightarrow 4k) \quad \rhd \quad \exists n.\, (r \hookrightarrow n) * [\mathsf{even}\, n]$$

Remark: "$\mathsf{even}\, n$" is defined as "$\exists m.\, n = 2m$".

Exercise: verification of a recursive function

Complete the verification of the function:

```
let rec f n =
  if n = 0
    then 0
    else let y = f(n-1) in y+2
```

with respect to the following specification:

$$\forall n.\, n \geqslant 0 \Rightarrow \{\,[\,]\,\}\ (f\ n)\ \{\lambda r.\,[r = 2n]\}.$$

Exercise: exchange

Specify `exchange` and show that the rest of the code prints 0.

```
let a = ref 0 in
let r = ref a in

let exchange s =
  let u = !r in
  r := s;
  u in

let b = ref 1 in
let a' = exchange b in
let _ = exchange a' in
!(!r)
```

Solution to exercises

Solution: proof of commutativity

Goal: prove the equality $H_1 * H_2 \;=\; H_2 * H_1$.

By predicate extensionality, it suffices to prove:

$$\forall m. \quad (H_1 * H_2)\, m \quad \Leftrightarrow \quad (H_2 * H_1)\, m$$

By symmetry, it suffices to assume $(H_1 * H_2)\, m$ and prove $(H_2 * H_1)\, m$.

By definition of $*$, there exist m_1 and m_2 such that:

$$m_1 \perp m_2 \;\; \wedge \;\; m = m_1 \uplus m_2 \;\; \wedge \;\; H_1\, m_1 \;\; \wedge \;\; H_2\, m_2$$

The goal that we have to prove is equivalent to:

$$\exists m_1' m_2'. \quad m_1' \perp m_2' \;\; \wedge \;\; m = m_1' \uplus m_2' \;\; \wedge \;\; H_2\, m_1' \;\; \wedge \;\; H_2\, m_2'$$

We conclude by instantiating m_1' as m_2 and m_2' as m_1.

Solution: a heap implication

Exercise: prove the following heap implication.

$$\exists k.\,(r \hookrightarrow 4k) \quad \rhd \quad \exists n.\,(r \hookrightarrow n) * [\mathsf{even}\, n]$$

Solution:

$$\dfrac{\forall k. \quad \dfrac{\dfrac{\dfrac{\dfrac{}{(r \hookrightarrow 4k) \rhd (r \hookrightarrow 4k)} \qquad \dfrac{}{\mathsf{even}\,(4k)}}{(r \hookrightarrow 4k) \rhd (r \hookrightarrow 4k) * [\mathsf{even}\, 4k]}}{(r \hookrightarrow 4k) \rhd \langle 4k/n \rangle ((r \hookrightarrow n) * [\mathsf{even}\, n])}}{(r \hookrightarrow 4k) \rhd \exists n.\,(r \hookrightarrow n) * [\mathsf{even}\, n]}}{\exists k.\,(r \hookrightarrow 4k) \rhd \exists n.\,(r \hookrightarrow n) * [\mathsf{even}\, n]}$$

Exercise: verification of a recursive function

Complete the verification of the function:

```
let rec f n =
  if n = 0
    then 0
    else let y = f(n-1) in y+2
```

with respect to the following specification:

$$\forall n.\, n \geqslant 0 \Rightarrow \{\,[\,]\,\}\ (f\, n)\ \{\lambda r.\, [r = 2n]\}.$$

By induction on n. The goal is to prove $\{[n \geqslant 0]\}\ (f\, n)\ \{\lambda r.\, [r = 2n]\}$ using the induction hypothesis:

$$\{[n - 1 \geqslant 0]\}\ (f\, (n-1))\ \{\lambda r.\, [r = 2(n-1)]\}.$$

Solution: verification of a recursive function

Using the reasoning rule for functions, the goal becomes:

$$\{\,[\,]\,\}\ (\texttt{if n = 0 then 0 else let y = f(n-1) in y+2})\ \{\lambda r.\,[r = 2n]\}.$$

By the rule for conditionals:

- Case $n = 0$. The goal is: $\{\,[\,]\,\}\ 0\ \{\lambda r.\,[r = 2n]\}$.
  - we prove $[\,] \rhd ((\lambda r.\,[r = 2n])\,0)$, using the fact that $n = 0$.
- Case $n \neq 0$. Not that since $n \geqslant 0$, we have $n > 0$, thus $n - 1 \geqslant 0$. The goal is $\{\,[\,]\,\}\ (\texttt{let y = f(n-1) in y+2})\ \{\lambda r.\,[r = 2n]\}$.
  - by the let-rule, using $Q' \equiv \lambda r.\,[r = 2(n-1)]$.
  - first premise: $\{\,[\,]\,\}\ (f\ (n-1))\ \{Q'\}$, which is exactly the IH.
  - second premise: $\forall y.\ \{Q'\ y\}\ (y + 2)\ \{\lambda r.\,[r = 2n]\}$.
    - by the rule for values: $Q'\ y \rhd (\lambda r.\,[r = 2n])\,(y + 2)$.
    - after simplification: $[y = 2(n-1)] \rhd [y + 2 = 2n]$, which is true.

Solution: exchange

Program:

```
let a = ref 0 in
let r = ref a in

let exchange s =
  let u = !r in
  r := s;
  u in

let b = ref 1 in
let a' = exchange b in
let b' = exchange a' in
!(!r)
```

Specification of exchange:

$$\forall ts. \quad \{r \hookrightarrow t\}\ (\texttt{exchange s})\ \{\lambda x.\, [x = t] * r \hookrightarrow s\}$$

We frame on $a \hookrightarrow 0 * b \hookrightarrow 1$, and see the rest of the state evolve from $r \hookrightarrow a$ to $r \hookrightarrow b * [a' = a]$ and then to $r \hookrightarrow a' * [b' = b]$.