Temporary Read-Only Permissions for Separation Logic

Making Separation Logic's
Small Axioms
Smaller

Arthur Charguéraud   François Pottier

LTP meeting
Saclay, November 28, 2016

Separation and ownership

Separation logic (Reynolds, 2002) is about disjointness of heap fragments.

- what “we” own, versus what “others” own.

Therefore, it is about unique ownership.

- if we don’t own a memory cell, we cannot write it, or even read it.
- if we own it, we can read and write it.

We have either no permission or read-write permission.

The read and write axioms

The reasoning rule for writing requires and returns a unique permission :

SET

$$\{l \hookrightarrow v'\}\ (\text{set}\ l\ v)\ \{\lambda y.\ l \hookrightarrow v\}$$

So does the reasoning rule for reading :

TRADITIONAL READ AXIOM

$$\{l \hookrightarrow v\}\ (\text{get}\ l)\ \{\lambda y.\ [y = v] \star l \hookrightarrow v\}$$

They are known as “small axioms”.

But are they as small as they could be ? ...

Consequences

From memory cells and arrays,
the dichotomy extends to user-defined data structures.

For every data structure, we have either no permission or read-write permission.

Consequences

Here a specification of an array concatenation function :

$$\{a_1 \leadsto \mathsf{Array}\, L_1 \ \star\ a_2 \leadsto \mathsf{Array}\, L_2\}$$
$$(\texttt{Array.append}\, a_1\, a_2)$$
$$\{\lambda a_3.\ a_3 \leadsto \mathsf{Array}\,(L_1 \mathbin{+\!\!+} L_2) \ \star\ a_1 \leadsto \mathsf{Array}\, L_1 \ \star\ a_2 \leadsto \mathsf{Array}\, L_2\}$$

It is a bit noisy.

It also has several deeper drawbacks (see next slide).

Our goal

We would like the specification to look like this instead :

$$\{\mathrm{RO}(a_1 \rightsquigarrow \mathrm{Array}\ L_1) \star \mathrm{RO}(a_2 \rightsquigarrow \mathrm{Array}\ L_2)\}$$
$$(\texttt{Array.append}\ a_1\ a_2)$$
$$\{\lambda a_3.\ a_3 \rightsquigarrow \mathrm{Array}\ (L_1 +\!\!+ L_2)\}$$

This would be more concise,

require less bookkeeping,

make it clear that the arrays are unmodified,

and in fact would not require the arrays to be distinct.

Our means

For this purpose, we introduce temporary read-only permissions.

Thank you for your attention.

Remboursez !

What ! ?

Remboursez !

What ! ?

Couldn't one view $\mathrm{RO}(\cdot)$ as syntactic sugar ?

- No.

Remboursez !

What ! ?

Couldn't one view RO(·) as syntactic sugar ?

- No.

Couldn't one express this using fractional permissions ?

- Yes. More heavily.

Remboursez !

What ! ?

Couldn't one view RO(·) as syntactic sugar ?

- No.

Couldn't one express this using fractional permissions ?

- Yes. More heavily.

Isn't the metatheory of RO(·) very simple ?

- Yes, it is. If and once you get it right. That's the point !

Motivation

More Motivation

Separation Logic with Read-Only Permissions

The sugar hypothesis

The sugar hypothesis

Could the Hoare triple :

$$\{\mathrm{RO}(H_1) \star H_2\}\ t\ \{Q\}$$

be syntactic sugar for :

$$\{H_1 \star H_2\}\ t\ \{H_1 \star Q\}$$

?

Sugar does not reduce work

Sugar reduces apparent redundancy in specifications,

but has no effect on the proof obligations,

so does not reduce redundancy and bookkeeping in proofs.

If we must prove this :

$$\{H_1 \star H_2\}\ t\ \{H_1 \star Q\}$$

then we must work to ensure and argue that the permission H_1 is returned.

If “RO” was native, proving $\{\mathrm{RO}(H_1) \star H_2\}\ t\ \{Q\}$ would require no such work.

Sugar does not allow aliasing

If "RO" is sugar, then this specification requires a_1 and a_2 to be disjoint arrays :

$$\{\mathrm{RO}(a_1 \rightsquigarrow \mathrm{Array}\, L_1) \star \mathrm{RO}(a_2 \rightsquigarrow \mathrm{Array}\, L_2)\}$$
$$(\texttt{Array.append}\; a_1\; a_2)$$
$$\{\lambda a_3.\; a_3 \rightsquigarrow \mathrm{Array}\,(L_1 \mathbin{+\!\!+} L_2)\}$$

As a result, we must prove another specification to allow aliasing :

$$\{a \rightsquigarrow \mathrm{Array}\, L\}$$
$$(\texttt{Array.append}\; a\; a)$$
$$\{\lambda a_3.\; a_3 \rightsquigarrow \mathrm{Array}\,(L \mathbin{+\!\!+} L) \star a \rightsquigarrow \mathrm{Array}\, L\}$$

Duplicate work for us ; increased complication for the user.

If "RO" was native and duplicable, the first spec above would allow aliasing.

Sugar is deceptive

A read-only function admits an "RO" specification.

$$\{\mathrm{RO}(h \rightsquigarrow \mathrm{HashTable}\, M)\}\ (\textit{population}\ h)\ \{\lambda y.\ [y = \mathrm{card}\, M]\}$$

If "RO" is sugar, a function that can have an effect also admits an "RO" spec.

$$\{\mathrm{RO}(h \rightsquigarrow \mathrm{HashTable}\, M)\}\ (\textit{resize}\ h)\ \{\lambda().\ []\}$$

An "RO" specification, interpreted as sugar, does not mean "read-only".

Such sugar, if adopted, should use another keyword, e.g., **preserves**.

If "RO" was native, *resize* would not admit the second spec above.

Sugar causes amnesia and weakness

Suppose *population* has this "RO" specification :

$$\{\mathrm{RO}(h \leadsto \mathrm{HashTable}\, M)\}\ (\mathit{population}\ h)\ \{\lambda y.\ [y = \mathrm{card}\, M]\}$$

Suppose a hash table is a mutable record whose *data* field points to an array :

$$\begin{array}{l} h \leadsto \mathrm{HashTable}\, M := \\ \quad \exists\!\!\exists a.\, \exists\!\!\exists L.\, (h \leadsto \{\mathit{data} = a; \ldots\} \star a \leadsto \mathrm{Array}\, L \star \ldots) \end{array}$$

Suppose there is an operation *foo* on hash tables :

$$\begin{array}{ll} \mathrm{let}\ \mathit{foo}\ h = & \\ \quad \mathrm{let}\ d = h.\mathit{data}\ \mathrm{in} & \text{– read the address of the array} \\ \quad \mathrm{let}\ p = \mathit{population}\ h\ \mathrm{in} & \text{– call } \mathit{population} \\ \quad \ldots & \end{array}$$

If "RO" is sugar, then the proof of *foo* runs into a problem...

Sugar causes amnesia and weakness

Reasoning about *foo* might go like this :

1 let *foo* $h =$
2 $\{h \rightsquigarrow \text{HashTable}\, M\}$ – *foo*'s precondition
3 $\{h \rightsquigarrow \{data = a; \ldots\} \star a \rightsquigarrow \text{Array}\, L \star \ldots\}$ – by unfolding
4 let $d = h.data$ in
5 $\{h \rightsquigarrow \{data = a; \ldots\} \star a \rightsquigarrow \text{Array}\, L \star \ldots \star [d = a]\}$ – by reading
6 $\{h \rightsquigarrow \text{HashTable}\, M \star [d = a]\}$ – by folding
7 let $p = population\ h$ in – we have to fold
8 $\{h \rightsquigarrow \text{HashTable}\, M \star [d = a] \star [p = \#M]\}$
9 $\ldots$

At line 8, the equation $d = a$ is useless.

We have forgotten what d represents, and lost the benefit of the read at line 4.

With “RO” as sugar, the specification of *population* is weaker than it seems.

If “RO” was native, there would be a way around this problem. (Details omitted.)

Motivation

More Motivation

Separation Logic with Read-Only Permissions

Permissions

Permissions are as follows :

$$H \quad := \quad [P] \mid l \hookrightarrow v \mid H_1 \star H_2 \mid H_1 \vee H_2 \mid \exists x.\, H \mid \mathrm{RO}(H)$$

Every permission H has a read-only form $\mathrm{RO}(H)$.

Properties of RO

RO is well-behaved :

$$
\begin{array}{rcll}
\mathrm{RO}([P]) & = & [P] & \\
\mathrm{RO}(H_1 \star H_2) & \triangleright & \mathrm{RO}(H_1) \star \mathrm{RO}(H_2) & \text{(the reverse is false)} \\
\mathrm{RO}(H_1 \vee H_2) & = & \mathrm{RO}(H_1) \vee \mathrm{RO}(H_2) & \\
\mathrm{RO}(\exists x.\, H) & = & \exists x.\, \mathrm{RO}(H) & \\
\mathrm{RO}(\mathrm{RO}(H)) & = & \mathrm{RO}(H) & \\
\mathrm{RO}(H) & \triangleright & \mathrm{RO}(H') & \text{if } H \triangleright H' \\
\mathrm{RO}(H) & = & \mathrm{RO}(H) \star \mathrm{RO}(H) &
\end{array}
$$

A new read axiom

The traditional read axiom :

TRADITIONAL READ AXIOM

$$\{l \hookrightarrow v\}\ (\text{get}\ l)\ \{\lambda y.\ [y = v] \star l \hookrightarrow v\}$$

is replaced with a “smaller” axiom :

NEW READ AXIOM

$$\{\text{RO}(l \hookrightarrow v)\}\ (\text{get}\ l)\ \{\lambda y.\ [y = v]\}$$

A new frame rule

The traditional frame rule is subsumed by a new "read-only frame rule" :

FRAME RULE

$$\frac{\{H\}\ t\ \{Q\} \qquad \text{normal}\ H'}{\{H \star H'\}\ t\ \{Q \star H'\}}$$

READ-ONLY FRAME RULE

$$\frac{\{H \star \mathrm{RO}(H')\}\ t\ \{Q\} \qquad \text{normal}\ H'}{\{H \star H'\}\ t\ \{Q \star H'\}}$$

Upon entry into a block, H' is temporarily replaced with $\mathrm{RO}(H')$,
and upon exit, magically re-appears.

The side condition "normal H'" means roughly "H' has no RO components", so $\mathrm{RO}(H')$ cannot escape through Q.

That’s all, folks !

That’s all there is to it !

That's all, folks !

The paper gives a simple model that explains why the logic is sound.

The proof is machine-checked.

We believe that temporary read-only permissions sometimes help state more concise, accurate, useful specifications, and lead to simpler proofs.

Possible future work : an implementation in CFML.