Concurrent Separation Logic Lecture Notes

Viktor Vafeiadis

April 30, 2014

1 The Programming Language

1.1 Syntax

Consider the following simple programming language. Arithmetic expressions, E, consist of program variables, integer constants, and arithmetic operations.

$$E ::= x \mid n \mid E + E \mid E - E \mid \dots$$

Boolean expressions, B, consist of arithmetic equalities and inequalities and Boolean operations.

$$B ::= B \wedge B \mid \neg B \mid E = E \mid E \leq E \mid \dots$$

Commands, C, include the empty command, variable assignments, memory reads, memory writes, memory allocation, memory deallocation, sequential composition, parallel composition, non-deterministic choice, loops, assume commands and atomic commands.

$$\begin{aligned} C ::= {} & \mathbf{skip} \mid x := E \mid x := [E] \mid [E] := E \mid x := \mathbf{alloc}(E) \mid \mathbf{dispose}(E) \\ & \mid C_1; C_2 \mid C_1 \| C_2 \mid C_1 \oplus C_2 \mid C^* \mid \mathbf{assume}(B) \mid \mathbf{atomic}\ C \mid \mathbf{inatom}\ C \end{aligned}$$

The final command form, **inatom** C, is not meant to be used in programs, and is only used to define the semantics of atomic commands.

We can define conditionals and while-loops in terms of the more primitive constructs as follows:

$$\mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2 \stackrel{\text{def}}{=} (\mathbf{assume}(B); C_1) \oplus (\mathbf{assume}(\neg B); C_2)$$

$$\mathbf{while}\ B\ \mathbf{do}\ C \stackrel{\text{def}}{=} (\mathbf{assume}(B); C)^*; \mathbf{assume}(\neg B)$$

1.2 Semantic Domains

We assume a domain of variable names (VarName), a domain of memory locations (Loc) and a domain of values (Val) that includes memory locations

and define the following composite domains:

$$
\begin{array}{rcll}
s \in \mathsf{Stack} & \stackrel{\text{def}}{=} & \mathsf{VarName} \to \mathsf{Val} & \text{stacks (interpretations for variables)} \\
h \in \mathsf{Heap} & \stackrel{\text{def}}{=} & \mathsf{Loc} \rightharpoonup_{\text{fin}} \mathsf{Val} & \text{heaps (dynamically allocated memory)} \\
\sigma \in \mathsf{State} & \stackrel{\text{def}}{=} & \mathsf{Stack} \times \mathsf{Heap} & \text{program states} \\
\mathsf{Config} & \stackrel{\text{def}}{=} & \mathsf{Cmd} \times \mathsf{Stack} \times \mathsf{Heap} & \text{program configurations}
\end{array}
$$

1.3 Semantics of Expressions

Arithmetic and Boolean expressions are interpreted denotationally as total functions from stacks to values or Boolean values respectively:

$$
\begin{array}{l|l}
[\![\_]\!] : \mathsf{Exp} \to \mathsf{Stack} \to \mathsf{Val} & [\![\_]\!] : \mathsf{BoolExp} \to \mathsf{Stack} \to \{\mathsf{true}, \mathsf{false}\} \\
[\![x]\!](s) \stackrel{\text{def}}{=} s(x) & [\![B_1 \wedge B_2]\!](s) \stackrel{\text{def}}{=} [\![B_1]\!](s) \wedge [\![B_2]\!](s) \\
[\![E_1 + E_2]\!](s) \stackrel{\text{def}}{=} [\![E_1]\!](s) + [\![E_2]\!](s) & [\![E_1 \leq E_2]\!](s) \stackrel{\text{def}}{=} [\![E_1]\!](s) \leq [\![E_2]\!](s)
\end{array}
$$

1.4 Semantics of Commands

Commands are given a small-step operational semantics. Configurations are pairs (C, σ) of a command and a state. There are transitions from one configuration to another as well as transitions from a configuration to **abort** denoting execution errors such as accessing an unallocated memory location.

- The assignment rule evaluates E and then stores its result in $s(x)$.

$$\frac{}{\langle x := E, s, h\rangle \to \langle \mathbf{skip}, s[x := [\![E]\!](s)], h\rangle} \qquad (\textsc{Assign})$$

- The assume statement blocks if the condition is not satisfied.

$$\frac{[\![B]\!](s)}{\langle \mathbf{assume}(B), s, h\rangle \to \langle \mathbf{skip}, s, h\rangle} \qquad (\textsc{Assume})$$

- Memory accesses. These abort if the memory location is not allocated.

$$\frac{h([\![E]\!](s)) = v}{\langle x := [E], s, h\rangle \to \langle \mathbf{skip}, s[x := v], h\rangle} \qquad (\textsc{Read})$$

$$\frac{[\![E]\!](s) \notin \mathbf{dom}(h)}{\langle x := [E], s, h\rangle \to \mathbf{abort}} \qquad (\textsc{ReadA})$$

$$\frac{[\![E]\!](s) \in \mathbf{dom}(h)}{\langle [E]{:=}E', s, h\rangle \to \langle \mathbf{skip}, s, h[[\![E]\!](s) := [\![E']\!](s)]\rangle} \qquad (\textsc{Wri})$$

$$\frac{[\![E]\!](s) \notin \mathbf{dom}(h)}{\langle [E]{:=}E', s, h\rangle \to \mathbf{abort}} \qquad (\textsc{WriA})$$

- Memory allocation. (This cannot abort as we assume Loc is infinite, and at any point in time, the heap h contains only finitely many locations.)

$$\frac{\ell \notin \mathbf{dom}(h)}{\langle x := \mathbf{alloc}(E), s, h\rangle \to \langle \mathbf{skip}, s[x := \ell], h[\ell := [\![E]\!](s)]\rangle} \quad (\textsc{Alloc})$$

- Memory deallocation.

$$\frac{[\![E]\!](s) \in \mathbf{dom}(h)}{\langle \mathbf{dispose}(E), s, h\rangle \to \langle \mathbf{skip}, s, h[[\![E]\!](s) := \bot]\rangle} \quad (\textsc{Free})$$

$$\frac{[\![E]\!](s) \notin \mathbf{dom}(h)}{\langle \mathbf{dispose}(E), s, h\rangle \to \mathbf{abort}} \quad (\textsc{FreeA})$$

- Sequential composition:

$$\frac{}{\langle \mathbf{skip}; C_2, s, h\rangle \to \langle C_2, s, h\rangle} \quad (\textsc{Seq1})$$

$$\frac{\langle C_1, s, h\rangle \to \langle C_1', s', h'\rangle}{\langle C_1; C_2, s, h\rangle \to \langle C_1'; C_2, s', h'\rangle} \quad (\textsc{Seq2})$$

$$\frac{\langle C_1, s, h\rangle \to \mathbf{abort}}{\langle C_1; C_2, s, h\rangle \to \mathbf{abort}} \quad (\textsc{SeqA})$$

- Non-deterministic choice:

$$\frac{}{\langle C_1 \oplus C_2, s, h\rangle \to \langle C_1, s, h\rangle} \quad (\textsc{Choice1})$$

$$\frac{}{\langle C_1 \oplus C_2, s, h\rangle \to \langle C_2, s, h\rangle} \quad (\textsc{Choice2})$$

- Loops just reduce to a non-deterministic choice:

$$\frac{}{\langle C^*, s, h\rangle \to \langle (\mathbf{skip} \oplus (C; C^*)), s, h\rangle} \quad (\textsc{Loop})$$

- Atomic commands, **atomic** C, first acquire a global lock denoted by reducing to the **inatom** C step. Then, **inatom** C reduces so long as C reduces. At the end, when $C = \mathbf{skip}$, we let **inatom skip** reduce to **skip** thereby releasing the global lock.

$$\frac{}{\langle \mathbf{atomic}\ C, s, h\rangle \to \langle \mathbf{inatom}\ C, s, h\rangle} \quad (\textsc{Atom})$$

$$\frac{\langle C, s, h\rangle \to \langle C', s', h'\rangle}{\langle \textbf{inatom } C, s, h\rangle \to \langle \textbf{inatom } C', s', h'\rangle} \text{ (InAtomStep)}$$

$$\frac{}{\langle \textbf{inatom skip}, s, h\rangle \to \langle \textbf{skip}, s, h\rangle} \text{ (InAtomEnd)}$$

$$\frac{\langle C, s, h\rangle \to \textbf{abort}}{\langle \textbf{inatom } C, s, h\rangle \to \textbf{abort}} \text{ (InAtomA)}$$

Whether a command holds the global lock is represented by the predicate $\textbf{locked}(C)$, defined as follows:

$$\textbf{locked}(C) \stackrel{\text{def}}{=} \begin{cases} \mathsf{true} & \text{if } C = \textbf{inatom } C' \\ \textbf{locked}(C_1) \vee \textbf{locked}(C_2) & \text{if } C = (C_1 \| C_2) \\ \textbf{locked}(C_1) & \text{if } C = (C_1; C_2) \\ \mathsf{false} & \text{otherwise} \end{cases}$$

- Parallel composition interleaves the executions of its two components respecting the semantics of the global lock. That is, it does not execute the first thread if the second has the lock, and vice versa.

$$\frac{}{\langle (\textbf{skip} \| \textbf{skip}), s, h\rangle \to \langle \textbf{skip}, s, h\rangle} \text{ (ParEnd)}$$

$$\frac{\langle C_1, s, h\rangle \to \langle C_1', s', h'\rangle \qquad \neg\textbf{locked}(C_2)}{\langle C_1 \| C_2, s, h\rangle \to \langle C_1' \| C_2, s', h'\rangle} \text{ (Par1)}$$

$$\frac{\langle C_2, s, h\rangle \to \langle C_2', s', h'\rangle \qquad \neg\textbf{locked}(C_1)}{\langle C_1 \| C_2, s, h\rangle \to \langle C_1 \| C_2', s', h'\rangle} \text{ (Par2)}$$

$$\frac{\langle C_1, s, h\rangle \to \textbf{abort} \qquad \neg\textbf{locked}(C_2)}{\langle C_1 \| C_2, s, h\rangle \to \textbf{abort}} \text{ (ParA1)}$$

$$\frac{\langle C_2, s, h\rangle \to \textbf{abort} \qquad \neg\textbf{locked}(C_1)}{\langle C_1 \| C_2, s, h\rangle \to \textbf{abort}} \text{ (ParA2)}$$

2 Separation Logic Assertions

2.1 Syntax of Assertions

Separation logic assertions include Boolean expressions, all the classical connectives, first order quantification, and five assertions pertinent to separation logic. These are the empty heap assertion (**emp**), the points-to assertion

$(E_1 \mapsto E_2)$ indicating that the heap consists of a single memory cell with address E_1 and contents E_2, separating conjunction $(*)$, separating implication $(-\!\!*)$, and an iterative version of separating conjunction $(\circledast)$:

$$\begin{array}{rcl} P, Q, R, J ::= & B \mid P \vee Q \mid P \wedge Q \mid P \Rightarrow Q \mid \neg P \mid \forall x.\ P \mid \exists x.\ P \\ \mid & \mathbf{emp} \mid E_1 \mapsto E_2 \mid P * Q \mid P -\!\!* Q \mid \circledast_{i \in \mathcal{I}} P_i \end{array}$$

There are also a number of derived assertions. The most common are:

$$\begin{array}{rcll} E \hookrightarrow E' & \stackrel{\text{def}}{\iff} & E \mapsto E' * \mathsf{true} & \\ E \mapsto - & \stackrel{\text{def}}{\iff} & \exists v.\ E \mapsto v & \\ E \hookrightarrow - & \stackrel{\text{def}}{\iff} & \exists v.\ E \hookrightarrow v & \\ P -\!\!\circledast Q & \stackrel{\text{def}}{\iff} & \neg(P -\!\!* \neg Q) & \text{“septraction”} \end{array}$$

2.2 Semantics of Assertions

Assertions denote predicates of states (pairs of stacks and heaps). Formally, we define the denotation of an assertion,

$$[\![\_]\!] : \mathbf{Assn} \to (\mathsf{Stack} \times \mathsf{Heap}) \to \{\mathsf{true}, \mathsf{false}\}$$

by induction on the syntax of assertions as follows:[1]

$$\begin{array}{rcl} [\![B]\!](s,h) & \stackrel{\text{def}}{\iff} & [\![B]\!](s) \qquad \text{— N.B., the heap } h \text{ can be arbitrary} \\ [\![P \vee Q]\!](s,h) & \stackrel{\text{def}}{\iff} & [\![P]\!](s,h) \vee [\![Q]\!](s,h) \\ [\![P \wedge Q]\!](s,h) & \stackrel{\text{def}}{\iff} & [\![P]\!](s,h) \wedge [\![Q]\!](s,h) \\ [\![P \Rightarrow Q]\!](s,h) & \stackrel{\text{def}}{\iff} & [\![P]\!](s,h) \implies [\![Q]\!](s,h) \\ [\![\neg P]\!](s,h) & \stackrel{\text{def}}{\iff} & \neg [\![P]\!](s,h) \\ [\![\forall x.\ P]\!](s,h) & \stackrel{\text{def}}{\iff} & \forall v.\ [\![P]\!](s[x := v], h) \\ [\![\exists x.\ P]\!](s,h) & \stackrel{\text{def}}{\iff} & \exists v.\ [\![P]\!](s[x := v], h) \\ [\![\mathbf{emp}]\!](s,h) & \stackrel{\text{def}}{\iff} & \mathbf{dom}(h) = \emptyset \\ [\![E \mapsto E']\!](s,h) & \stackrel{\text{def}}{\iff} & \mathbf{dom}(h) = [\![E]\!](s) \wedge h([\![E]\!](s)) = [\![E']\!](s) \\ [\![P * Q]\!](s,h) & \stackrel{\text{def}}{\iff} & \exists h_1, h_2.\ h = h_1 \uplus h_2 \wedge [\![P]\!](s,h_1) \wedge [\![Q]\!](s,h_2) \\ [\![P -\!\!* Q]\!](s,h) & \stackrel{\text{def}}{\iff} & \forall h_1.\ \mathsf{def}(h \uplus h_1) \wedge [\![P]\!](s,h_1) \implies [\![Q]\!](s, h \uplus h_1) \\ [\![\circledast_{i \in \mathcal{I}} P_i]\!](s,h) & \stackrel{\text{def}}{\iff} & \exists H : \mathcal{I} \to \mathsf{Heap}. h = \biguplus_{i \in \mathcal{I}} H(i) \wedge \forall i.\ [\![P_i]\!](s, H(i)) \end{array}$$

Here, $h_1 \uplus h_2$ stands for the union of the two heaps h_1 and h_2 and is undefined unless $\mathbf{dom}(h_1) \cap \mathbf{dom}(h_2) = \emptyset$. We write $\mathsf{def}(X)$ to say that X is defined. For example, $\mathsf{def}(h_1 \uplus h_2)$ simply means that $\mathbf{dom}(h_1) \cap \mathbf{dom}(h_2) = \emptyset$.

[1] In the literature, $[\![P]\!](s,h)$ is often written as $s, h \models P$.

2.3 Precise Assertions

An important class of assertions are the so-called *precise* assertions, which are assertions satisfied by at most one subheap of any given heap. Formally, if there are satisfied by two such heaps, h_1 and h_1', the two must be equal:

Definition 1. *An assertion, P, is* precise *iff for all s, h_1, h_2, h_1', and h_2', if $\mathsf{def}(h_1 \uplus h_2)$ and $h_1 \uplus h_2 = h_1' \uplus h_2'$ and $[\![P]\!](s, h_1)$ and $[\![P]\!](s, h_1')$, then $h_1 = h_1'$ and $h_2 = h_2'$.*

3 Concurrent Separation Logic Judgments

CSL judgments are of the form, $J \vdash \{P\}\ C\ \{Q\}$, where J is known as the resource invariant, P as the precondition, and Q as the postcondition. Informally, these specifications say that if C is executed from an initial state satisfying $P * J$, then J will be satisfied throughout execution and the final state (if the command terminates) will satisfy $Q * J$. There is also an ownership reading attached to the specifications saying that the command 'owns' the state described by its precondition: the command can change it and can assume that no other parallel thread can change it. In contrast, the state described by J can be changed by other concurrently executing threads. The only guarantee is that it will always satisfy the resource invariant, J.

First, we have the so called structural rules from Hoare logic. The most important of these is the consequence rule:

$$\frac{P' \Rightarrow P \qquad J \vdash \{P\}\ C\ \{Q\} \qquad Q \Rightarrow Q'}{J \vdash \{P'\}\ C\ \{Q'\}} \qquad (\textsc{Conseq})$$

Next, we have the disjunction rule and the existential rules, which basically allow us to do a case split on the precondition.

$$\frac{J \vdash \{P_1\}\ C\ \{Q\} \qquad J \vdash \{P_2\}\ C\ \{Q\}}{J \vdash \{P_1 \vee P_2\}\ C\ \{Q\}} \qquad (\textsc{Disj})$$

$$\frac{J \vdash \{P\}\ C\ \{Q\} \qquad x \notin \mathsf{fv}(C, Q)}{J \vdash \{\exists x.\ P\}\ C\ \{Q\}} \qquad (\textsc{Ex})$$

Similarly, there is a conjunction rule that allows us to combine two proofs to derive the conjunction of the postconditions. For soundness purposes, however, this rule has to be restricted so that the resource invariant, J, is precise.

$$\frac{J \vdash \{P\}\ C\ \{Q_1\} \quad J \vdash \{P\}\ C\ \{Q_2\} \quad J \text{ precise}}{J \vdash \{P\}\ C\ \{Q_1 \wedge Q_2\}} \qquad (\textsc{Conj})$$

Next, we have the standard rules for skip, assume statements, sequential composition, non-deterministic choice, and loops. (Again, these rules are the same as in Hoare logic).

$$\frac{}{J \vdash \{P\}\ \mathbf{skip}\ \{P\}} \quad (\textsc{Skip})$$

$$\frac{x \notin \mathsf{fv}(J)}{J \vdash \{[E/x]P\}\ x := E\ \{P\}} \quad (\textsc{Assign})$$

$$\frac{}{J \vdash \{P\}\ \mathbf{assume}(B)\ \{P \wedge B\}} \quad (\textsc{Assume})$$

$$\frac{J \vdash \{P\}\ C_1\ \{Q\} \quad J \vdash \{Q\}\ C_2\ \{R\}}{J \vdash \{P\}\ C_1; C_2\ \{R\}} \quad (\textsc{Seq})$$

$$\frac{\begin{array}{c} J \vdash \{P\}\ C_1\ \{Q\} \\ J \vdash \{P\}\ C_2\ \{Q\} \end{array}}{J \vdash \{P\}\ C_1 \oplus C_2\ \{Q\}} \quad (\textsc{Choice})$$

$$\frac{J \vdash \{P\}\ C\ \{P\}}{J \vdash \{P\}\ C^*\ \{P\}} \quad (\textsc{Loop})$$

Note that from these rules, one can derive the following standard rules for conditionals and while-loops:

$$\frac{\begin{array}{c} J \vdash \{P \wedge B\}\ C_1\ \{Q\} \\ J \vdash \{P \wedge \neg B\}\ C_2\ \{Q\} \end{array}}{J \vdash \{P\}\ \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \{Q\}} \quad (\textsc{If})$$

$$\frac{J \vdash \{P \wedge B\}\ C\ \{P\}}{J \vdash \{P\}\ \mathbf{while}\ B\ \mathbf{do}\ C\ \{P \wedge \neg B\}} \quad (\textsc{While})$$

Next, we have a few rules for atomic accesses: reading, writing, allocation, and deallocation.

$$\frac{x \notin \mathsf{fv}(E, E', J)}{J \vdash \{E \mapsto E'\}\ x := [E]\ \{E \mapsto E' \wedge x = E'\}} \quad (\textsc{Read})$$

$$\frac{}{J \vdash \{E \mapsto -\}\ [E] := E'\ \{E \mapsto E'\}} \quad (\textsc{Write})$$

$$\frac{x \notin \mathsf{fv}(E, J)}{J \vdash \{\mathbf{emp}\}\ x{:=}\mathbf{alloc}(E)\ \{x \mapsto E\}} \quad (\textsc{Alloc})$$

$$\frac{}{J \vdash \{E \mapsto -\}\ \mathbf{dispose}(E)\ \{\mathbf{emp}\}} \quad \text{(Free)}$$

Note that Read and Write both require that the memory cell accessed is part of the precondition: this ensures that the cell is allocated (and hence, the access will be safe) and (from the yet to shown parallel composition rule) that no other thread is accessing it concurrently.

The parallel composition rule, Par, allows us to compose two threads in parallel if and only if their preconditions describe disjoint parts of the heap. This prevents data races on memory locations. The side-conditions ensure that there are also no data races on program variables—here, fv returns the set of free variables of a command or an assertion, whereas $\mathsf{wr}(C)$ returns the set of variables being assigned to by the command C.

$$\frac{\begin{array}{c} J \vdash \{P_1\}\ C_1\ \{Q_1\} \\ J \vdash \{P_2\}\ C_2\ \{Q_2\} \\ \mathsf{fv}(J, P_1, C_1, Q_1) \cap \mathsf{wr}(C_2) = \emptyset \\ \mathsf{fv}(J, P_2, C_2, Q_2) \cap \mathsf{wr}(C_1) = \emptyset \end{array}}{J \vdash \{P_1 * P_2\}\ C_1 \| C_2\ \{Q_1 * Q_2\}} \quad \text{(Par)}$$

The atomic command rule, Atom, allows the body of atomic blocks to use the resource invariant, J, and requires them to re-establish it at the postcondition.

$$\frac{\mathbf{emp} \vdash \{P * J\}\ C\ \{Q * J\}}{J \vdash \{P\}\ \mathbf{atomic}\ C\ \{Q\}} \quad \text{(Atom)}$$

Next, Share allows us at any time to extend the resource invariant by separatingly conjoining part of the local state, R.

$$\frac{J * R \vdash \{P\}\ C\ \{Q\}}{J \vdash \{P * R\}\ C\ \{Q * R\}} \quad \text{(Share)}$$

Finally, the Frame rule allows us to ignore part of the local state, the *frame* R, which is not used by the command, ensuring that R is still true at the postcondition.

$$\frac{\begin{array}{c} J \vdash \{P\}\ C\ \{Q\} \\ \mathsf{fv}(R) \cap \mathsf{wr}(C) = \emptyset \end{array}}{J \vdash \{P * R\}\ C\ \{Q * R\}} \quad \text{(Frame)}$$

4 The Meaning of CSL Judgments

First, let us introduce the following auxiliary predicate,

$$\mathsf{satU}(s,h,C,J) \stackrel{\text{def}}{=} (\mathbf{locked}(C) \wedge h = \emptyset) \vee (\neg\mathbf{locked}(C) \wedge [\![J]\!](s,h)),$$

stating that h satisfies the assertion J if the lock is free, and is empty in case the lock is held.

We define the semantics of CSL judgments in terms of an auxiliary predicate, $\mathsf{safe}_n(C,s,h,J,Q)$, stating that the command C executing with a stack, s, and a *local* heap, h, is safe with respect to the resource invariant J and the post-condition Q for up to n execution steps.

Definition 2 (Configuration Safety)**.**

$\mathsf{safe}_0(C,s,h,J,Q) \stackrel{\text{def}}{=} \mathsf{true}$

$\mathsf{safe}_{n+1}(C,s,h,J,Q) \stackrel{\text{def}}{=}$

$$\begin{array}{l}
\big(C = \mathbf{skip} \implies [\![Q]\!](s,h)\big) \\
\wedge \big(\nexists h_{\mathrm{J}}, h_{\mathrm{F}}.\ \ \mathsf{satU}(s,h_{\mathrm{J}},C,J) \wedge \langle C,s,h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \to \mathbf{abort}\big) \\
\wedge \left(\begin{array}{l}
\forall h_{\mathrm{J}}, h_{\mathrm{F}}, C', s', h'. \\
\ \mathsf{satU}(s,h_{\mathrm{J}},C,J) \wedge \langle C,s,h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \to \langle C',s',h'\rangle \\
\ \implies \exists h'', h'_{\mathrm{J}}.\ \ h' = h'' \uplus h'_{\mathrm{J}} \uplus h_{\mathrm{F}} \\
\qquad\qquad \wedge\, \mathsf{satU}(s',h'_{\mathrm{J}},C',J) \wedge \mathsf{safe}_n(C',s',h'',J,Q)
\end{array}\right)
\end{array}$$

A CSL judgment, $J \vdash \{P\}\ C\ \{Q\}$, simply says that the program C is safe with respect to J and Q for every initial local state satisfying the precondition, P, and for any number of step.

Definition 3. $J \vdash \{P\}\ C\ \{Q\}$ *iff* $\forall n,s,h.\ [\![P]\!](s,h) \implies \mathsf{safe}_n(C,s,h,J,Q)$.

Intuitively, any configuration is safe for zero steps. For $n+1$ steps, it must (*i*) satisfy the postcondition if it is a terminal configuration, (*ii*) not abort, and (*iii*) after any step, re-establish the resource invariant and be safe for another n steps. The number of steps merely ensures the definition is structurally decreasing.

In more detail, h is the part of the heap that is 'owned' by the command: the command can update h and no other command can access it in parallel. In conditions (*ii*) and (*iii*), h_{J} represents the part of the heap that is shared among threads, and must hence satisfy the resource invariant. So, condition (*iii*) ensures that after the transition a new such component, h'_{J}, can be found. Finally, h_{F} represents the remaining part of the heap owned by the rest of the system. In condition (*ii*), the command must not abort regardless of what that remaining part is. In condition (*iii*), the command must not change any part of the heap that could be owned by another thread. Therefore, h_{F} must be a subheap of the new heap h'.

5 Soundness Proof

We start with some basic –but important– properties of the semantics. In the following, let $[s \sim s']^X$ stand for $\forall x \in X.\ s(x) = s'(x)$ and $\overline{X}$ for the complement of set X.

Proposition 4. *If* $\langle C, s, h\rangle \to \langle C', s', h'\rangle$, *then* $\mathsf{fv}(C') \subseteq \mathsf{fv}(C)$, $\mathsf{wr}(C') \subseteq \mathsf{wr}(C)$, *and* $[s \sim s']^{\overline{\mathsf{wr}(C)}}$.

Proposition 5.
(i) If $[s \sim s']^{\mathsf{fv}(E)}$, *then* $[\![E]\!](s) = [\![E]\!](s')$.
(ii) If $[s \sim s']^{\mathsf{fv}(B)}$, *then* $[\![B]\!](s) = [\![B]\!](s')$.
(iii) If $[s \sim s']^{\mathsf{fv}(P)}$, *then* $[\![P]\!](s, h) = [\![P]\!](s', h)$.
(iv) If $[s \sim s']^{\mathsf{fv}(C)}$ *and* $\langle C, s, h\rangle \to \mathbf{abort}$, *then* $\langle C, s', h\rangle \to \mathbf{abort}$.
(v) If $X \supseteq \mathsf{fv}(C)$ *and* $[s \sim s']^X$ *and* $\langle C, s, h\rangle \to \langle C_1, s_1, h_1\rangle$, *then there exists* s_1' *such that* $\langle C, s', h\rangle \to \langle C_1', s_1', h\rangle$ *and* $[s_1 \sim s_1']^X$.

Now, consider Definition 2. By construction, safe is monotonic with respect to n: if a configuration is safe for a number of steps, n, it is also safe for a smaller number of steps, m. (This is proved by induction on m.)

Lemma 6. *If* $\mathsf{safe}_n(C, s, h, J, Q)$ *and* $m \leq n$, *then* $\mathsf{safe}_m(C, s, h, J, Q)$.

Further, as a corollary of Proposition 5, $\mathsf{safe}_n(C, s, h, J, Q)$ depends only on the values of variables that are mentioned in C, J, Q.

Lemma 7. *If* $\mathsf{safe}_n(C, s, h, J, Q)$ *and* $[s \sim s']^{\mathsf{fv}(C,J,Q)}$, *then* $\mathsf{safe}_n(C, s', h, J, Q)$.

The soundness theorem for CSL is the following:

Theorem 8 (CSL Soundness)**.** *All the rules shown in Section 2 are valid with respect to the* $J \vdash \{P\}\ C\ \{Q\}$ *definition.*

For brevity, we only show the proofs of the most interesting rules.

(Skip) The rule for **skip** follows immediately from the following lemma, whose proof is trivial because there are no transitions from **skip**.

Lemma 9. *If* $[\![Q]\!](s, h)$, *then* $\mathsf{safe}_n(\mathbf{skip}, s, h, J, Q)$.

(Atom) We first prove the following auxiliary lemma.

Lemma 10. *If* $\mathsf{safe}_n(C, s, h, \mathbf{emp}, J * Q)$, *then* $\mathsf{safe}_n(\mathbf{inatom}\ C, s, h, J, Q)$.

Proof. By induction on n. The base case is trivial. For the $n+1$ case, we assume the induction hypothesis and (*) $\mathsf{safe}_{n+1}(C, s, h, \mathbf{emp}, J * Q)$ and we have to show $\mathsf{safe}_{n+1}(\mathbf{inatom}\ C, s, h, J, Q)$. Unfolding the definition of safe, we have three conditions to show.
(i) is trivial as $\mathbf{inatom}\ C \neq \mathbf{skip}$.

(*ii*) By contradiction. Pick h_J and h_F such that $\mathsf{satU}(s, h_J, \mathbf{inatom}\ C, J)$ (i.e., $h_J = \emptyset$) and $\langle \mathbf{inatom}\ C, s, h \uplus h_J \uplus h_F\rangle \to \mathbf{abort}$. From the operational semantics, this entails that $\langle C, s, h \uplus h_J \uplus h_F\rangle \to \mathbf{abort}$, which contradicts (*).
(*iii*) Pick arbitrary h_J, h_F, C', s', h' such that $\mathsf{satU}(s, h_J, \mathbf{inatom}\ C, J)$ (i.e., $h_J = \emptyset$) and $\langle \mathbf{inatom}\ C, s, h \uplus h_J \uplus h_F\rangle \to \langle C', s', h'\rangle$. The operational semantics has two possible transitions for $\mathbf{inatom}\ C$.
Case (InAtomStep). $C' = \mathbf{inatom}\ C''$ and $\langle C, s, h \uplus h_J \uplus h_F\rangle \to \langle C'', s', h'\rangle$.

From (*), since $h_J = \emptyset$ implies $\mathsf{satU}(s, h_J, C, \mathbf{emp})$, we know that there exist h'' and h'_J such that $h' = h'' \uplus h'_J \uplus h_F$ and $\mathsf{satU}(s, h'_J, C', \mathbf{emp})$ (i.e., $h'_J = \emptyset$), and $\mathsf{safe}_n(C'', s, h'', \mathbf{emp}, J * Q)$.

Therefore, from the ind. hyp., we also get $\mathsf{safe}_n(\mathbf{inatom}\ C'', s, h'', J, Q)$, as required.
Case (InAtomEnd). $C = C'' = \mathbf{skip}$, $s' = s$, and $h' = h \uplus h_F$.

From (*), we know that $[\![J * Q]\!](s, h)$; so there exist h_1 and h_2 such that $h = h_1 \uplus h_2$ and $[\![J]\!](s, h_1)$ and $[\![Q]\!](s, h_2)$.

Therefore, we pick as witnesses $h'' := h_2$ and $h'_J := h_1$, and use Lemma 9 to get $\mathsf{safe}_n(\mathbf{skip}, s, h_2, J, Q)$, thereby completing the proof. □

The main lemma for atomic commands is as follows:

Lemma 11. *If* $\mathbf{emp} \vdash \{P * J\}\ C\ \{Q * J\}$, *then* $J \vdash \{P\}\ \mathbf{atomic}\ C\ \{Q\}$.

Proof. Assume (*) $\mathbf{emp} \vdash \{P * J\}\ C\ \{Q * J\}$, and pick arbitrary $[\![P]\!](s, h)$ and n. We have to show that $\mathsf{safe}_n(\mathbf{atomic}\ C, s, h, J, Q)$. If $n = 0$, this is trivial; so consider $n = m + 1$. Condition (*i*) is trivial as $\mathbf{atomic}\ C \neq \mathbf{skip}$. Condition (*ii*) is trivial as $\neg\langle \mathbf{atomic}\ C, s, \_\rangle \to \mathbf{abort}$.
(*iii*) Pick arbitrary h_J, h_F, C', s', h' such that $\mathsf{satU}(s, h_J, \mathbf{atomic}\ C, J)$ (i.e., $[\![J]\!](s, h_J)$) and $\langle \mathbf{atomic}\ C, s, h \uplus h_J \uplus h_F\rangle \to \langle C', s', h'\rangle$.

The operational semantics has only one possible transition for $\mathbf{atomic}\ C$: $C' = \mathbf{inatom}\ C$, $s' = s$ and $h' = h \uplus h_J \uplus h_F$. Therefore, picking $h'' := h \uplus h_J$ and $h'_J := \emptyset$, it suffices to show $\mathsf{safe}_n(\mathbf{inatom}\ C, s, h \uplus h_J, J, Q)$. This follows immediately from (*) and Lemma 10. □

(Par) For parallel composition, we need the following auxiliary lemma:

Lemma 12. *If* $\mathsf{safe}_n(C_1, s, h_1, J, Q_1)$, $\mathsf{safe}_n(C_2, s, h_2, J, Q_1)$, $h_1 \uplus h_2$ *is defined,* $\mathsf{fv}(J, C_1, Q_1) \cap \mathsf{wr}(C_2) = \emptyset$, *and* $\mathsf{fv}(J, C_2, Q_2) \cap \mathsf{wr}(C_1) = \emptyset$, *then* $\mathsf{safe}_n(C_1 \| C_2, s, h_1 \uplus h_2, J, Q_1 * Q_2)$.

Proof. By induction on n. In the inductive step, we know $IH(n) \stackrel{\text{def}}{=}$

$$\begin{aligned}\forall C_1, h_1, C_2, h_2.\ &\mathsf{safe}_n(C_1, s, h_1, J, Q_1) \wedge \mathsf{safe}_n(C_2, s, h_2, J, Q_1) \wedge \mathsf{def}(h_1 \uplus h_2) \\ &\wedge \mathsf{fv}(J, C_1, Q_1) \cap \mathsf{wr}(C_2) = \emptyset \wedge \mathsf{fv}(J, C_2, Q_2) \cap \mathsf{wr}(C_1) = \emptyset \\ \implies\ &\mathsf{safe}_n(C_1 \| C_2, s, h_1 \uplus h_2, J, Q_1 * Q_2)\end{aligned}$$

and we have to show $IH(n+1)$. So, pick arbitrary C_1, h_1, C_2, h_2 and assume (1) $\mathsf{safe}_{n+1}(C_1, s, h_1, J, Q_1)$, (2) $\mathsf{safe}_{n+1}(C_2, s, h_2, J, Q_2)$, (3) $\mathsf{def}(h_1 \uplus h_2)$ and (4) the variable side-conditions, and try to show $\mathsf{safe}_{n+1}(C_1 \| C_2, s, h_1 \uplus h_2, J, Q_1 * Q_2)$. Condition (i) is trivial.
(ii) If $\langle C_1 \| C_2, s, h_1 \uplus h_2 \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \mathbf{abort}$, then according to the semantics $\langle C_1, s, h_1 \uplus h_2 \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \mathbf{abort}$ or $\langle C_2, s, h_1 \uplus h_2 \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \mathbf{abort}$, contradicting our assumptions (1) and (2).
(iii) Pick arbitrary h_J, h_F, C', s', h' such that $\mathsf{satU}(s, h_\mathrm{J}, C_1 \| C_2, J)$ and $\langle C_1 \| C_2, s, h_1 \uplus h_2 \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \langle C', s', h' \rangle$. The operational semantics has three possible transitions for $C_1 \| C_2$.
Case (PAR1). $C' = C_1' \| C_2$ and $\langle C_1, s, h_1 \uplus h_2 \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \langle C_1', s', h' \rangle$ and $\neg \mathbf{locked}(C_2)$.

From (1), choosing as frame $h_2 \uplus h_\mathrm{F}$, there exist h_1' and h_J' such that $h' = h_1' \uplus h_\mathrm{J}' \uplus (h_2 \uplus h_\mathrm{F})$, $\mathsf{satU}(s', h_\mathrm{J}', C_1', J)$, and $\mathsf{safe}_n(C_1', s', h_1', J, Q_1)$.

Since $\neg \mathbf{locked}(C_2)$, we also have $\mathsf{satU}(s', h_\mathrm{J}', C_1' \| C_2, J)$.

From (2) and Proposition 6, we have $\mathsf{safe}_n(C_2, s, h_2, J, Q_2)$. Then, from Propositions 7 and 4, and assumption (4), we have $\mathsf{safe}_n(C_2, s', h_2, J, Q_2)$. Also, from Proposition 4 and (4), $\mathsf{fv}(C_1', Q_1) \cap \mathsf{wr}(C_2) = \emptyset$ and $\mathsf{fv}(C_2, Q_2) \cap \mathsf{wr}(C_1') = \emptyset$, and hence from $IH(n)$, $\mathsf{safe}_n(C_1' \| C_2, s', h_1' \uplus h_2, J, Q_1 * Q_2)$.
Case (PAR2). This case is completely symmetric.
Case (PAR3). $C_1 = C_2 = C' = \mathbf{skip}$, $h' = h_1 \uplus h_2 \uplus h_\mathrm{J} \uplus h_\mathrm{F}$.

From (1) and (2), unfolding the definition of safe, we have that $[\![Q_1]\!](s, h_1)$ and $[\![Q_2]\!](s, h_2)$. So, $[\![Q_1 * Q_2]\!](s, h_1 \uplus h_2)$. Therefore, from Lemma 9, we derive $\mathsf{safe}_n(\mathbf{skip}, s, h_1 \uplus h_2)$, as required. □

(FRAME) The frame rule is a cut-down version of the parallel composition rule. It follows directly from the following lemma:

Lemma 13. *If* $\mathsf{safe}_n(C, s, h, J, Q)$, $\mathsf{fv}(R) \cap \mathsf{wr}(C) = \emptyset$, $h \uplus h_\mathrm{R}$ *is defined, and* $[\![R]\!](s, h_\mathrm{R})$, *then* $\mathsf{safe}_n(C, s, h \uplus h_\mathrm{R}, J, Q * R)$.

Proof. By induction on n. The base case is trivial. For the inductive step, assume (*) $\mathsf{safe}_{n+1}(C, s, h, J, Q)$, (†) $\mathsf{fv}(R) \cap \mathsf{wr}(C) = \emptyset$, and (‡) $[\![R]\!](s, h_\mathrm{R})$. Now, we have to prove $\mathsf{safe}_{n+1}(C, s, h \uplus h_\mathrm{R}, J, Q * R)$.
(i) From (*), we get $[\![Q]\!]s, h$ and so, using (‡), $[\![Q * R]\!](s, h \uplus h_\mathrm{R})$.
(ii) We argue by contradiction. Pick h_J and h_F such that $\mathsf{satU}(s, h_\mathrm{J}, C, J)$ and $\langle C, s, h \uplus h_\mathrm{R} \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \mathbf{abort}$. But this contradicts (*) for $h_\mathrm{F} := (h_\mathrm{R} \uplus h_\mathrm{F})$.
(iii) If $\langle C, s, h \uplus h_\mathrm{R} \uplus h_\mathrm{J} \uplus h_\mathrm{F} \rangle \to \langle C', s', h' \rangle$ and $\mathsf{satU}(s, h_\mathrm{J}, C, J)$, then from (*), there exist h'', h_J' such that $h' = h'' \uplus h_\mathrm{J}' \uplus (h_\mathrm{R} \uplus h_\mathrm{F})$ and $\mathsf{satU}(s', h_\mathrm{J}', C', J)$ and $\mathsf{safe}_n(C', s', h'', J, Q)$. Now, from (†), (‡), Prop. 4 and 5, we get $[\![R]\!](s', h_\mathrm{R})$ and $\mathsf{fv}(R) \cap \mathsf{wr}(C') = \emptyset$. Therefore, from the induction hypothesis, we conclude $\mathsf{safe}_n(C', s', h' \uplus h_\mathrm{R}, J, Q * R)$, as required. □

(SHARE) We need the following lemma, which is similar to the previous one.

Lemma 14. *If* $\mathsf{safe}_n(C, s, h, J{*}R, Q)$*,* $h{\uplus}h_{\mathrm{R}}$ *is defined, and* $\mathsf{satU}(s, h_{\mathrm{R}}, C, R)$*, then* $\mathsf{safe}_n(C, s, h \uplus h_{\mathrm{R}}, J, Q * R)$*.*

Proof. As an exercise. □

(Conseq, Disj, Ex) The proofs of these rules are trivial. For the consequence rule, we need the following lemma, whose proof is by a trivial induction on n.

Lemma 15. *If* $\mathsf{safe}_n(C, s, h, J, Q)$ *and* $Q \Rightarrow Q'$ *then* $\mathsf{safe}_n(C, s, h, J, Q')$*.*

(Seq) Here we prove the following lemma:

Lemma 16. *If* $\mathsf{safe}_n(C_1, s, h, J, Q)$ *and* $J \vdash \{Q\}\ C_2\ \{R\}$*, then* $\mathsf{safe}_n(C_1; C_2, s, h, J, R)$*.*

Proof. Assume $(*)$ $J \vdash \{Q\}\ C_2\ \{R\}$ and prove

$$IH(n) \stackrel{\text{def}}{=} \forall C_1, s, h.\ \mathsf{safe}_n(C_1, s, h, J, Q) \Rightarrow \mathsf{safe}_n(C_1; C_2, s, h, J, R)$$

by induction on n. For the inductive step, assume $IH(n)$, pick arbitrary C_1, h and assume $(\dagger)$ $\mathsf{safe}_{n+1}(C_1, s, h, J, Q)$.
Now, we have to show $\mathsf{safe}_{n+1}(C_1; C_2, s, h, J, R)$.
(i) Trivial, as $(C_1; C_2) \neq \mathbf{skip}$.
(ii) Trivial, since in the operational semantics $\langle C_1; C_2, s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \rightarrow \mathbf{abort}$ if and only if $\langle C_1, s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \rightarrow \mathbf{abort}$.
(iii) Pick arbitrary C', h_{J}, h_{F}, s', h' such that $s, h_{\mathrm{J}} \models J$, $(h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}})$ is defined, and $\langle C_1; C_2, s', h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \rightarrow \langle C', s', h'\rangle$. There are two possible transitions for $C_1; C_2$.
Case (Seq1). $C_1 = \mathbf{skip}$, $C' = C_2$ and $h' = h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}$. From $(\dagger)$, we have $[\![Q]\!](s, h)$. Thus, from $(*)$, $\mathsf{safe}_n(C_2, s, h, J, R)$ as required.
Case (Seq2). $C' = C_1'; C_2$ and $\langle C_1, s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \rightarrow \langle C_1', s', h'\rangle$. From $(\dagger)$, there exist h'' and h_{J}' such that $h' = h'' \uplus h_{\mathrm{J}}' \uplus h_{\mathrm{F}}$ and $\mathsf{satU}(s', h_{\mathrm{J}}', C_1', J)$ and $\mathsf{safe}_n(C_1, s', h'', J, Q)$. Thus, from the induction hypothesis, we conclude that $\mathsf{safe}_n(C_1; C_2, s', h'', J, R)$, as required. □

(Read) For memory reads, we have the following proof:

Proof. Assume $(\dagger)$ $x \notin \mathsf{fv}(E, E', J)$. Also assume $[\![E \mapsto E']\!](s, h)$, from which we can decude that $(*)$ $\mathbf{dom}(h) = \{[\![E]\!](s)\}$ and $h([\![E]\!](s)) = \{[\![E']\!](s)\}$. We have to show that $\mathsf{safe}_n(x := [E], s, h, J, E \mapsto E' \wedge x = E')$. If $n = 0$, this is trivial. Now, if $n = m + 1$, condition (i) is trivial.
(ii) If $\langle x := [E], s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \rightarrow \mathbf{abort}$, then $[\![E]\!](s) \notin \mathbf{dom}(h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}})$, thereby contradicting $(*)$.
(iii) If $\langle x := [E], s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}\rangle \rightarrow \langle C', s', h'\rangle$, then by the only applicable rule (Read), $C' = \mathbf{skip}$, $s' = s[x := [\![E]\!](s)]$ and $h' = h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}$.

By $(*)$ and $(\dagger)$, using Prop. 5, we deduce that $(*)$ $\mathbf{dom}(h) = \{[\![E]\!](s')\}$ and $h([\![E]\!](s')) = \{[\![E']\!](s')\}$. Therefore, $[\![E \mapsto E' \wedge x = E']\!](s', h')$ and, from Lemma 9, $\mathsf{safe}_n(C', s', h', J, E \mapsto E' \wedge x = E')$. □

(WRITE) For the memory writes, the following:

Proof. Assume $[\![E \mapsto -]\!](s,h)$, from which we can decude that $(*)$ $\mathbf{dom}(h) = \{[\![E]\!](s)\}$. We have to show that $\mathsf{safe}_n([E] := E', s, h, J, E \mapsto E')$. If $n = 0$, this is trivial. Now, if $n = m + 1$, condition (i) is trivial.
(ii) If $\langle [E] := E', s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}} \rangle \to \mathbf{abort}$, then by the only applicable rule (WRIA), $[\![E]\!](s) \notin \mathbf{dom}(h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}})$, thereby contradicting $(*)$.
(iii) If $\langle [E] := E', s, h \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}} \rangle \to \langle C', s', h' \rangle$, then by the only applicable rule (WRI), $C' = \mathbf{skip}$, $s' = s$ and $h' = h[[\![E]\!](s) := [\![E]\!](s)] \uplus h_{\mathrm{J}} \uplus h_{\mathrm{F}}$.

Hence, using $(*)$ we get $[\![E \mapsto E']\!](s', h')$ and, from Lemma 9, we also get $\mathsf{safe}_n(C', s', h', J, E \mapsto E')$, as required. □

(CONJ) Now consider the conjunction rule. Its soundness rests upon the validity of the following implication:

$$\mathsf{safe}_n(C, s, h, J, Q_1) \wedge \mathsf{safe}_n(C, s, h, J, Q_2) \implies \mathsf{safe}_n(C, s, h, J, Q_1 \wedge Q_2)\,.$$

Naturally, one would expect to prove this implication by induction on n with an induction hypothesis quantifying over all C and h. The base case is trivial; so consider the $n+1$ case. The first two subcases are easy; so consider subcase (iii). From the first assumption, we know that there exist h^1 and h^1_J such that $h' = h^1 \uplus h^1_J$ and $\mathsf{satU}(s', h^1_J, C', J)$ and $\mathsf{safe}_n(C', s', h^1, J, Q_1)$. Similarly, from the first assumption, there exist h^2 and h^2_J such that $h' = h^2 \uplus h^2_J$ and $\mathsf{satU}(s', h^2_J, C', J)$ and $\mathsf{safe}_n(C', s', h^2, J, Q_2)$, but, in general, we do not know that $h^1 = h^2$ which would allow us to complete the proof. Since, however, J must be precise, then (from Definition 1) $h^1_J = h^2_J$, and since $\uplus$ is cancellative, we also have $h^1 = h^2$ and the result follows by applying the induction hypothesis. □

Other Rules. The other proof rules for the sequential commands (i.e., ASSUME, ASSIGN, ALLOC, FREE, NONDET and LOOP) are similar to the ones seen already.

6 Precision and the Reynolds Counterexample

To show why the "precise J" condition on the conjunction rule is necessary, John Reynolds came up with a counterexample showing that without this condition, the system is unsound. Consider the following two derivations:

$$\begin{array}{ll}
& \mathbf{emp} \vdash \{\mathbf{emp} * \mathsf{true}\}\ \mathbf{skip}\ \{\mathbf{emp} * \mathsf{true}\} & \text{— (Skip)} \\
\Longrightarrow & \mathsf{true} \vdash \{\mathbf{emp}\}\ \mathbf{atomic\ skip}\ \{\mathbf{emp}\} & \text{— (Atom)} \\
\Longrightarrow & \mathsf{true} \vdash \{\mathbf{emp} * (1 \mapsto 2)\}\ \mathbf{atomic\ skip}\ \{\mathbf{emp} * (1 \mapsto 2)\} & \text{— (Frame)} \\
\Longleftrightarrow & \mathsf{true} \vdash \{1 \mapsto 2\}\ \mathbf{atomic\ skip}\ \{1 \mapsto 2\}
\end{array}$$

$$\begin{array}{ll}
& \mathbf{emp} \vdash \{\mathsf{true}\}\ \mathbf{skip}\ \{\mathsf{true}\} & \text{— (Skip)} \\
\Longrightarrow & \mathbf{emp} \vdash \{(1 \mapsto 2) * \mathsf{true}\}\ \mathbf{skip}\ \{\mathbf{emp} * \mathsf{true}\} & \text{— (Conseq)} \\
\Longrightarrow & \mathsf{true} \vdash \{1 \mapsto 2\}\ \mathbf{atomic\ skip}\ \{\mathbf{emp}\} & \text{— (Atom)}
\end{array}$$

If were allowed to use the conjunction rule, we could derive

$$\mathsf{true} \vdash \{1 \mapsto 2\}\ \mathbf{atomic\ skip}\ \{1 \mapsto 2 \wedge \mathbf{emp}\}$$

which is clearly wrong, as $(1 \mapsto 2) \wedge \mathbf{emp}$ is false.

The problem really is that the resource invariant true is not precise, which allows the two derivations to put choose a different splitting between the local state owned by the command's postcondition and the shared state described by the resource invariant.

7 Fractional Permissions

Generally, CSL does not allow us to reason about programs with data races (i.e., with concurrent accesses to the same location or variable, where at least one of the accesses is a write access).

In doing so, however, it also forbids us to reason about some perfectly fine race-free programs, where the concurrent accesses are all reads. For example, there is no way to derive the following triple

$$\mathbf{emp} \vdash \{x \mapsto 5\}\ a := [x] \parallel b := [x]\ \{x \mapsto 5 \wedge a = 5 \wedge b = 5\}$$

with the rules we have seen so far, because we would have to split the permission to access x (namely, the assertion $x \mapsto 5$) to the two threads.

Fractional permissions allow us to do exactly this. We extend the language of assertions with an assertion

$$P ::= \ldots \mid E \overset{F}{\mapsto} E'$$

where F is an expression representing a number in the range $0 < F \leq 1$, where 1 represents a full permission, and fractions less than 1 are partial

(read-only) permissions. In this extension, the normal points-to assertion $E \mapsto E'$ can be seen as an abbreviation for $E \overset{1}{\mapsto} E'$.

Assuming $0 < F_1 \leq 1$ and $0 < F_2 \leq 2$, we have the following rule for spliting a fractional permission.

$$E \overset{F_1}{\mapsto} E_1 * E \overset{F_2}{\mapsto} E_2 \iff E \overset{F_1+F_2}{\longmapsto} E_1 \wedge E_1 = E_2 \wedge F_1 + F_2 \leq 1$$

The read rule can be relaxed to require only a partial permission.

$$\frac{x \notin \mathsf{fv}(E, E', F, J)}{J \vdash \{E \overset{F}{\mapsto} E'\}\ x := [E]\ \{E \overset{F}{\mapsto} E' \wedge x = E''\}} \quad (\textsc{ReadFrac})$$

References

[1] O'Hearn, P. W., *Resources, concurrency and local reasoning*, Theor. Comput. Sci. **375** (2007), pp. 271–307.

[2] Vafeiadis, V., *Concurrent separation logic and operational semantics*, ENTCS 276, pp. 335-351. Elsevier (2011)