Einzelpraktikum:

Combination of Rely/Guarantee and Separation Logic in SecC

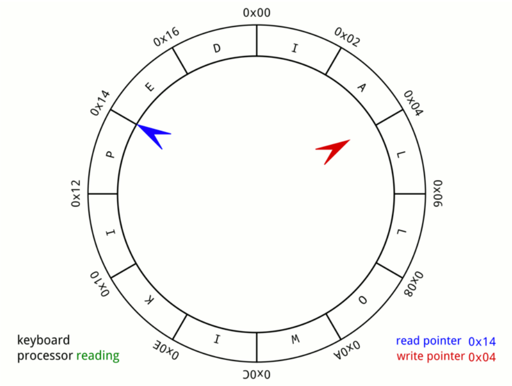
Bernhard Pöttinger

29.01.2020

Starting Point

- Last semester: Seminar paper on SL and Verifast
- Now: Einzelpraktikum
  - SecC: CDDC case study requires ring-buffer
  - Task: verify ring-buffer

Single-Producer-Single-Consumer Ring-Buffer



Single-Producer-Single-Consumer Ring-Buffer

```
T buf[N];
size_t rpos = 0, wpos = 0;

void write(T x) {
  size_t _wpos = atomic_load(&wpos);
  while (!(_wpos < atomic_load(&rpos) + N)) { continue; }
  buf[_wpos % N] = x;      // <-- we must be sure that no other thread interferes fatally
  atomic_increment(&wpos);
}

T read() {
  size_t _rpos = atomic_load(&rpos);
  while (!(atomic_load(&wpos) > _rpos)) { continue; }
  T x = buf[_rpos % N];   // <-- we must be sure that no other thread interferes fatally
  atomic_increment(&rpos);
  return x;
}
```

Problem Statement

Ring-buffer is a **lock-free** data-structure.

However, SecC's SL only supports **lock-based** data-structures [OHe07; EM19].

We need support for:

- Atomic operations
- Sequences of atomic operations preserving knowledge

Related Work

Basics of SL, CSL, and SecC:

- SL [Rey02]
- CSL [OHe07; Bro07; Vaf11]
- SecCSL [EM19]

Research:

- RSL: Weak Memory Model [VN13]
- Verifast: Shared Boxes [Sma+14]
- RGSep: Combination of SL and RG [VP07; Vaf08; Vaf10]
- Rely/Guarantee [Jon83; Vaf08]
- CaReSL, GPS, Iris, ...

Outline

Cross-section of SL, RG, and RGSep.

- SL: modularity, but no interference
- RG: interference, but no modularity
- RGSep: modularity and interference
- RGSep for SecC

Separation Logic

Separation Logic [Rey02; OHe07; Bro07]

"Global Reasoning": accesses, that are not forbidden, are permitted
"Local Reasoning": accesses, that are not permitted, are forbidden

"Local Reasoning" enables modular proofs!

Separation Logic [Rey02; OHe07; Bro07]

- Permission for heap accesses: $e_p \mapsto e_v$

Separation Logic [Rey02; OHe07; Bro07]

- Permission for heap accesses: $e_p \mapsto e_v$
- Require explicit permission to access heap:

$$\frac{}{\vdash \{e_p \mapsto \_\}\ [e_p] := e_v\ \{e_p \mapsto e_v\}}\ \textsc{SL-Store}$$

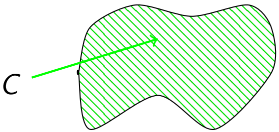
Separation Logic [Rey02; OHe07; Bro07]

- Permission for heap accesses: $e_p \mapsto e_v$
- Require explicit permission to access heap:

$$\frac{}{\vdash \{e_p \mapsto \_\}\ [e_p] := e_v\ \{e_p \mapsto e_v\}}\ \textsc{SL-Store}$$

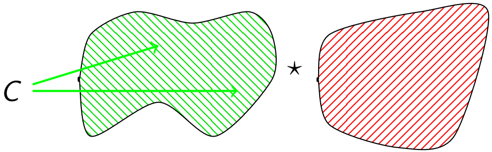
- Compose permissions:

$$P_1 \star P_2$$

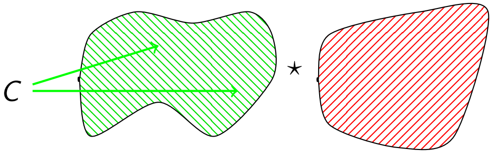


Restriction: P_1, P_2 describe disjoint parts of heap

Separation Logic [Rey02; OHe07; Bro07]



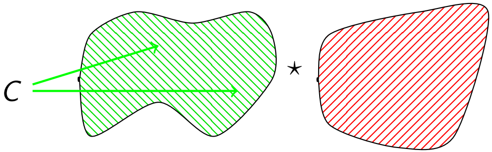
Separation Logic [Rey02; OHe07; Bro07]



- Permissions can be restricted:

$$\frac{\vdash \{P\}\ c\ \{Q\} \quad \ldots}{\vdash \{P \star F\}\ c\ \{Q \star F\}}\ \textsc{SL-Frame}$$

Separation Logic [Rey02; OHe07; Bro07]



- Permissions can be restricted:

$$\frac{\vdash \{P\}\ c\ \{Q\} \quad \ldots}{\vdash \{P \star F\}\ c\ \{Q \star F\}}\ \textsc{SL-Frame}$$

- Parallel rule:

$$\frac{\vdash \{P_1\}\ c_1\ \{Q_1\} \quad \vdash \{P_2\}\ c_2\ \{Q_2\} \quad \ldots}{\vdash \{P_1 \star P_2\}\ c_1 \parallel c_2\ \{Q_1 \star Q_2\}}\ \textsc{SL-Par}$$

Separation Logic [Rey02; OHe07; Bro07]

Stength: Modularity

- SL-FRAME enables reuse of function verification in different contexts

Separation Logic [Rey02; OHe07; Bro07]

Stength: Modularity

- SL-FRAME enables reuse of function verification in different contexts

Limitation: No lock-free algorithms

- Those usually require some kind of interference
- Separation logic prevents interference by construction: always only a single owner of heap cells

Rely/Guarantee

Rely/Guarantee [Jon83; Vaf08]

Atomic modifications on shared resources must adhere to permissions.

Rely/Guarantee [Jon83; Vaf08]

Atomic modifications on shared resources must adhere to permissions.

Each thread t is allowed to execute operations adhering to R_t:

- What is **this thread** allowed to do? (Guarantee: R_t)
- What are **other threads** allowed to do? (Rely: $\bigvee_{t' \neq t} R_{t'}$)

Rely/Guarantee [Jon83; Vaf08]

Atomic modifications on shared resources must adhere to permissions.

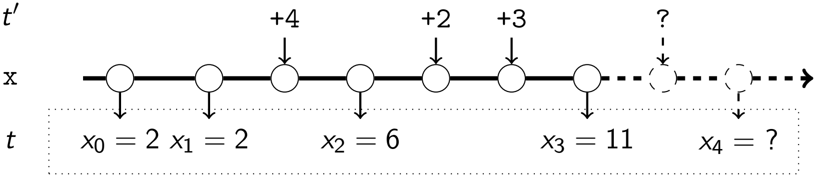
Each thread t is allowed to execute operations adhering to R_t:

- What is **this thread** allowed to do? (Guarantee: R_t)
- What are **other threads** allowed to do? (Rely: $\bigvee_{t' \neq t} R_{t'}$)

$$\frac{C_1 \textbf{ sat } (P, R \vee G_2, G_1, Q_1) \quad C_2 \textbf{ sat } (P, R \vee G_1, G_2, Q_2)}{C_1 \parallel C_2 \textbf{ sat } (P, R, G_1 \vee G_2, Q_1 \wedge Q_2)} \text{ RG-PAR}$$

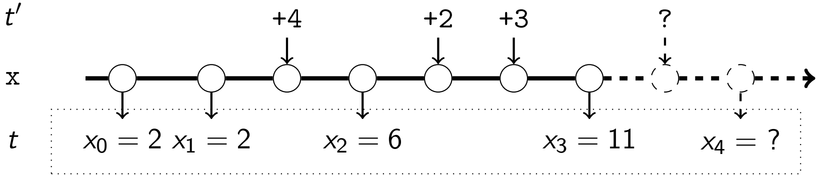
Rely/Guarantee [Jon83; Vaf08]

Example: $R = (x \leq x')$



Rely/Guarantee [Jon83; Vaf08]

Example: $R = (x \leq x')$

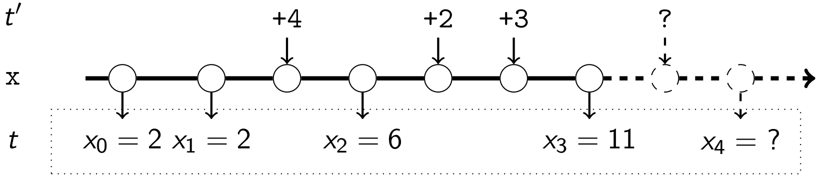


Necessary property of predicates (single-state variant):

- **Stability**: $\forall x\ x'.\ P(x) \longrightarrow R(x, x') \longrightarrow P(x')$

Rely/Guarantee [Jon83; Vaf08]

Example: $R = (x \leq x')$



Necessary property of predicates (single-state variant):

- **Stability**: $\forall x\ x'.\ P(x) \longrightarrow R(x, x') \longrightarrow P(x')$

$$\frac{C\ \textbf{sat}\ (P, \mathsf{Id}, \top, Q \wedge G) \qquad P, Q \text{ stable under } R}{\langle C \rangle\ \textbf{sat}\ (P, R, G, Q)}\ \text{RG-ATOM}$$

Rely/Guarantee [Jon83; Vaf08]

Stength:

- Interference can be described

Rely/Guarantee [Jon83; Vaf08]

Stength:

- Interference can be described

Limitation:

- Non-modular proofs (with respect to heap)

RGSep

RGSep [VP07; Vaf08]

Problems identified so far:

- SL: missing interference
- RG: missing modularity

RGSep [VP07; Vaf08]

Problems identified so far:

- SL: missing interference
- RG: missing modularity

Combination of SL and RG:

- Describe shared state using boxes: $P \star \boxed{S}$
- Access shared state via atomic blocks: `atomic { ... }`
- Rely and guarantee are sets of actions $S \rightsquigarrow S'$

RGSep [Vaf08]

$$\frac{\vdash C\ \textbf{sat}\ (P, R, G, Q) \quad F \text{ stable under } (R \cup G)}{\vdash C\ \textbf{sat}\ (P \star F, R, G, Q \star F)}\ \textsc{RGSep-Frame}$$

RGSep [Vaf08]

$$\frac{\vdash C \ \textbf{sat}\ (P, R, G, Q) \quad F \text{ stable under } (R \cup G)}{\vdash C \ \textbf{sat}\ (P \star F, R, G, Q \star F)} \ \textsc{RGSep-Frame}$$

$$\frac{\vdash C_1 \ \textbf{sat}\ (P_1, R \cup G_2, G_1, Q_1) \quad \vdash C_2 \ \textbf{sat}\ (P_2, R \cup G_1, G_2, Q_2)}{\vdash C_1 \parallel C_2 \ \textbf{sat}\ (P_1 \star P_2, R, G_1 \cup G_2, Q_1 \star Q_2)} \ \textsc{RGSep-Par}$$

RGSep [Vaf08]

$$\frac{\vdash C \ \textbf{sat}\ (P \star P', \emptyset, \emptyset, Q \star Q') \quad P \rightsquigarrow Q \subseteq G \quad \ldots}{\vdash \langle C \rangle \ \textbf{sat}\ (\boxed{P} \star P', \emptyset, G, \boxed{Q} \star Q')} \ \textsc{RGSep-Atom}$$

RGSep [Vaf08]

$$\frac{\vdash C \ \textbf{sat}\ (P \star P', \emptyset, \emptyset, Q \star Q') \quad P \rightsquigarrow Q \subseteq G \quad \ldots}{\vdash \langle C \rangle \ \textbf{sat}\ (\boxed{P} \star P', \emptyset, G, \boxed{Q} \star Q')} \ \textsc{RGSep-Atom}$$

$$\frac{\vdash \langle C \rangle \ \textbf{sat}\ (P, \emptyset, G, Q) \quad P, Q \text{ stable under } R}{\vdash \langle C \rangle \ \textbf{sat}\ (P, R, G, Q)} \ \textsc{RGSep-AtomR}$$

RGSep in SecC

RGSep in SecC

```
void acquire_lock(struct lock *l, int *p)
  // shared      exists int o, int x. mylock(l; p, o, x)
  //        rely old(o) == TID ==> old(x) == LOCKED ==> x == old(x) && o == old(o)
  //   guarantee old(o) != TID ==> old(x) == LOCKED ==> x == old(x) && o == old(o)
  //
  // ensures inv(p) && o == TID && x == LOCKED
{
  // atomic begin
  // unfold mylock(l; p, o, x)
  int r = atomic_compare_exchange(&l->is_locked, UNLOCKED, LOCKED);
  // apply if (r == UNLOCKED) { l->owner = TID; }
  // fold mylock(l; p, r == UNLOCKED ? TID : o, LOCKED)
  // atomic end

  if (r != UNLOCKED) {
    acquire_lock(l, p);
  }
}
```

RGSep in SecC

Goal:

- Simple integration of RG into SecC

RGSep in SecC

Goal:

- Simple integration of RG into SecC

Solution:

- Specification $S(x)$ of shared state (invariant)
- Atomic blocks for access to shared state
- Rely und guarantee as pure formulas over $\exists$-vars x of shared state
- Case distinctions for different heap shapes

RGSep in SecC

Goal:

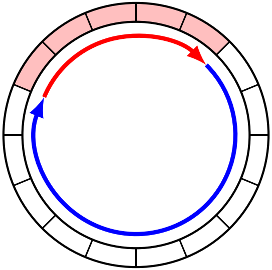
- Simple integration of RG into SecC

Solution:

- Specification $S(x)$ of shared state (invariant)
- Atomic blocks for access to shared state
- Rely und guarantee as pure formulas over $\exists$-vars x of shared state
- Case distinctions for different heap shapes
- Rely R must be reflexive and transitive
- No stability checks. With $P(e) := \exists x.\ R(e, x) \star S(x)$:

$$R(e, x) \star S(x) \Longrightarrow R(x, x') \Longrightarrow R(e, x') \star S(x')$$

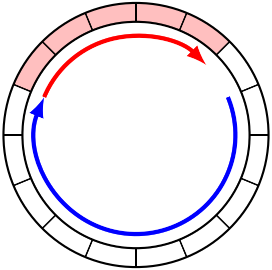
Example: SPSC-Ringbuffer



$S = slice(r_1, w_0 \quad) \star slice(w_1 \quad , r_0 + N)$

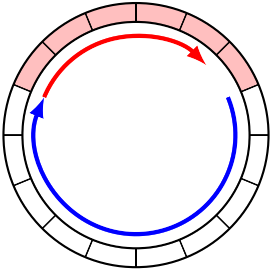
$L = \texttt{emp}$

Example: SPSC-Ringbuffer



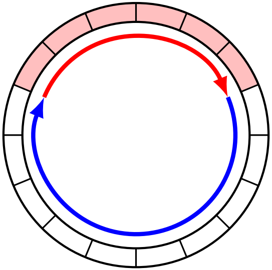
$$S = slice(r_1, w_0 \quad) \star slice(w_1{+}1, r_0 + N)$$
$$L = (p + w_0) \mapsto -$$

Example: SPSC-Ringbuffer



$$S = slice(r_1, w_0 \quad) \star slice(w_1{+}1, r_0 + N)$$
$$L = (p + w_0) \mapsto x$$

Example: SPSC-Ringbuffer



$$S = slice(r_1, w_0{+}1) \star slice(w_1{+}1, r_0 + N)$$
$$L = \texttt{emp}$$

Example: SPSC-Ringbuffer

Invariant:

$$\ldots \wedge$$
$$0 \leq r_0 \leq r_1 \leq w_0 \leq w_1 \leq r_0 + N \wedge$$
$$r_0 \leq r_1 \leq r_0 + 1 \wedge$$
$$w_0 \leq w_1 \leq w_0 + 1 \wedge$$
$$slice(d, r_1, w_0, N, I) \star slice(d, w_1, r_0 + N, N, I)$$

Example: SPSC-Ringbuffer

Invariant:

$$\ldots \land$$
$$0 \le r_0 \le r_1 \le w_0 \le w_1 \le r_0 + N \land$$
$$r_0 \le r_1 \le r_0 + 1 \land$$
$$w_0 \le w_1 \le w_0 + 1 \land$$
$$slice(d, r_1, w_0, N, l) \star slice(d, w_1, r_0 + N, N, l)$$

Rely (read), Guarantee (write):

$$r_0' = r_0 \land r_1' = r_1 \land w_0' \ge w_0 \land w_1' \ge w_1 \land \ldots$$

Guarantee (read), Rely (write):

$$r_0' \ge r_0 \land r_1' \ge r_1 \land w_0' = w_0 \land w_1' = w_1 \land \ldots$$

Example: SPSC-Ringbuffer

Guarantee: $r'_0 = r_0 \land r'_1 = r_1 \land w'_0 \geq w_0 \land w'_1 \geq w_1 \land \ldots$

Rely: $r'_0 \geq r_0 \land r'_1 \geq r_1 \land w'_0 = w_0 \land w'_1 = w_1 \land \ldots$

```
// modularity allows us to restrict our attention to relevant details
T buf[N];
size_t rpos = 0, wpos = 0;
// size_t rpos1 = 0, wpos1 = 0;

void write(T x) {
  size_t _wpos = atomic_load(&wpos);
  while (!(_wpos < atomic_load(&rpos) + N)) { continue; }

  // rely allows us to deduce: wpos = _wpos < _rpos + N <= rpos
  // required to re-establish the ringbuffer invariant

  // buf->wpos1 += 1;
  buf[_wpos % N] = x;
  atomic_increment(&wpos);
}
```

Implementation Details

- Add check that rely is reflexive and transitive
- Atomic blocks: rely step + guarantee check + only single atomic operation in block
- Check compatible shared block at function calls

Summary

Summary:

- Design and implementation of variant of RGSep for SecC
- Verification of spinlock and ring-buffer
- Implementation of ghost fields for structs

The End

Questions?

Literature

[Bro07] Stephen Brookes. „A Semantics for Concurrent Separation Logic“. In: *Theor. Comput. Sci.* 375.1-3 (Apr. 2007), S. 227–270.

[EM19] Gidon Ernst und Toby Murray. „SecCSL: Security Concurrent Separation Logic“. In: *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*. Hrsg. von Isil Dillig und Serdar Tasiran. Bd. 11562. Lecture Notes in Computer Science. Springer, 2019, S. 208–230.

[Jon83] Cliff B. Jones. „Specification and Design of (Parallel) Programs“. In: *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983*. Hrsg. von R. E. A. Mason. North-Holland/IFIP, 1983, S. 321–332.

[OHe07] Peter W. O'Hearn. „Resources, Concurrency, and Local Reasoning“. In: *Theor. Comput. Sci.* 375.1-3 (Apr. 2007), S. 271–307.

[Rey02] 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. Washington, DC, USA: IEEE Computer Society, 2002, S. 55–74.

[Sma+14] Jan Smans u. a. *Shared boxes: rely-guarantee reasoning in VeriFast*. CW Reports CW662. Department of Computer Science, KU Leuven, Mai 2014.

[Vaf08] Viktor Vafeiadis. „Modular fine-grained concurrency verification“. Diss. University of Cambridge, UK, 2008.

[Vaf10] Viktor Vafeiadis. „RGSep Action Inference“. In: *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*. Hrsg. von Gilles Barthe und Manuel V. Hermenegildo. Bd. 5944. Lecture Notes in Computer Science. Springer, 2010, S. 345–361.

[Vaf11] Viktor Vafeiadis. *Concurrent Separation Logic Soundness*. 2011.

[VN13] Viktor Vafeiadis und Chinmay Narayan. „Relaxed Separation Logic: A Program Logic for C11 Concurrency“. In: *SIGPLAN Not.* 48.10 (Okt. 2013), S. 867–884.

[VP07] Viktor Vafeiadis und Matthew J. Parkinson. „A Marriage of Rely/Guarantee and Separation Logic“. In: *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*. Hrsg. von Luis Caires und Vasco Thudichum Vasconcelos. Bd. 4703. Lecture Notes in Computer Science. Springer, 2007, S. 256–271.

RGSep in SecC

$$\frac{x \notin \mathsf{free}(e)}{\ell, R, G \vdash \{\mathtt{emp}\}\ x := e\ \{x = e\}}\ \textsc{Asg}$$

$$\frac{x \notin \mathsf{free}(e_p, e_v, e_l)}{\ell, R, G \vdash \{e_p \overset{e_l}{\longmapsto} e_v\}\ x := [e_p]\ \{x = e_v \wedge e_p \overset{e_l}{\longmapsto} e_v\}}\ \textsc{Load}$$

$$\frac{}{\ell, R, G \vdash \{e_v :: e_l \wedge e_p \overset{e_l}{\longmapsto} \_\}\ [e_p] := e_v\ \{e_p \overset{e_l}{\longmapsto} e_v\}}\ \textsc{Store}$$

$$\frac{}{\ell, R, G \vdash \{\mathtt{emp}\}\ \mathtt{lock}\ l\ \{\mathtt{inv}(l)\}}\ \textsc{Lock}$$

$$\frac{}{\ell, R, G \vdash \{\mathtt{inv}(l)\}\ \mathtt{unlock}\ l\ \{\mathtt{emp}\}}\ \textsc{Unlock}$$

$$\frac{\ell, R, G \vdash \{b \wedge P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\} \quad \ell, R, G \vdash \{\neg b \wedge P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\}}{\ell, R, G \vdash \{b :: \ell \wedge P \star \boxed{S(e)}\}\ \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \{Q \star \boxed{S(e')}\}}\ \textsc{If}$$

$$\frac{\ell, R, G \vdash \{\phi \wedge P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\} \quad \ell, R, G \vdash \{\neg\phi \wedge P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\}}{\ell, R, G \vdash \{\phi :: \ell \wedge P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\}}\ \textsc{Split}$$

$$\frac{\ell, R, G \vdash \{P \star \boxed{S(e)}\}\ c_1\ \{R \star \boxed{S(e')}\} \quad \ell, R, G \vdash \{R \star \boxed{S(e')}\}\ c_2\ \{Q \star \boxed{S(e'')}\}}{\ell, R, G \vdash \{P \star \boxed{S(e)}\}\ c_1; c_2\ \{Q \star \boxed{S(e'')}\}}\ \textsc{Seq}$$

$$S(e) = \exists x.\ R(e, x) \star S_0(x)$$

$$\frac{\ell, R, G \vdash \{b \wedge b :: \ell \wedge P[x \mapsto x_0] \star \boxed{S(x_0)}\}\ c\ \{b :: \ell \wedge P[x \mapsto e'] \star \boxed{S(e')}\} \quad x \text{ fresh}}{\ell, R, G \vdash \{b :: \ell \wedge P[x \mapsto e] \star \boxed{S(e)}\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{\neg b \wedge P[x \mapsto x_1] \star \boxed{S(x_1)}\}}\ \textsc{While}$$

$$\frac{\ell, R, G \vdash \{P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\} \quad \text{modified}(c) \cap \text{free}(F) = \varnothing}{\ell, R, G \vdash \{P \star F \star \boxed{S(e)}\}\ c\ \{Q \star F \star \boxed{S(e')}\}}\ \textsc{Frame}$$

$$\frac{\ell, R', G' \vdash \{P' \star \boxed{S(e)}\}\ c\ \{Q' \star \boxed{S(e')}\} \quad P \overset{\ell}{\Longrightarrow} P' \quad Q' \overset{\ell}{\Longrightarrow} Q \quad R \Longrightarrow R' \quad G' \Longrightarrow G}{\ell, R, G \vdash \{P \star \boxed{S(e)}\}\ c\ \{Q \star \boxed{S(e')}\}}\ \textsc{Conseq}$$

$$\frac{\ell, R \vee G_2, G_1 \vdash \{P_1 \star \boxed{S(e)}\}\ c_1\ \{Q_1 \star \boxed{S(e_1')}\} \quad \ell, R \vee G_1, G_2 \vdash \{P_2 \star \boxed{S(e)}\}\ c_2\ \{Q_2 \star \boxed{S(e_2')}\} \quad \text{modified}(c_i) \cap \text{free}(c_j, P_j, Q_j) = \varnothing \text{ for } i \neq j}{\ell, R, G_1 \vee G_2 \vdash \{P_1 \star P_2 \star \boxed{S(e)}\}\ c_1 \parallel c_2\ \{Q_1 \star Q_2 \star \boxed{S(e_1') \wedge S(e_2')}\}}\ \textsc{Par}^1$$

$$\frac{\ell, \bot, \top \vdash \{P \star R(e, x) \star S_0(x)\}\ c\ \{P' \star G(x, e') \star S_0(e')\} \quad x \text{ fresh}}{\ell, R, G \vdash \{P \star \boxed{S(e)}\}\ \langle c \rangle\ \{P' \star \boxed{S(e')}\}}\ \textsc{Atom}^2$$

---

[1]See [Vaf08] for handling of S: $\boxed{P} \star \boxed{Q} \iff \boxed{P \wedge Q}$

[2]Using implicitly Ex, Conseq, and definition of S