

Syntactic Control of Interference for Concurrent Separation Logic

Uday S. Reddy¹ John C. Reynolds²

¹School of Computer Science
University of Birmingham

²School of Computer Science
Carnegie-Mellon University

Principles of Prog. Languages, Philadelphia, Jan 2012

Outline

Motivation

Syntactic Control of Interference (SCI)

Concurrent Separation Logic

SCI for Conditional Critical Regions

SCI for Concurrent Separation Logic

Comparisons

Semantics

Permission Inference

Conclusion

Details - Natural deduction framework

Details - Comparisons

Details - Semantics

Concurrent Programming

- ▶ Concurrent programming requires a tight control over resources
 - ▶ Be clear about what resources are being used by each process
 - ▶ Ensure that these resources are disjoint as far as possible
 - ▶ Devise suitable protocols for sharing resources when necessary
- ▶ Concurrent Separation Logic does all of these very well for heap locations.
- ▶ But it brushes under the carpet the very same issues for variables.
- ▶ This work is an attempt to change that.

How are variables different?

- ▶ Variables are syntactic symbols.
- ▶ It should be possible to control their usage in the formulation of “syntax”.
- ▶ Variables participate in expressions, which represent read-only uses of the resources.
- ▶ Need to make this convenient.

Section 2

Syntactic Control of Interference (SCI)

Syntactic Control of Interference (SCI)

John C. Reynolds,

Syntactic Control of Interference,

Conf. Record of the Fifth Ann. ACM Symp. on

Principles of Programming Languages,

pages 39–46, Tucson, Arizona, January 1978.



*In programming languages which permit both assignment and procedures, distinct identifiers can represent data structures which **share storage** or procedures with interfering side effects.*

*In addition to being a direct source of programming errors, this phenomenon, which we call **interference**, can impact type structure and parallelism.*

*We show how to eliminate these difficulties by imposing **syntactic restrictions**, without prohibiting the kind of constructive interference. . . .*

*The basic idea is to prohibit interference between **identifiers**. . .*

Hoare [3,4] and Brinch Hansen [5] have argued convincingly that intelligible programming requires all interactions between parallel processes to be mediated by some mechanism such as a critical region or monitor.

In this paper, we will not consider interacting parallel processes, but we will permit the parallel construct $S_1 \parallel S_2$ when it is syntactically evident that S_1 and S_2 do not interfere.

Syntactic Control of Interference

Two terms

$$T_1 \quad T_2$$

are deemed to *interfere*:

- ▶ if any free variable actively used in one term is used in the other term (as a free variable again).
- ▶ The two terms can share passively used free variables.
- ▶ **Procedure call:** $F \quad (\quad A \quad)$
- ▶ **local declarations:** $\text{let } f = A \quad \text{in } B$
- ▶ **Parallel composition:** $C_1 \quad || \quad C_2$

Further work on SCI

[O'Hearn 1991] *Linear logic and interference control*, CTCS

[O'Hearn 1993] *A model for syntactic control of interference*, MSCS

[Reddy 1996] *Global state considered unnecessary: An introduction to object-based semantics*, J. Lisp & SC

[McCusker, 2010] *A graph model for imperative computation*, LMCS

[O'Hearn, Power, Takeyama, Tennent 1995] *Syntactic control of interference Revisited*, MFPS

[McCusker, 2007] *Categorical models of syntactic control of interference revisited, revisited*, LMSJCM

[Ghica, Murawski, Ong] *Syntactic control of concurrency*, TCS

[Ghica 2007] *Geometry of synthesis: A structured approach to VLSI design*, POPL.

Further work on SCI (contd)

- ▶ Bunched typing arose from an effort combine SCI with regular function application:
[O'Hearn 2003] *On bunched typing*, JFP
- ▶ BI arose from viewing these type systems as logics:
[Pym, O'Hearn 1999] *The logic of bunched implications*, BSL
- ▶ **Separation Logic** uses BI as its assertion logic.

O'Hearn's formulation of SCI

(inspired by Linear Logic)

The contexts are combined multiplicatively.

$$\frac{\Sigma_1 \vdash F : \tau_1 \rightarrow \tau_2 \quad \Sigma_2 \vdash A : \tau_1}{\Sigma_1, \Sigma_2 \vdash F(A) : \tau_2}$$

$$\frac{\Sigma_1 \vdash A : \tau_1 \quad \Sigma_2, f : \tau_1 \vdash B : \tau_2}{\Sigma_1, \Sigma_2 \vdash \mathbf{let } f = A \mathbf{ in } B : \tau_2}$$

$$\frac{\Sigma_1 \vdash C_1 \mathbf{Comm} \quad \Sigma_2 \vdash C_2 \mathbf{Comm}}{\Sigma_1, \Sigma_2 \vdash C_1 \parallel C_2}$$

There is also additional machinery to deal with passive uses.

SCI is the “mother” of Separation Logic.

- ▶ **However:** Remarkably, SCI has never been used to structure **programming logics!**
- ▶ SCI never extended to deal with **interacting parallel processes!**

SCI is the “mother” of Separation Logic.

- ▶ **However:** Remarkably, SCI has never been used to structure **programming logics!**
- ▶ SCI never extended to deal with **interacting parallel processes!**

Section 3

Concurrent Separation Logic

Concurrent Separation Logic

Parallel composition

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \star P_2\} C_1 \parallel C_2 \{Q_1 \star Q_2\}}$$

Owicki-Gries:

- ▶ if no variable free in P_i or Q_i is changed in C_j ($j \neq i$).
- ▶ if a variable x is changed in a process C_j , it cannot appear in C_i ($j \neq i$) unless it belongs to a resource.
- ▶ if a variable x belongs to a resource, it cannot appear in a parallel process except in a critical section for r .

Brookes:

- ▶ $\mathbf{free}(P_1, Q_1) \cap \mathbf{writes}(C_2) = \mathbf{free}(P_2, Q_2) \cap \mathbf{writes}(C_1) = \emptyset$
- ▶ $(\mathbf{free}(C_1) \cap \mathbf{writes}(C_2)) \cup (\mathbf{free}(C_2) \cap \mathbf{writes}(C_1)) \subseteq \mathbf{owned}(\Gamma)$ where Γ lists all the resources in the context

Concurrent Separation Logic

Parallel composition

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \star P_2\} C_1 \parallel C_2 \{Q_1 \star Q_2\}}$$

Owicki-Gries:

- ▶ if no variable free in P_i or Q_i is changed in C_j ($j \neq i$).
- ▶ if a variable x is changed in a process C_j , it cannot appear in C_i ($j \neq i$) unless it belongs to a resource.
- ▶ if a variable x belongs to a resource, it cannot appear in a parallel process except in a critical section for r .

Brookes:

- ▶ $\text{free}(P_1, Q_1) \cap \text{writes}(C_2) = \text{free}(P_2, Q_2) \cap \text{writes}(C_1) = \emptyset$
- ▶ $(\text{free}(C_1) \cap \text{writes}(C_2)) \cup (\text{free}(C_2) \cap \text{writes}(C_1)) \subseteq \text{owned}(\Gamma)$ where Γ lists all the resources in the context

Concurrent Separation Logic

Parallel composition

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \star P_2\} C_1 \parallel C_2 \{Q_1 \star Q_2\}}$$

Owicki-Gries:

- ▶ if no variable free in P_i or Q_i is changed in C_j ($j \neq i$).
- ▶ if a variable x is changed in a process C_i , it cannot appear in C_j ($j \neq i$) unless it belongs to a resource.
- ▶ if a variable x belongs to a resource, it cannot appear in a parallel process except in a critical section for r .

Brookes:

- ▶ $\text{free}(P_1, Q_1) \cap \text{writes}(C_2) = \text{free}(P_2, Q_2) \cap \text{writes}(C_1) = \emptyset$
- ▶ $(\text{free}(C_1) \cap \text{writes}(C_2)) \cup (\text{free}(C_2) \cap \text{writes}(C_1)) \subseteq \text{owned}(\Gamma)$ where Γ lists all the resources in the context

Concurrent Separation Logic

Parallel composition

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \star P_2\} C_1 \parallel C_2 \{Q_1 \star Q_2\}}$$

Owicki-Gries:

- ▶ if no variable free in P_i or Q_i is changed in C_j ($j \neq i$).
- ▶ if a variable x is changed in a process C_i , it cannot appear in C_j ($j \neq i$) unless it belongs to a resource.
- ▶ if a variable x belongs to a resource, it cannot appear in a parallel process except in a critical section for r .

Brookes:

- ▶ $\text{free}(P_1, Q_1) \cap \text{writes}(C_2) = \text{free}(P_2, Q_2) \cap \text{writes}(C_1) = \emptyset$
- ▶ $(\text{free}(C_1) \cap \text{writes}(C_2)) \cup (\text{free}(C_2) \cap \text{writes}(C_1)) \subseteq \text{owned}(\Gamma)$ where Γ lists all the resources in the context

Concurrent Separation Logic

Parallel composition

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \star P_2\} C_1 \parallel C_2 \{Q_1 \star Q_2\}}$$

Owicki-Gries:

- ▶ if no variable free in P_i or Q_i is changed in C_j ($j \neq i$).
- ▶ if a variable x is changed in a process C_i , it cannot appear in C_j ($j \neq i$) unless it belongs to a resource.
- ▶ if a variable x belongs to a resource, **it cannot appear in a parallel process except in a critical section for r .**

Brookes:

- ▶ $\mathbf{free}(P_1, Q_1) \cap \mathbf{writes}(C_2) = \mathbf{free}(P_2, Q_2) \cap \mathbf{writes}(C_1) = \emptyset$
- ▶ $(\mathbf{free}(C_1) \cap \mathbf{writes}(C_2)) \cup (\mathbf{free}(C_2) \cap \mathbf{writes}(C_1)) \subseteq \mathbf{owned}(\Gamma)$ where Γ lists all the resources in the context

Concurrent Separation Logic (contd)

$$\frac{\{P \star R \wedge B\} C \{Q \star R\}}{\{P\} \mathbf{with } r \mathbf{ when } B \mathbf{ do } C \mathbf{ od } \{Q\}} \quad \mathit{CRIT}$$

Owicki-Gries:

- ▶ No variable free in P or Q is changed in any “other process”.

The reference to “other processes” makes this rule non-compositional.

Brookes:

- ▶ $r \notin \text{dom}(\Gamma)$
- ▶ $X \cap \mathbf{owned}(\Gamma) = \emptyset$
- ▶ $\mathbf{free}(R) \cap \mathbf{owned}(\Gamma) = \emptyset$
- ▶ $\mathbf{free}(P, Q) \cap X = \emptyset$

*These conditions are compositional. But they are not enough!
The system was found to be unsound [Wehrman].*

Concurrent Separation Logic (contd)

$$\frac{\{P \star R \wedge B\} C \{Q \star R\}}{\{P\} \mathbf{with} \ r \ \mathbf{when} \ B \ \mathbf{do} \ C \ \mathbf{od} \ \{Q\}} \quad \mathit{CRIT}$$

Owicki-Gries:

- ▶ No variable free in P or Q is changed in any “other process”.

The reference to “other processes” makes this rule non-compositional.

Brookes:

- ▶ $r \notin \text{dom}(\Gamma)$
- ▶ $X \cap \mathbf{owned}(\Gamma) = \emptyset$
- ▶ $\mathbf{free}(R) \cap \mathbf{owned}(\Gamma) = \emptyset$
- ▶ $\mathbf{free}(P, Q) \cap X = \emptyset$

*These conditions are compositional. But they are not enough!
The system was found to be unsound [Wehrman].*

Concurrent Separation Logic (contd)

$$\frac{\{P \star R \wedge B\} C \{Q \star R\}}{\{P\} \mathbf{with} \ r \ \mathbf{when} \ B \ \mathbf{do} \ C \ \mathbf{od} \ \{Q\}} \quad \text{CRIT}$$

Owicki-Gries:

- ▶ No variable free in P or Q is changed in any “other process”.

The reference to “other processes” makes this rule non-compositional.

Brookes:

- ▶ $r \notin \text{dom}(\Gamma)$
- ▶ $X \cap \mathbf{owned}(\Gamma) = \emptyset$
- ▶ $\mathbf{free}(R) \cap \mathbf{owned}(\Gamma) = \emptyset$
- ▶ $\mathbf{free}(P, Q) \cap X = \emptyset$

*These conditions are compositional. But they are not enough!
The system was found to be unsound [Wehrman].*

Concurrent Separation Logic (contd)

$$\frac{\Gamma \vdash \{P \star R \wedge B\} C \{Q \star R\}}{\Gamma, r(X) : R \vdash \{P\} \text{with } r \text{ when } B \text{ do } C \text{ od } \{Q\}} \quad \text{CRIT}$$

Owicki-Gries:

- ▶ No variable free in P or Q is changed in any “other process”.

The reference to “other processes” makes this rule non-compositional.

Brookes:

- ▶ $r \notin \text{dom}(\Gamma)$
- ▶ $X \cap \text{owned}(\Gamma) = \emptyset$
- ▶ $\text{free}(R) \cap \text{owned}(\Gamma) = \emptyset$
- ▶ $\text{free}(P, Q) \cap X = \emptyset$

*These conditions are compositional. But they are not enough!
The system was found to be unsound [Wehrman].*



Section 4

SCI for Conditional Critical Regions

SCI for Conditional Critical Regions

- ▶ The traditional solution for passive uses is to mark normal free variables as passive free variables in limited contexts.
- ▶ However, once a free variable is marked as passive, it cannot be used actively within that context.

Fractional permissions to the rescue!

- ▶ we can annotate passively used variables with fractional permissions, and
- ▶ combine permissions to recover the whole (active) variable again.

SCI for Conditional Critical Regions

- ▶ The traditional solution for passive uses is to mark normal free variables as passive free variables in limited contexts.
- ▶ However, once a free variable is marked as passive, it cannot be used actively within that context.

Fractional permissions to the rescue!

- ▶ we can annotate passively used variables with fractional permissions, and
- ▶ combine permissions to recover the whole (active) variable again.

Example

```
{x = 0}  
resource r(x) {true} in begin  
  with r do  
    x := x+1;  
  od  
end  
{x = 2}
```

||

```
with r do  
  x := x+1;  
od
```

Owicki-Gries Example (with auxiliary variables)

```
{x = 0}
a := 0; b := 0;
resource r(x, a, b) {x = a + b} in begin
  {a = 0}                                {b = 0}
  with r do                               with r do
    x := x+1;                               x := x+1;
    a := 1                                   b := 1
  od                                       od
  {a = 1}                                {b = 1}
end
{x = a + b * a = 1 * b = 1}
{x = 2}
```

Question: How can we use a and b outside critical regions?

Owicki-Gries Example (with permissions)

```
{x = 0}
a := 0; b := 0;
resource r(x1, a1/2, b1/2) {x = a + b} in begin
  {a = 0} // owns a1/2
  with r do
    x := x+1;
    a := 1
  od
  {a = 1}
end
  {b = 0} // owns b1/2
  with r do
    x := x+1;
    b := 1
  od
  {b = 1}
{x = a + b * a = 1 * b = 1}
{x = 2}
```

Question: How can we use a and b outside critical regions?

Answer: By using the remaining half permissions for a and b .

Berdine-Reynolds Example

$\{42 \mapsto _ \}$
resource r() **in**
 resource s() **in**

with r **do**
 with s **do** od;
 [42] := 3
od

||

with s **do**
 with r **do** od;
 [42] := 4;
od

$\{42 \mapsto _ \}$

Berdine-Reynolds Example (with auxiliary variables)

$\{42 \mapsto _ \}$

resource $r(v) : R_r$ **in**

resource $s(v) : R_s$ **in**

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

||

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

$\{42 \mapsto _ \}$

$R_r \iff (v = 0 \wedge 42 \mapsto _) \vee (v = 1 \wedge \mathbf{emp})$

$R_s \iff (v = 0 \wedge \mathbf{emp}) \vee (v = 1 \wedge 42 \mapsto _)$

Berdine-Reynolds Example (with permissions)

$\{42 \mapsto _ \}$
resource $r(v^{\frac{1}{2}}) : R_r$ **in**
resource $s(v^{\frac{1}{2}}) : R_s$ **in**

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

\parallel

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

$\{42 \mapsto _ \}$

$R_r \iff (v = 0 \wedge 42 \mapsto _) \vee (v = 1 \wedge \mathbf{emp})$

$R_s \iff (v = 0 \wedge \mathbf{emp}) \vee (v = 1 \wedge 42 \mapsto _)$

Permissions

- ▶ Use real interval $[0, 1]$ with partial addition as \oplus .
- ▶ 1 denotes full permission.
- ▶ 0 denotes no permission.
- ▶ This also generalizes to permission algebras a la Parkinson et al.

Well-formedness judgements

$$x_1^{p_1}, \dots, x_n^{p_n} \vdash E \mathbf{Exp}$$

Variable contexts (Σ)

- ▶ The same variable can have multiple occurrences in Σ :

$$x^{p_{i_1}}, \dots, x^{p_{i_k}}$$

- ▶ But the permissions should be combinable:

$$p_{i_1} \oplus \dots \oplus p_{i_k} \text{ defined}$$

E.g., $x^1, y^{\frac{1}{2}}, x^{\frac{1}{2}} \vdash \dots$ is illegal.

Example rules of well-formedness

$$\frac{}{\Sigma, x^p \vdash x \text{ Exp}} \quad (p \neq 0)$$

$$\frac{\Sigma \vdash E_1 \text{ Exp} \quad \Sigma \vdash E_2 \text{ Exp}}{\Sigma \vdash E_1 + E_2 \text{ Exp}}$$

$$\frac{\Sigma \vdash E_1 \text{ Exp} \quad \Sigma \vdash E_2 \text{ Exp}}{\Sigma \vdash E_1 \xrightarrow{p} E_2 \text{ Assert}} \quad \frac{\Sigma \vdash P_1 \text{ Assert} \quad \Sigma \vdash P_2 \text{ Assert}}{\Sigma \vdash P_1 * P_2 \text{ Assert}}$$

$$\frac{\Sigma, x^1 \vdash P \text{ Assert}}{\Sigma \vdash \exists x. P \text{ Assert}}$$

Structural rules

Contraction rule:

$$\frac{\Sigma, x^p, x^q \vdash \mathcal{S}}{\Sigma, x^{p \oplus q} \vdash \mathcal{S}}$$

Weakening (an admissible rule):

$$\frac{\Sigma \vdash \mathcal{S}}{\Sigma, \Sigma' \vdash \mathcal{S}}$$

Substitution (an admissible rule)

$$\frac{\Sigma \vdash E \mathbf{Exp} \quad \Sigma, x^1 \vdash \mathcal{S}}{\Sigma \vdash \mathcal{S}[E/x]}$$

Well-formedness of commands

$$\Sigma \mid \Gamma \vdash C \text{ Comm}$$

Σ is a variable context: $x_1^{\rho_1}, \dots, x_n^{\rho_n}$

Γ is a resource context: $r_1(\Sigma_1), \dots, r_m(\Sigma_m)$

For a legal context:

- ▶ All the r_i 's should be distinct.
- ▶ $\Sigma, \Sigma_1, \dots, \Sigma_m$ should be legal.

Example: $a^{\frac{1}{2}}, b^{\frac{1}{2}} \mid r(x^1, a^{\frac{1}{2}}, b^{\frac{1}{2}}) \vdash C_1 \parallel C_2 \text{ Comm}$

Example rules for commands

$$\frac{\Sigma, x^1 \vdash E \mathbf{Exp}}{\Sigma, x^1 \mid \Gamma \vdash (x := E) \mathbf{Comm}}$$

$$\frac{\Sigma_1 \mid \Gamma \vdash C_1 \mathbf{Comm} \quad \Sigma_2 \mid \Gamma \vdash C_2 \mathbf{Comm}}{\Sigma_1, \Sigma_2 \mid \Gamma \vdash (C_1 \parallel C_2) \mathbf{Comm}}$$

$$\frac{\Sigma, \Sigma_0 \mid \Gamma \vdash C \mathbf{Comm}}{\Sigma \mid \Gamma, r(\Sigma_0) \vdash (\mathbf{with } r \mathbf{ do } C \mathbf{ od}) \mathbf{Comm}}$$

- ▶ *There are no side conditions for the parallel rule!*
A bit deceptive because Σ_1, Σ_2 should be legal.
- ▶ A critical region unlocks the resource's context and provides it to the body of **with** .

Section 5

SCI for Concurrent Separation Logic

Program logic with SCI

$$\Sigma \mid \Gamma \vdash \{P\} C \{Q\}$$

requires well-formedness:

- ▶ $\Sigma \vdash P$ **Assert** and $\Sigma \vdash Q$ **Assert**.
- ▶ $\Sigma \mid \Gamma \vdash C$ **Comm**

For example:

$$a^{\frac{1}{2}}, b^{\frac{1}{2}} \mid r(x^1, a^{\frac{1}{2}}, b^{\frac{1}{2}}) \vdash \{a = 0 \star b = 0\} C_1 \parallel C_2 \{a = 1 \star b = 1\}$$

Examples of Logic rules

$$\text{ASSIGN} \quad \frac{\Sigma, x^1 \vdash E \text{ Exp} \quad \Sigma, x^1 \vdash P \text{ Assert}}{\Sigma, x^1 \mid \Gamma \vdash \{P[E/x]\} x := E \{P\}}$$

$$\text{PAR} \quad \frac{\Sigma_1 \mid \Gamma \vdash \{P_1\} C_1 \{Q_1\} \quad \Sigma_2 \mid \Gamma \vdash \{P_2\} C_2 \{Q_2\}}{\Sigma_1, \Sigma_2 \mid \Gamma \vdash \{P_1 \star P_2\} C_1 \parallel C_2 \{Q_1 \star Q_2\}}$$

- ▶ In *PAR*, the well-formedness of Σ_1, Σ_2 is equivalent to the Owicki-Gries side condition “ C_i does not modify $\text{free}(P_j, Q_j)$ (for $j \neq i$).”

Examples of Logic rules (contd)

CRIT rule for critical regions:

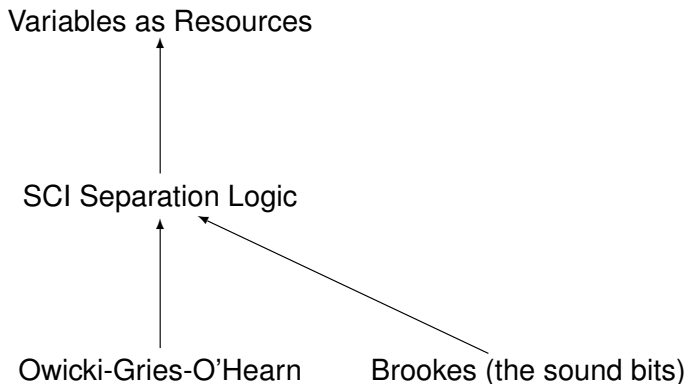
$$\frac{\Sigma \vdash P \text{ Assert} \quad \Sigma \vdash Q \text{ Assert} \quad \Sigma, \Sigma_0 \mid \Gamma \vdash \{P \star R\} C \{Q \star R\}}{\Sigma \mid \Gamma, r(\Sigma_0): R \vdash \{P\} \text{ with } r \text{ do } C \text{ od } \{Q\}}$$

- ▶ Since P and Q are well-formed in the context Σ , it is obvious that no “other process” can modify the variables in Σ .
- ▶ However, “this process” can modify them, because Σ is part of the context for C .
- ▶ Thus, we have a *compositional formulation* of the Owicki-Gries side conditions.

Section 6

Comparisons

Comparisons



Comparison with “Variables as Resource” systems

[Parkinson et al.]

“Variable as resource” systems treat variable usage in *assertions* instead of the syntax.

Our rules can be translated into “Variables as resource” logics.
(Hence, the latter are more general.)

$$\Sigma = (x_1^{p_1}, \dots, x_n^{p_n}) \rightsquigarrow O_\Sigma \equiv \mathbf{own}_{p_1}(x_1) \star \dots \star \mathbf{own}_{p_n}(x_n)$$

$$\Sigma \mid \Gamma \vdash \{P\} C \{Q\} \rightsquigarrow \Gamma \vdash \{O_\Sigma \wedge P\} C \{O_\Sigma \wedge Q\}$$

But “Variables as resource” logics are strange in practice:

- ▶ Program variables cannot be treated as logical variables.
- ▶ Substitution is not always legal.
- ▶ Well-formedness issues pollute the logic,
 - ▶ $E = E$ is not universally true.
 - ▶ $\neg(E_1 = E_2)$ and $E_1 \neq E_2$ are not the same thing.

Section 7

Semantics

Semantics

- ▶ SCI is something like a type system.
- ▶ So, we would expect it to streamline the denotational semantics, and rule out unwanted behaviours.
- ▶ We give a “Church typing” semantics for the system, using Brookes’s framework of action traces and local enablings.
- ▶ Brookes’s original semantics treats heap via “local enabling,” but the store is treated globally.
- ▶ In contrast, we treat **both the store and the heap** via local enabling.

Theorem: The trace set T of every command $\Sigma \mid \Gamma \vdash C$ **Comm** satisfies $T :: \Sigma \mid \Gamma \longrightarrow \Sigma \mid \Gamma$. (Ignoring infinite traces.)

Section 8

Permission Inference

Permission Inference

- ▶ If we think of the permissions of variables as a form of types, it makes sense to ask whether they can be automatically inferred.
- ▶ Given a program proof where the resource declarations do not have any variables or their permissions associated with them, can we infer the permission contexts that must be associated with them?
- ▶ **Soundness**: Any permissions inferred must give a valid proof outline in the proof system.
- ▶ **Completeness**: If there is any way to fill in permission contexts to give a valid proof outline, the inferred permissions must cover them.

Berdine-Reynolds Example

$\{42 \mapsto _ \}$

resource $r(v^{\frac{1}{2}}) : R_r$ **in resource** $s(v^{\frac{1}{2}}) : R_s$ **in**

begin

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

end

$\{42 \mapsto _ \}$

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

||

- ▶ Permission inference has to determine how the permissions can be distributed so that the assignments to v are legal.
- ▶ The possible owners of permissions are:
 - ▶ resources, and
 - ▶ the process running the code (“self”).

Berdine-Reynolds Example

$\{42 \mapsto _ \}$

resource $r : R_r$ **in resource** $s : R_s$ **in**

begin

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

end

$\{42 \mapsto _ \}$

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

||

- ▶ Permission inference has to determine how the permissions can be distributed so that the assignments to v are legal.
- ▶ The possible owners of permissions are:
 - ▶ resources, and
 - ▶ the process running the code (“self”).

Algorithm - Phase 1 (Inside-out)

```
resource  $r : R_r$  in resource  $s : R_s$  in
begin
  with  $r$  do
    with  $s$  do  $v := 0$  od;
    {  $v = 0 \star R_r$  }
    [42] := 3
  od
end
||
with  $s$  do
  with  $r$  do  $v := 1$  od;
  {  $v = 1 \star R_s$  }
  [42] := 4;
od
```

For [42] := 3, we don't need any variable permissions!

Algorithm - Phase 1 (Inside-out)

```
resource  $r : R_r$  in resource  $s : R_s$  in
begin
  with  $r$  do
    with  $s$  do  $v := 0$  od;
     $\{v = 0 \star R_r\}$ 
     $[42] := 3$ 
  od
end
||
with  $s$  do
  with  $r$  do  $v := 1$  od;
   $\{v = 1 \star R_s\}$ 
   $[42] := 4;$ 
od
```

the process (“self”) must have the full permission.

self : $[v : 1]$

Algorithm - Phase 1 (Inside-out)

resource $r : R_r$ **in resource** $s : R_s$ **in**
begin

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

end

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

||

we conclude that the full permission could have come from self
and s :

$\text{self} : [v : \alpha_0], s : [v : \gamma]$ where $\alpha_0 + \gamma = 1$

Algorithm - Phase 1 (Inside-out)

resource $r : R_r$ **in resource** $s : R_s$ **in**
begin

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

end

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

||

we conclude that the permission of α_0 could have come from self and r :

self : $[v : \alpha]$, $r : [v : \beta]$, $s : [v : \gamma]$ where $\alpha + \beta + \gamma = 1$

Algorithm - Phase 1 (Inside-out)

resource $r : R_r$ **in resource** $s : R_s$ **in**

begin

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

end

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

||

left: $\text{self} : [v : \alpha], r : [v : \beta], s : [v : \gamma]$ **where** $\alpha + \beta + \gamma = 1$

right: $\text{self} : [v : \alpha'], r : [v : \beta'], s : [v : \gamma']$ **where** $\alpha' + \beta' + \gamma' = 1$

Algorithm - Phase 1 (Inside-out)

resource $r : R_r$ **in resource** $s : R_s$ **in**

begin

with r **do**

with s **do** $v := 0$ **od**;

$\{v = 0 \star R_r\}$

$[42] := 3$

od

end

with s **do**

with r **do** $v := 1$ **od**;

$\{v = 1 \star R_s\}$

$[42] := 4$;

od

||

left: $\text{self} : [v : \alpha]$, $r : [v : \beta]$, $s : [v : \gamma]$ **where** $\alpha + \beta + \gamma = 1$

right: $\text{self} : [v : \alpha']$, $r : [v : \beta']$, $s : [v : \gamma']$ **where** $\alpha' + \beta' + \gamma' = 1$

parallel composition:

$$\alpha = \alpha' = 0$$

$$\beta = \beta' \quad \text{and} \quad \gamma = \gamma'$$

self cannot have any non-zero permission for v in the two processes.

Algorithm - phase 1

In the first phase of the algorithm, we traverse the program *inside-out* and determine the owners that can contribute permissions for the variable assignments.

- ▶ $v := 0$ - the process (“self”) must have the full permission.

$\text{self} : [v : 1]$

- ▶ **with s do $v := 0$ od** - we conclude that the full permission could have come from self and s:

$\text{self} : [v : \alpha_0], s : [v : \gamma] \quad \text{where } \alpha_0 + \gamma = 1$

- ▶ $[42] := 3$ - we don't need any variable permissions!
- ▶ **with r do**

**with s do $v := 0$ od; $[42] := 3$
od**

we conclude that the permission of α_0 could have come from self and r:

$\text{self} : [v : \alpha], r : [v : \beta], s : [v : \gamma] \quad \text{where } \alpha + \beta + \gamma = 1$

Algorithm - phase 1 (contd)

Consider the parallel composition $P_1 \parallel P_2$.

- ▶ For P_1 , we have noted:

$$\text{self} : [v : \alpha], r : [v : \beta], s : [v : \gamma] \quad \text{where } \alpha + \beta + \gamma = 1$$

- ▶ For P_2 , we have a symmetric situation:

$$\text{self} : [v : \alpha'], r : [v : \beta'], s : [v : \gamma'] \quad \text{where } \alpha' + \beta' + \gamma' = 1$$

- ▶ For $P_1 \parallel P_2$ to be legal, we must have:

$$\alpha = 0 = \alpha'$$

$$\beta = \beta'$$

$$\gamma = \gamma'$$

self cannot have any non-zero permission for v in P_1 and P_2 .

Algorithm - Phase 2

- ▶ The main job of phase 1 is to discover all the 0 permissions that are required by the rules.
- ▶ At the end of phase 1, we conclude that the overall annotated program must be of the form:

resource $r(v^\beta) : R_r$ **in** **resource** $s(v^\gamma) : R_s$ **in**
 $P_1 \parallel P_2$

where $\beta + \gamma = 1$.

- ▶ In phase 2, we traverse the program *outside-in* and choose any non-zero permissions for β and γ .
- ▶ Since the phase 1 has already discovered all the necessary zero permissions, any non-zero permissions will do. This is a “maximally permissive” permission assignment.
- ▶ We also check that all the read-only occurrences have non-zero permissions, e.g., the assertion

$\{v = 0 \star R_r\}$

Actual algorithm

- ▶ It turns out that we don't actually need to manipulate permission variables in the algorithm. All we need is to keep track of which owners can contribute permissions.
- ▶ Permission restriction:

$$\Phi : \text{Vars} \rightarrow \mathcal{P}(\text{Owners})$$

- ▶ Permission assignment:

$$\Delta : \text{Vars} \rightarrow \text{Owners} \rightarrow [0, 1]$$

- ▶ Δ *satisfies* Φ if, for every $v \in \text{dom}(\Phi)$,
 - ▶ the permissions in Δ for all owners in $\Phi(v)$ sum up to 1.So, the owners not in $\Phi(v)$ have only 0 permissions for v in Δ .

Summary

- ▶ We have produced a clean, simple, compositional system of proof rules for Concurrent Separation Logic.
- ▶ The system is expressive, fitting somewhere between Owicki-Gries-O'Hearn rules and “Variables as resource” rules.
- ▶ It is sound and has a direct representation in the semantics.
- ▶ It is possible to infer the required permission contexts automatically in linear time, without any global program analysis.

Further work

- ▶ Implementation.
- ▶ Integration with type systems.
- ▶ Extensions to procedures and objects.

Thank you for your attention!

Further work

- ▶ Implementation.
- ▶ Integration with type systems.
- ▶ Extensions to procedures and objects.

Thank you for your attention!

An aside on natural deduction

A natural deduction starts with assumptions and applies rules to derive conclusions:

$$\begin{array}{ccc} A_1 & \dots & A_n \\ \vdots & & \vdots \\ & \mathcal{S} & \end{array}$$

For the sake of clarity on how the assumptions are handled, we write it in sequent form:

$$A_1, \dots, A_n \vdash \mathcal{S}$$

So the parallel rule is really saying:

$$\frac{\begin{array}{cc} \Sigma_1 & \Sigma_2 \\ \vdots & \vdots \\ C_1 \text{ **Comm** & C_2 \text{ **Comm** \end{array}}{(C_1 \parallel C_2) \text{ **Comm**}}$$

Comparison with Owicki-Gries-O'Hearn system

All Owicki-Gries-O'Hearn proof outlines can be transformed to our system.

Every resource declaration

resource $r(x_1, \dots, x_n)$ **in** C

must be annotated with permissions for the owned variables.

1. If x_i occurs only inside critical regions:
 - ▶ if it is modified there, annotate it as x_i^1 .
 - ▶ if it is not modified there, annotate it with a possibly partial permission p .
2. If x_i occurs outside critical regions, it can only do so in a *single* process and it must be *passively* used.
 - ▶ Annotate the resource with x_i^p (for some partial permission p).
 - ▶ Give the process $x_i^{p'}$ (where $p \oplus p' = 1$).

Comparison with Brookes's original system (TCS 2007)

Brookes allows every resource to deal with two sets of variables:

resource $r(x_1, \dots, x_n) : R \text{ in } C$

- ▶ **owned** $(r) = \{x_1, \dots, x_n\}$
- ▶ **free** (R) is the set of free variables in the resource invariant, which can include more variables than **owned** (r) .

Example: resource $r(x) : \{x = a + b\} \text{ in } C_1 \parallel C_2$

Rewrite the resource declaration as:

resource $r(x_1^1, \dots, x_n^1, y_1^{p_1}, \dots, y_m^{p_m}) : R \text{ in } C$

- ▶ Variables in **owned** (r) are fully owned by the resource.
- ▶ The additional variables in **free** (R) are partially owned by the resource.

However not all Brookes's proof outlines can be translated.

Unsoundness in Brookes's rules

[Ian Wehrman]

```
x := a;  
resource r(x) {x = a} in  
begin  
  {true}  
  with r do  
    t := x  
  od  
  {t = a}  
  with r do  
    x := t  
  od  
  {true}  
end  
{x = a}
```

||

```
{true}  
with r do  
  x := x+1;  
  a := a+1  
od  
{true}
```

Semantics - Action traces

Brookes's action traces:

$$\lambda ::= \delta \mid x = v \mid x := v \mid [l] = v \mid [l] := v \mid \text{try}(r) \mid \text{acq}(r) \mid \text{rel}(r) \mid \text{abort}$$

Actions are enabled in contexts, and may transform them, giving a notion of “types” for actions:

$$x = v \ :: \ \Sigma \mid \tilde{\Gamma} \longrightarrow \Sigma \mid \tilde{\Gamma} \quad \text{iff} \quad x^p \in \Sigma \text{ for some } p$$
$$x := v \ :: \ \Sigma \mid \tilde{\Gamma} \longrightarrow \Sigma \mid \tilde{\Gamma} \quad \text{iff} \quad x^1 \in \text{norm}(\Sigma)$$
$$\text{acq}(r) \ :: \ \Sigma \mid \tilde{\Gamma}, r(\Sigma_0) \longrightarrow \Sigma, \Sigma_0 \mid \tilde{\Gamma}, [r(\Sigma_0)]$$
$$\text{rel}(r) \ :: \ \Sigma, \Sigma_0 \mid \tilde{\Gamma}, [r(\Sigma_0)] \longrightarrow \Sigma \mid \tilde{\Gamma}, r(\Sigma_0)$$

Semantics - Local state transitions

- ▶ A **local state** is a triple (s, h, A) consisting of *store* s , *heap* h and a set of *acquired resources* A .
- ▶ Local states have **types**, $(s, h, A) :: \Sigma \mid \tilde{\Gamma}$ if
 - ▶ $\text{dom}(s)$ is the set of variables listed in Σ .
 - ▶ A is a subset of the resources marked busy in $\tilde{\Gamma}$.
 - ▶ h is unrestricted.
- ▶ Every action $\lambda :: \Sigma \mid \tilde{\Gamma} \longrightarrow \Sigma' \mid \tilde{\Gamma}'$ has a **transition relation** $\xrightarrow{\lambda}$ from local states of type $\Sigma \mid \tilde{\Gamma}$ to local states of type $\Sigma' \mid \tilde{\Gamma}'$.
- ▶ Example:

$$x := v :: \Sigma \mid \tilde{\Gamma} \longrightarrow \Sigma \mid \tilde{\Gamma} \quad \text{iff} \quad x^1 \in \text{norm}(\Sigma)$$
$$(s, h, A) \xrightarrow{x:=v} (s[x \mapsto v], h, A)$$

Semantics - Resource actions

► Acquire action:

$$\begin{aligned} \text{acq}(r) &:: \Sigma \mid \tilde{\Gamma}, r(\Sigma_0): R \longrightarrow \Sigma, \Sigma_0 \mid \tilde{\Gamma}, [r(\Sigma_0): R] \\ (s, h, A) &\xrightarrow{\text{acq}(r)} (s \cdot s_0, h \cdot h_0, A \cup \{r\}) \\ &\text{iff } s \cdot s_0 \text{ defined, } h \perp h_0, (s_0, h_0) \models R \end{aligned}$$

$s \cdot s_0$ is defined if the two states agree on all common variables.

► Release action:

$$\begin{aligned} \text{rel}(r) &:: \Sigma, \Sigma_0 \mid \tilde{\Gamma}, [r(\Sigma_0): R] \longrightarrow \Sigma \mid \tilde{\Gamma}, r(\Sigma_0): R \\ (s \cdot s_0, h \cdot h_0, A \uplus \{r\}) &\xrightarrow{\text{rel}(r)} (s, h, A) \text{ iff } (s_0, h_0) \models R \\ (s, h, A) &\xrightarrow{\text{rel}(r)} \mathbf{abort} \text{ iff } \forall h_0 \subseteq h. \neg(s, h_0) \models R \end{aligned}$$

Soundness

- ▶ A judgement $\Sigma \mid \Gamma \vdash \{P\} C \{Q\}$ is **valid** iff, for
 - ▶ all traces $\alpha \in \llbracket C \rrbracket_{\Sigma \mid \Gamma}$,
 - ▶ all states (s, h, \emptyset) of type $\Sigma \mid \Gamma$ satisfying P , and
 - ▶ all states σ' of type $\Sigma \mid \Gamma$ such that $(s, h, \emptyset) \xrightarrow{\alpha} \sigma'$,the state σ' satisfies Q .
- ▶ **Theorem:** All provable judgements $\Sigma \mid \Gamma \vdash \{P\} C \{Q\}$ are valid.