Separation Logic

Verifying concurrent programs, compositionally

NRU HSE

Author: Pavel Sokolov

June 25, 2025

The language

Definitions adapted from [1].

$$
\begin{array}{lll}
\langle \mathit{aexpr} \rangle & ::= & \langle \mathit{int} \rangle \\
& | & \langle \mathit{ident} \rangle \\
& | & \langle \mathit{aexpr} \rangle + \langle \mathit{aexpr} \rangle \\
& | & \langle \mathit{aexpr} \rangle - \langle \mathit{aexpr} \rangle
\end{array}
$$

$$
\langle \mathit{bexpr} \rangle ::= \langle \mathit{aexpr} \rangle \text{ `<=' } \langle \mathit{aexpr} \rangle
$$

$$
\begin{array}{lll}
\langle \mathit{cmd} \rangle & ::= & \text{`}\texttt{skip}\text{'} \\
& | & \langle \mathit{ident} \rangle \text{ `}\texttt{:=}\text{' } \langle \mathit{aexpr} \rangle \\
& | & \langle \mathit{cmd} \rangle \text{ `}\texttt{;}\text{' } \langle \mathit{cmd} \rangle \\
& | & \text{`}\texttt{if}\text{' } \langle \mathit{bexpr} \rangle \text{ `}\texttt{then}\text{' } \langle \mathit{cmd} \rangle \text{ `}\texttt{else}\text{' } \langle \mathit{cmd} \rangle \text{ `}\texttt{end}\text{'} \\
& | & \text{`}\texttt{while}\text{' } \langle \mathit{bexpr} \rangle \text{ `}\texttt{do}\text{' } \langle \mathit{cmd} \rangle \text{ `}\texttt{end}\text{'}
\end{array}
$$

Hoare Triples

- Express how *commands* change the *program state*.
- Take the form `{P}c{Q}`, where *c* is a command, and *P*, *Q* are *assertions* about the program state before and after execution of *c*, correspondingly.

Examples of valid Hoare triples:

- $\{\texttt{X} = m\}\ \texttt{X:=X+1}\ \{\texttt{X} = m + 1\}$
- $\{\texttt{X} = 2 \wedge \texttt{X} = 3\}\ \texttt{X:=5}\ \{\texttt{X} = 0\}$
-

$\{\text{True}\}$
```
if X <= 0
then Y := 2
else Y := X + 1
end
```
$\{\texttt{X} \leq \texttt{Y}\}$

Hoare Logic

Cons
$$\frac{P \Rightarrow P' \qquad \{P'\}c\{Q'\} \qquad Q' \Rightarrow Q}{\{P\}c\{Q\}}$$

Skip
$$\{P\}\texttt{skip}\{P\}$$

Asgn
$$\{P[X \mapsto a]\}\, X{:=}a\, \{P\}$$

Seq
$$\frac{\{P\}c_1\{Q\} \qquad \{Q\}c_2\{R\}}{\{P\}c_1; c_2\{R\}}$$

If
$$\frac{\{P \wedge b\}c_1\{Q\} \qquad \{P \wedge \neg b\}c_2\{Q\}}{\{P\}\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}\{Q\}}$$

While
$$\frac{\{P \wedge b\}c\{P\}}{\{P\}\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}\{P \wedge \neg b\}}$$

Pointers!

$\langle \textit{expr} \rangle$ ::= $\langle \textit{aexpr} \rangle$
| $\langle \textit{bexpr} \rangle$
| '`[`'$\langle \textit{expr} \rangle$'`]`'

$\langle \textit{cmd} \rangle$ ::= $\langle \textit{ident} \rangle$ '`:=`' $\langle \textit{expr} \rangle$
| $\langle \textit{ident} \rangle$ '`:=`' '`cons`' '`(`' ($\langle \textit{expr} \rangle$,)* '`)`'
| '`[`'$\langle \textit{expr} \rangle$'`]`' '`:=`' $\langle \textit{expr} \rangle$
| '`dispose`' $\langle \textit{expr} \rangle$
| ...

Program example

In-place list reversal:

```
j := nil;
while i != nil do
  k := [i + 1];
  [i + 1] := j;
  j := i;
  i := k
end
```

How to prove properties about it?

Enter Separation Logic

Definitions adapted from [2].

$$\begin{array}{lcl} \langle \mathit{assert} \rangle & ::= & \langle \mathit{bexpr} \rangle \\ & | & \langle \mathit{assert} \rangle \wedge \langle \mathit{assert} \rangle \\ & | & \neg \langle \mathit{assert} \rangle \\ & | & \mathtt{emp} \\ & | & \langle \mathit{expr} \rangle \mapsto \langle \mathit{expr} \rangle \\ & | & \langle \mathit{assert} \rangle * \langle \mathit{assert} \rangle \\ & | & \langle \mathit{assert} \rangle -\!\!* \langle \mathit{assert} \rangle \end{array}$$

Note that assertion now depends both on the local store and on the heap!

Meaning of operators

- `emp` asserts that the heap is empty.
- $e \mapsto e'$ asserts that heap contains a single cell with address e and value e'.
- $p * p'$ asserts that heap can be **split** into two parts where p and p' hold, respectively.
- $p \mathrel{-\!\!*} p'$ asserts that, if the current heap is extended with a **disjoint** part where p holds, then p' would hold for the new state.

Examples

- $(x \mapsto y) * (x + 1 \mapsto 3)$
- $(x \mapsto y) * \texttt{True}$
- $(x \mapsto 3) * (x + 1 \mapsto y) * (y \mapsto 5) * (y + 1 \mapsto x)$
- $((x \mapsto 3) * (x + 1 \mapsto y)) \wedge ((y \mapsto 3) * (y + 1 \mapsto x))$

Separation Logic Rules

FRAME

$$\frac{\{P\}c\{Q\}}{\{P * R\}c\{Q * R\}}$$

MUT

$$\{(e \mapsto -)\}[e]{:=}e'\{e \mapsto e'\}$$

DISPOSE

$$\{e \mapsto -\}\,\mathtt{dispose}\,e\,\{\mathtt{emp}\}$$

And more...

One more example

```
  {emp}
x := cons(a, a) ;
  {(x |-> a) * (x + 1 |-> a)}
y := cons(b, b) ;
  {(x |-> a) * (x + 1 |-> -)
    * (y |-> b) * (y + 1 |-> -)}
[x + 1] := y - x ;
  {(x |-> a) * (x + 1 |-> y - x)
    * (y |-> b) * (y + 1 |-> -)}
[y + 1] := x - y ;
  {(x |-> a) * (x + 1 |-> y - x)
    * (y |-> b) * (y + 1 |-> x - y)}
```

Concurrent Separation Logic

What about programs executed in parallel?

```
x := cons(a, b);
put(x);
```

```
get(y);
use(y);
dispose(y);
```

Two more rules: PARALLEL COMPOSITION and CRITICAL REGION [3].

CSL Rules

Parallel Composition

$$\frac{\{P_1\}c_1\{Q_1\} \qquad \ldots \qquad \{P_n\}c_n\{Q_n\}}{\{P_1 * \ldots * P_n\}(c_1 \| \ldots \| c_n)\{Q_1 * \ldots * Q_n\}}$$

Critical Region

$$\frac{\{(P * R_r) \wedge b\}c\{Q * R_r\}}{\{P\}\,\texttt{with}\,r\,\texttt{when}\,b\,\texttt{do}\,c\,\{Q\}}$$

Applications

- Iris framework for Rocq prover;
- GhostCell formal verification for Rust;
- Verification of interrupts and preemptive threads in OS kernels, GC allocators,...;

Bibliography

Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey.
Programming Language Foundations, volume 2 of *Software Foundations*.
Electronic textbook, 2024.
Version 6.7, `http://softwarefoundations.cis.upenn.edu`.

John C. Reynolds.
Separation logic: A logic for shared mutable data structures.
In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, page 55–74, USA, 2002. IEEE Computer Society.

Paweł Sobociński.
Report on lics 2016.
ACM SIGLOG News, 4(1):38–39, February 2017.