Concurrent Separation Logic

Beta Ziliani

Max Planck Institute for Software Systems (MPI-SWS)

May 23, 2011

$$\frac{\{P \wedge B\}S\{Q\}}{\{P\}\textbf{await } B \textbf{ then } S\{Q\}}$$

$$\frac{\{P_1\}S_1\{Q_1\} \ldots \{P_n\}S_n\{Q_n\} \text{ are } \textit{interference-free}}{\{P_1 \wedge \ldots \wedge P_n\}S_1 \parallel \ldots \parallel S_n\{Q_1 \wedge \ldots \wedge Q_n\}}$$

Interference-freedom is hard to prove and not composable.

Even more previously, but on life

Hoare (1972)

Symmetric Parallel Rule

$$\frac{\{P_1\}S_1\{Q_1\} \ldots \{P_n\}S_n\{Q_n\} \text{ are } \textit{disjoint}}{\{P_1 \wedge \ldots \wedge P_n\}S_1 \parallel \ldots \parallel S_n\{Q_1 \wedge \ldots \wedge Q_n\}}$$

Where *disjointness* is "they work with different data spaces".
For shared *resources*, the following construct is added:

$$\textbf{with } \overbrace{r}^{\text{resource}} \textbf{ do } \underbrace{S}_{\text{critical region}}$$

Hoare, *Towards a Theory of Parallel Programming*

Walking over eggs

- These systems statically ensure *cautiousness*.
- Shared variables accessed only in critical regions.
- Most concurrent systems are *daring*.
- Semaphores example:

$$\begin{array}{l} \mathsf{P}(s) \\ \text{critical code} \\ \mathsf{V}(s) \end{array}$$

- Rely/Guarantee allows this kind of interactions.

Assertions P, Q extended with

- $x \mapsto v$
- $P_1 * P_2$
- In $\{P\}S\{Q\}$:
  "S can only access locations in the domain of P"
- *Frame rule*:

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

Key ideas

- Parallel rule:

$$\frac{\{P_1\}S_1\{Q_1\}\ldots\{P_n\}S_n\{Q_n\}}{\{P_1 * \ldots * P_n\}S_1 \parallel \ldots \parallel S_n\{Q_1 * \ldots * Q_n\}}\ \textit{conditions}$$

- Resource invariants. From S. Owicki and D. Gries. *Verifying properties of parallel programs: An axiomatic approach.*

  Through the talk Owicki/Gries (O/G) mean this work.

Key ideas

- In $\{P\}S\{Q\}$:
  "S owns locations in the domain of P"
- *Ownership Hypothesis*: A code fragment can access only those portions of state that it owns.
- *Separation Property*: At any time, the state can be partitioned into that owned by each process and each grouping of mutual exclusion.
- The proof system will ensure this property.

Separation Conjunction
Separating a lot!

Par rule:

$$\frac{\{P_1\}S_1\{Q_1\}\ldots\{P_n\}S_n\{Q_n\}}{\{P_1 * \ldots * P_n\}S_1 \parallel \ldots \parallel S_n\{Q_1 * \ldots * Q_n\}}$$

where $\mathbf{modify}(S_i) \cap \mathbf{FV}(P_j, S_j, Q_j) = \emptyset$ for $i \neq j$

- It is restrictive.
- But has its uses.

$\{array(a, i, j)\}$

procedure msort(a, i, j)

$m := (i + j)/2;$

if $i < j$ **then**

$\big(\text{msort}(a, i, m) \parallel \text{msort}(a, m + 1, j)\big);$

$\text{merge}(a, i, m + 1, j);$

$\{sorted(a, i, j)\}$

We can split the array:

$$\{array(a, i, m) * array(a, m+1, j)\}$$
$$\{array(a, i, m)\} \qquad \{array(a, m+1, j)\}$$
$$(\text{msort}(a, i, m) \quad \| \quad \text{msort}(a, m+1, j));$$
$$\{sorted(a, i, m)\} \qquad \{sorted(a, m+1, j)\}$$
$$\text{merge}(a, i, m+1, j)$$
$$\{sorted(a, i, j)\}$$

Language with CCRs (O/G)

$$
\begin{array}{l}
\mathit{init}; \\
\textbf{resource } r_1(\text{variable list}) \ldots r_m(\text{variable list}) \\
C_1 \parallel \ldots \parallel C_n
\end{array}
$$

We said x *belongs to* r if it's in r's list.

$$
\begin{array}{rl}
C ::= & \textbf{skip} \mid x := E \mid x := [E] \mid [E] := E' \mid C_1; C_2 \\
& \mid i := \textbf{cons}(E_0, \ldots, E_n) \mid \textbf{dispose } E \\
& \mid \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2 \mid \textbf{while } B \textbf{ do } C \\
& \mid \textbf{with } r \textbf{ when } B \textbf{ do } C
\end{array}
$$

Brookes uses a less restricted syntax.

Well-formedness

Restriction taken from Owicki/Gries to avoid interference:

1. a variable belongs *at most* to one resource;
2. if x belongs to r, it can only appear within a parallel process inside a critical region for r;
3. if x is changed in one process, it cannot appear in another unless it belongs to a resource.

In the presence of pointers these rules are not enough! Consider

$$[x] := 3 \parallel [y] := 4$$

Separation conjunction will help us here.

Proof rules

- Par rule not requiring looking at other's code.
- Each resource r_i has a *resource invariant* I_i satisfying:
  - any command $x := \ldots$ where $x \in \mathbf{FV}(I_i)$ must occur within a critical region for r_i.
    In O/G they ask x to belong to r_i.
  - Each I_i should be "precise" (to be defined).
- Rule for a program:

$$\frac{\{P\}\mathit{init}\{\boxed{I_1 * \ldots * I_m} * P'\} \quad \{P'\}C_1 \parallel \ldots \parallel C_n\{Q\}}{\begin{array}{l}\{P\}\\ \mathit{init};\\ \textbf{resource } r_1(\text{variable list}) \ldots r_m(\text{variable list})\\ C_1 \parallel \ldots \parallel C_n\\ \{\boxed{I_1 * \ldots * I_m} * Q\}\end{array}}$$

Proof rules

- Rule for critical region:

$$\frac{\{(P * I_i) \wedge B\} C \{Q * I_i\}}{\{P\}\textbf{with } r_i \textbf{ when } B \textbf{ do } C\{Q\}}$$

  No other process modifies variables free in P or Q.

- Brookes's rules:

$$\frac{\boxed{\Gamma, r_i(X) : I_i \vdash} \{P\} C \{Q\}}{\{P * I_i\}\textbf{resource } r_i \textbf{ in } C\{Q * I_i\}}$$

  if $r \notin \textbf{dom}(\Gamma)$, $X \cap \textbf{owned}(\Gamma) = \emptyset$, $\textbf{free}(I_i) \cap \textbf{owned}(\Gamma) = \emptyset$, and I_i is precise. Where does X comes from?

- Rule for erasing temporal variables.

- Key point: there is “no interference from the outside”.
- Process communication at explicit synchronization points.
- Dijkstra: “apart from the (rare) moments of explicit intercommunication, the individual processes are to be regarded as completely independent from each other”.
- Ownership Hypothesis is essential.
- *Well specified processes mind their own business.*

$$
\begin{array}{l}
\vdots \\
\{\mathbf{emp}\} \\
\mathsf{P}(\mathit{free}); \\
\{10 \mapsto -\} \\
[10] := m; \\
\{10 \mapsto -\} \\
\mathsf{V}(\mathit{busy}); \\
\{\mathbf{emp}\} \\
\vdots
\end{array}
\quad \| \quad
\begin{array}{l}
\vdots \\
\{\mathbf{emp}\} \\
\mathsf{P}(\mathit{busy}); \\
\{10 \mapsto -\} \\
n := [10]; \\
\{10 \mapsto -\} \\
\mathsf{V}(\mathit{free}); \\
\{\mathbf{emp}\} \\
\vdots
\end{array}
$$

$$\mathsf{P}(s) = \textbf{with } s \textbf{ when } s > 0 \textbf{ do } s := s - 1$$
$$\mathsf{V}(s) = \textbf{with } s \textbf{ when true do } s := s + 1$$

$$I_s = (s = 0 \wedge \textbf{emp}) \vee (s = 1 \wedge 10 \mapsto -)$$

$$\frac{\{(A * I_s) \wedge s > 0\} s := s - 1 \{A' * I_s\}}{\{A\}\mathsf{P}(s)\{A'\}}$$

$$\frac{\{A * I_s\} s := s + 1 \{A' * I_s\}}{\{A\}\mathsf{V}(s)\{A'\}}$$

$$
\begin{array}{l}
\vdots \\
\{\mathbf{emp}\} \\
\mathsf{P}(\mathit{free}); \\
\{10 \mapsto -\} \\
[10] := m; \\
\{10 \mapsto -\} \\
\mathsf{V}(\mathit{busy}); \\
\{\mathbf{emp}\} \\
\vdots
\end{array}
\quad \| \quad
\begin{array}{l}
\vdots \\
\{\mathbf{emp}\} \\
\mathsf{P}(\mathit{busy}); \\
\{10 \mapsto -\} \\
n := [10]; \\
\{10 \mapsto -\} \\
\mathsf{V}(\mathit{free}); \\
\{\mathbf{emp}\} \\
\vdots
\end{array}
$$

$$\frac{\{(A * I_s) \wedge s > 0\} s := s - 1 \{A' * I_s\}}{\{A\} \mathrm{P}(s) \{A'\}}$$

$$\{\big(\overbrace{\mathbf{emp}}^{A} * \big(\overbrace{(free = 0 \wedge \mathbf{emp}) \vee (free = 1 \wedge 10 \mapsto -)}^{I_{free}}\big)\big) \wedge free > 0\}$$
$$\{free = 1 \wedge 10 \mapsto -\}$$
$$free := free - 1$$
$$\{free = 0 \wedge 10 \mapsto -\}$$
$$\{(free = 0 \wedge \mathbf{emp}) * 10 \mapsto -\}$$
$$\{\big((free = 0 \wedge \mathbf{emp}) \vee (free = 1 \wedge 10 \mapsto -)\big) * \underbrace{10 \mapsto -}_{A'}\}$$

$$\{10 \mapsto -\}$$
$$\{10 \mapsto - * \mathbf{emp}\}$$
$$free := 1;\ \ busy := 0;$$
$$\{(free = 1 \wedge 10 \mapsto -) * (busy = 0 \wedge \mathbf{emp})\}$$
$$\{\big((free = 1 \wedge 10 \mapsto -) \vee (free = 0 \wedge \mathbf{emp})\big) *$$
$$\quad * \big((busy = 1 \wedge 10 \mapsto -) \vee (busy = 0 \wedge \mathbf{emp})\big)\}$$
$$\{I_{free} * I_{busy}\}$$

What if we set both to 1?

Ownership transfer

$$prog = \ldots; \left(\begin{array}{lll} x := \mathbf{cons}(a, b); & \| & \mathbf{get}(y); \\ \mathbf{put}(x) & & use(y); \\ & & \mathbf{dispose}\,(y); \end{array} \right)$$

Where

$$\mathbf{put}(x) \triangleq \mathbf{with}\ buf\ \mathbf{when}\ \neg full\ \mathbf{do}\ c := x; full := \mathbf{true}$$

$$\mathbf{get}(y) \triangleq \mathbf{with}\ buf\ \mathbf{when}\ full\ \mathbf{do}\ y := c; full := \mathbf{false}$$

- Prove that, either x is used and deallocated by the receiver,
- or it is not used (nor deallocated) by the receiver.
- As it should, it cannot prove x can be deallocated by the two processes.

Logical garbage

- We'd like to prove: $\{\mathbf{emp}\}prog\{\mathbf{emp}\}$
- But we end up proving: $\{\mathbf{emp}\}prog\{I_{buf}\}$
- Where $I_{buf} = (full \wedge c \mapsto -, -) \vee (\neg full \wedge \mathbf{emp})$
- But we know $\neg full$, but we cannot leak it (soundness of **with**).
- Remember:

$$\frac{\{(P * I_i) \wedge B\}C\{Q * I_i\}}{\{P\}\mathbf{with}\ r_i\ \mathbf{when}\ B\ \mathbf{do}\ C\{Q\}}$$

  No other process modifies variables free in P or Q.
- (We use the magic address 10 before because of this.)
- Who can save us now?

Super-semaphore!

Super-semaphore!
Idea

- Add logical variables *start* and *finish*.

$$I_{buf} = \begin{array}{l} (\neg full \wedge start \wedge \neg finish \wedge \mathbf{emp}) \\ \vee (full \wedge \neg start \wedge \neg finish \wedge c \mapsto -, -) \\ \vee (\neg full \wedge \neg start \wedge finish \wedge \mathbf{emp}) \end{array}$$

- *start* modified by **put**, *finish* modified by **get**.
- We can leak them out!

$$\{\mathbf{emp}\}\, prog' \,\{I_{buf} * (\mathbf{emp} \wedge \neg start \wedge finish)\}$$

- Therefore getting

$$\{\mathbf{emp}\}\, prog' \,\{\mathbf{emp}\}$$

- And then we erase them!

$$\{\mathbf{emp}\}\, prog \,\{\mathbf{emp}\}$$

Memory allocation

$$\mathit{list}\ x \triangleq (x = \mathbf{nil} \wedge \mathbf{emp}) \vee (\exists y.x \mapsto -, y * \mathit{list}\ y)$$

$$I_m = \mathit{list}\ f$$

$$\mathbf{alloc}(x, a, b) \triangleq \mathbf{with}\ m\ \mathbf{when\ true\ do}$$
$$\mathbf{if}\ f = \mathbf{nil}\ \mathbf{then}\ x := \mathbf{cons}(a, b)$$
$$\mathbf{else}\ x := f; f := x.2; x.1 := a; x.2 := b$$

$$\mathbf{dealloc}(y) \triangleq \mathbf{with}\ m\ \mathbf{when\ true\ do}\ y.2 := f; f := y$$

Proved under invariant $\mathit{list}\ f$:

$$\{\mathbf{emp}\}\mathbf{alloc}(x, a, b)\{x \mapsto a, b\}$$

$$\{y \mapsto -, -\}\mathbf{dealloc}(y)\{\mathbf{emp}\}$$

Composition

- We can replace **cons** and **dispose** by **alloc** and **dealloc** in our *prog* example:

$$prog = \ldots; \left(\begin{array}{lcl} \textbf{alloc}(x, a, b); & \| & \textbf{get}(y); \\ \textbf{put}(x) & & use(y); \\ & & \textbf{dealloc}(y); \end{array} \right)$$

- Initialization requires the setup of *both* invariants:

$$full := \textbf{false};$$
$$\textbf{resource}\ buf(c, full), m(f)$$

- Once proved both invariants I_{buf} and I_m are set, the rest of the proof is maintained.

Composition

- Brookes's is *really* composable... (unified language).
- Key idea in rule for CCR:

$$\frac{\{(P * I_i) \wedge B\}C\{Q * I_i\}}{\{P\}\textbf{with } r_i \textbf{ when } B \textbf{ do } C\{Q\}}$$

No invariant needed in the conclusion!

Counting Semaphore Program

Chapter 10. Skipped for the sake of time.

Interesting remark: This logic does not allow for dynamic resource allocation.

Are there logics with this capability?

Reynolds Counterexample

- At the beginning we said "invariants should be precise".
- If not, then the system is incompatible with the rule

$$\frac{\{P\}C\{Q\} \quad \{P'\}C\{Q'\}}{\{P \wedge P'\}C\{Q \wedge Q'\}}$$

- Counterexample:

resource $r()$

$I_r = \mathbf{true}$

$C \triangleq$ **with** r **when true do skip**

Reynolds Counterexample

1. We derive $\{\mathbf{emp} \vee 10 \mapsto -\}C\{\mathbf{emp}\}$:

$$\dfrac{\dfrac{\dfrac{}{\{\mathbf{true}\}\mathbf{skip}\{\mathbf{true}\}}\text{Skip}}{\{(\mathbf{emp} \vee 10 \mapsto -) * \mathbf{true}\}\mathbf{skip}\{\mathbf{emp} * \mathbf{true}\}}\text{Conseq}}{\{\mathbf{emp} \vee 10 \mapsto -\}C\{\mathbf{emp}\}}\text{CCR}$$

2. Then $\{10 \mapsto -\}C\{10 \mapsto -\}$:

$$\dfrac{\dfrac{\dfrac{\{\mathbf{emp} \vee 10 \mapsto -\}C\{\mathbf{emp}\}}{\{\mathbf{emp}\}C\{\mathbf{emp}\}}\text{Conseq}}{\{\mathbf{emp} * 10 \mapsto -\}C\{\mathbf{emp} * 10 \mapsto -\}}\text{Frame}}{\{10 \mapsto -\}C\{10 \mapsto -\}}\text{Conseq}$$

3. Then $\{10 \mapsto -\}C\{\mathbf{emp}\}$:

$$\dfrac{\{\mathbf{emp} \vee 10 \mapsto -\}C\{\mathbf{emp}\}}{\{10 \mapsto -\}C\{\mathbf{emp}\}}\text{Conseq}$$

Reynolds Counterexample

Applying the conjunction rule:

$$\frac{\{10 \mapsto -\}C\{10 \mapsto -\} \quad \{10 \mapsto -\}C\{\mathbf{emp}\}}{\{10 \mapsto - \wedge 10 \mapsto -\}C\{\underbrace{10 \mapsto - \wedge \mathbf{emp}}_{\mathbf{false}}\}}$$

Blame the invariant!

(Or the precondition, in O/G, but not valid in this setting).

Precise invariants

An invariant I is **precise** if for all states (s, h) there's at most one subheap h' s.t.

$$h' \subseteq h, \quad (s, h) \Vdash I, \quad (s, h') \Vdash I$$

true is not precise...

> Precise predicates let us split the heap in an unique way.
>
> This is precisely what is needed to prove soundness!

Precise precise definition

From Viktor's: P is precise iff $\forall h_1, h_1', h_2, h_2',$

$\mathbf{def}(h_1 * h_2)$ and $h_1 * h_2 = h_1' * h_2'$ and $(s, h_1) \Vdash P$ and $(s, h_1') \Vdash P$

then $h_1 = h_1'$.

Not so restrictive in practice:

$$P ::= \mathbf{emp} \mid x \mapsto E \mid (P * P') \mid (P \wedge Q) \mid (Q \wedge P) \mid (B \wedge P) \vee (B' \wedge P')$$

Brookes say other definition is possible (with P and Q being *intuitionistic* for Region and $\mathrm{Resource}$, and just the precise part of the heap is used in the hypotheses of these rules).

A few words on Viktor's

$$I \vdash \{P\}C\{Q\}$$

- State from I can be changed by other threads.
- State P is owned by C.
- I is maintained before, during, and after C.
- Definition of *safe*: A program C is safe to run under invariant I and precondition P, and will produce Q.
- Simpler proof. Bakes the frame rule into the definition of *safe*.

And a few more

- All proved in Isabelle.
- Do not require abort semantics to prove soundness (in contrast with Brookes).
- Same proof for soundness without the conjunction rule *or* with precise invariants.
- Sharing read permissions!

$$\{10 \mapsto -\}x := [10] \parallel y := [10]\{10 \mapsto -\}$$

- $10 \mapsto - = 10 \mapsto^{0.5} - * 10 \mapsto^{0.5} -$.
- Can only update if $10 \mapsto^{1} -$.
- Soundness of RGSep also proved (stay tuned!)

Thanks!