RGsep and Local Rely-Guarantee

Arthur Charguéraud

Concurrent Program Logics Seminar

Max Planck Institute for Software Systems Kaiserslautern, 2011/06/30

Overview

Separation Logic
– local reasoning (frame)
– no concurrency

Rely Guarantee
– global reasoning
– transitions (pairs of states)

Concurrent Separation Logic
– invariants on shared resources
– lots of auxiliary variables

SAGL
– shared and private parts
– primitive commands atomic
– lacks standard frame rule

RGSep
– two layers of assertions
– box P for a shared resource
– private resources are local

Local Rely Guarantee
– rely and guarantees are local
– single-layer assertion language
– uses invariant-fence actions

Plan of this talk

- **RGSep specification language**
- **Interference**
- **Stability**
- **RGSep reasoning rules**
- **LRG specification language**
- **LRG invariant-fenced actions**
- **LRG reasoning rules**

RG-sep assertion language

The logic is made of a layer built on top of Separation Logic (SL):

$$p, q, r ::= P \mid \boxed{P} \mid p * q \mid p \wedge q \mid p \vee q \mid \exists x.\, p \mid \forall x.\, p$$

$$P, Q, S ::= \mathbf{false} \mid \mathbf{emp} \mid e = e' \mid e \mapsto e' \mid \exists x.\, P \mid P \Rightarrow Q \mid P * Q \mid P \mathrel{-\!\!\circledast} Q$$

$\rightarrow$ "box P" denotes a shared resource described by heap prediate P

$\rightarrow$ boxes always occur in positive position in assertion formulae

$\rightarrow$ "p" interpreted as a triple (l,s,i): local state, shared state, logical state

Interpretation of separating conjunction in the outer layer:

– multiplicative over local state: P $*$ Q denotes disjoint union

– additive over shared state: [P] $*$ [Q] is defined as [P$\wedge$Q]

box

Observe that P $*$ Q can be interpreted using either of the two stars

$\rightarrow$ this is not a problem because the two interpretation are equivalent

Recall Rely-Guarantee

Form of the judgment:

R; G |- {p} C {q} also written **C sat (p,R,G,q)**

– **C** is a command
– **R** is a relation describing the interference caused by the environment
– **G** is a relation describing how the command changes the shared state
– **p** and **q** are the pre- and post-conditions for local and shared states

Question 1: how to express a rely/guarantee relation using SL?

Question 2: how to check that pre/post-conditions are stable w.r.t. rely?

Example:

$$\frac{\vdash C \ \mathbf{sat}\ (P, \{\}, \{\}, Q) \quad (P \rightsquigarrow Q) \subseteq G \quad \boxed{Q} \text{ stable under } R}{\vdash (\mathbf{atomic}\{C\})\ \mathbf{sat}\ (\boxed{P}, R, G, \boxed{Q})}$$

Description of interference

The modification of a piece of shared state is described by an action:

$$\mathbf{P \sim\!> Q}$$

$\rightarrow$ interpreted as a binary relation over states, defined as the set of pairs of state of the form $(h_p \oplus h_f, h_q \oplus h_f)$ where h_p satisfies P and h_q satisfies Q.
(ignore the valuation of the logical variables for the time being)

A rely R or a guarantee G is represented as a set of actions.

$\rightarrow$ interpreted as the set of pairs that belong to the reflexive-transitive closure of the binary relations associated with the actions in the set.

An action $P \sim\!> Q$ is allowed by a guarantee G when

$$(P \sim\!> Q) \subseteq G$$

$\rightarrow$ meaning that the relation described by the action is included in the relation described by the guarantee

Other rules for establishing interference

$$\frac{}{x \mapsto y \rightsquigarrow x \mapsto y \subseteq G}\ \text{G-Exact} \qquad \frac{P \rightsquigarrow Q \in G}{P \rightsquigarrow Q \subseteq G}\ \text{G-Ax}$$

$$\frac{P_1 \rightsquigarrow S * Q_1 \subseteq G \quad P_2 * S \rightsquigarrow Q_2 \subseteq G}{P_1 * P_2 \rightsquigarrow Q_1 * Q_2 \subseteq G}\ \text{G-Seq} \qquad \frac{P \rightsquigarrow Q \subseteq G}{P[e/x] \rightsquigarrow Q[e/x] \subseteq G}\ \text{G-Sub}$$

$$\frac{\vDash_{\mathrm{SL}} P' \Rightarrow P \quad P \rightsquigarrow Q \subseteq G \quad \vDash_{\mathrm{SL}} Q' \Rightarrow Q}{P' \rightsquigarrow Q' \subseteq G}\ \text{G-Cons} \qquad \frac{(P * F) \rightsquigarrow (Q * F) \subseteq G}{P \rightsquigarrow Q \subseteq G}\ \text{G-CoFrm}$$

Example for Cons: if the action (x → even) ~> (x → –) in the premise is allowed by the guarantee G, then the action (x → 6) ~> (x → 3) in the conclusion is also allowed by the guarantee G, because the latter is an action that is a particular case of the former.

Analysis of CoFrm: if the action (P∗F) ~> (Q∗F) in the premise is allowed by the guarantee G, then the action P ~> Q in the conclusion is also allowed by the guarantee G, because an action that changes P into Q and leaves all the rest unchanged can also be viewed as an action that changes P∗F into Q∗F and leaves all the rest unchanged.

Description of stability for SL assertions

Stability of a SL assertion with respect to a rely (relation over states):

sem_stable(P,R) also written **P;R ==> P**

→ interpretation: if **h** satisfies **P**, and if the transition **(h,h')** belongs to **R**, then **h'** also satisfies **P**.

Syntactic technique for establishing stability of SL assertions:

$\mathbf{sem\_stable(P, [|\ P_1\sim>Q_1, ..., P_n\sim>Q_n\ |]\ )}$

$\Leftrightarrow \quad \forall \mathbf{i},\ \mathbf{sem\_stable(P, [|\ P_i\sim>Q_i\ |])}$

$\Leftrightarrow \quad \forall \mathbf{i},\ ((\mathbf{P_i} -\otimes \mathbf{P}) * \mathbf{Q_i}) \Rightarrow \mathbf{P}$

relation that corresponds to the intepretation of the set of actions

→ $\mathbf{P_i} -\otimes \mathbf{P}$ describes a heap such that there exists another heap satisfying $\mathbf{P_i}$ such that if we take the disjoint union of both the result satisfies **P**

→ intuitively, we start from **P**, remove $\mathbf{P_i}$ then add $\mathbf{Q_i}$ and we should get **P**

Description of stability for RGSep assertions

Stability of a RGSep assertion with respect to a rely:

p stable under R

$\rightarrow$ **Definition:** (recall that $p, q, r ::= P \mid \boxed{P} \mid p * q \mid p \wedge q \mid p \vee q \mid \exists x.\, p \mid \forall x.\, p$)

– **P stable under R** always holds

– **[P] stable under R** iff **sem_stable(P,R)**

– **p1*p2 stable under R** iff **p1 stable under R** and **p2 stable under R**

... and similarly for other constructors

$\rightarrow$ **Interpretation:**

local state, shared state, logical state

– assume p stable under R holds

– assume the state (l,s,i) satisfies p, that is, (l,s,i) |= p

– assume s' is a shared state such that (s,s') $\in$ R

Then the state (l,s',i) also satisifies p, that is, (l,s',i) |= p

Towards a rule for atomic commands

A simple (and limited) version of the rule for atomic commands:

$$\frac{\vdash C \text{ sat } (P, \{\}, \{\}, Q) \qquad (P \rightsquigarrow Q) \subseteq G \qquad \boxed{Q} \text{ stable under } R}{\vdash (\mathbf{atomic}\{C\}) \text{ sat } (\boxed{P}, R, G, \boxed{Q})}$$

$\rightarrow$ the shared states **P** and **Q** becomes private state inside the atomic section

$\rightarrow$ the transition **P ~> Q** made on the shared state must satisfy the guarantee

$\rightarrow$ the new post-condition **Q** needs to be stable under the rely **R**

The CONCUR'07 paper includes a generalized version of this rules that

– adds the possiblity to modify only a portion of the shared state

– adds the possibility for the atomic section to access the local state

– allow to pull existential quantifiers out from the shared state

Rules for atomic commands

Vafeiadis' dissertation instead includes two more primitive rules

(1)

$$\frac{\vdash \langle C \rangle \ \mathbf{sat}\ (p, \emptyset, G, q) \qquad p \text{ stable under } R \quad q \text{ stable under } R}{\vdash \langle C \rangle \ \mathbf{sat}\ (p, R, G, q)} \ (\text{ATOMR})$$

$\longrightarrow$ the pre- and post- condition of the atomic section must be stable under **R**

$\longrightarrow$ in such a case, the rely can be made empty in the analysis of the section

not needed if we don't include the conjunction rule

(2)

$$\frac{P, Q \text{ precise} \qquad \vdash C\ \mathbf{sat}\ (P * P', \emptyset, \emptyset, Q * Q') \qquad (P \leadsto Q) \subseteq G}{\vdash \langle C \rangle \ \mathbf{sat}\ (\boxed{P * F} * P', \emptyset, G, \boxed{Q * F} * Q')} \ (\text{ATOM})$$

$\longrightarrow$ the shared states **P** and **Q** becomes private state inside the atomic section

$\longrightarrow$ the transition **P ~> Q** made on the shared state must satisfy the guarantee

$\longrightarrow$ the shared state **F** is not involved in the analysis of the atomic section

Other interesting rules

***) Rule for commands that do not access the shared state:**

$$\frac{\vdash_{\mathsf{SL}} \{P\}\ c\ \{Q\}}{\vdash c\ \mathbf{sat}\ (P, R, G, Q)}\ (\textsc{Prim})$$

***) Frame rule:**

$$\frac{\vdash C\ \mathbf{sat}\ (p, R, G, q) \qquad r \text{ stable under } (R \cup G)}{\vdash C\ \mathbf{sat}\ (p * r, R, G, q * r)}\ (\textsc{Frame})$$

$\rightarrow$ if **r** mentions the shared state, then it must be stable under interference

***) Parallel**

$$\frac{\begin{array}{c}\vdash C_1\ \mathbf{sat}\ (p_1, R \cup G_2, G_1, q_1) \\ \vdash C_2\ \mathbf{sat}\ (p_2, R \cup G_1, G_2, q_2)\end{array}}{\vdash (C_1 \| C_2)\ \mathbf{sat}\ (p_1 * p_2, R, G_1 \cup G_2, q_1 * q_2)}$$

$\rightarrow$ $\mathbf{C_1}$ can undergo interference from the environment **(R)** and from the other thread **($\mathbf{G_2}$)** –recall that the rely of one is the guarantee of the other.

Overview of the soundness proof for RGSep

1) Model a heap as a partial commutative cancellative monoid

2) Model a structured heap **σ** as a triple: local, shared, and environment states

3) Interpret each comand as a binary relation over heaps

4) Introduce a small-step reduction relation annotated with a set of possible interference; reduction steps are annotated with a label indicating whether the action is that of the program or that of the environment

5) Define **(C, σ, R) guard$_n$ G** to express that the execution of **C** in a state **σ** under possible interference **R** satisfies the guarantee **G** for at least **n** steps

6) Define **|= C sat (p,R,G,Q)** to express that for any **R'⊆R**, the execution of **C** in a state **σ** satisfying **p** under possible intereference **R'** satisfies the garantee **G** for an arbitrary number of steps, i.e. **∀n, (C, σ, R) guard$_n$ G,** and, if the command terminates, the final result satisfies the post-condition **Q**

7) Soundness theorem: if **|- C sat (p,R,G,Q)** then **|= C sat (p,R,G,Q)**

LRG: motivation

Goal is to allow rely and guarantees to be local to a sub-computation:

– ability to hide from a computation the shared resources that it does not use

– ability to declare a rely and a guarantee locally in a given computation

$\rightarrow$ it is in fact necessary to support local declarations of rely and guarantees for reasoning about dynamically-allocated shared memory cells

Technical ingredient #1: give a meaning to $\mathbf{R*R'}$ and $\mathbf{G*G'}$

$\rightarrow$ $\mathbf{a * a'}$ is interpreted as the set of pairs of the form $\mathbf{(h_1 \oplus h_2, h_1' \oplus h_2')}$ where $\mathbf{(h_1, h_1')}$ is a pair in the interpretation of the action **a** and $\mathbf{(h_2, h_2')}$ is a pair in the interpretation of the action **a'**

Technical ingredient #2: invariant-fenced actions

$\rightarrow$ it is needed for deriving that $\mathbf{p_1}$ **stable under** $\mathbf{a_1}$ and $\mathbf{p_2}$ **stable under** $\mathbf{a_2}$ implies $\mathbf{(p_1 * p_2)}$ **stable under** $\mathbf{(a_1 * a_2)}$ --notation in the paper is "Sta(p,a)"

Technical ingredient #3: coming back to a single-layer logic

Language of actions in LRG

An action a is a syntactic object whose interpretation, written [|a|], is a set of pairs of states. (A rely is an action, and a guarantee is an action.)

a ::= here "action" can be "an atomic action" or "a set of actions"

- **p~>q** pairs of heap whose fst satisfies **p** and snd satisfies **q** (same as $p \ltimes q$)
- **[p]** preserves a heap that satisfies **p** (the heap cannot change at all)
- **a ∗ a'** disjoint union of two actions, as explained earlier
- **∃x.a** used to quantify logical variables in actions (not detailed here)

Inclusion between actions: a ⇒ a' holds if the set of pairs denoted by the action **a** is included in the set of pairs denoted by **a'**, that is, **[|a|] ⊆ [|a'|]**

Useful shorthands:

- **Emp** the empty action takes empty heap to empty heap, i.e. **emp~>emp**
- **Id** the identity action takes any heap to itself, i.e. **[true]**
- **True** the true action takes any heap to any other, i.e. **true ~> true**

Stability in LRG

Definition: **p stable under a** ---written in the paper "Sta(p,a)"

$\Leftrightarrow$ for any state **h** satisfying **p**, for any pair **(h,h')** that belongs to the interpretation of the action **a**, the state **h'** also satisfies **P**.

To frame out parts of the rely/garantee, we need a result of the form:

if p_1 stable under a_1 and p_2 stable under a_2 then $(p_1 * p_2)$ stable under $(a_1 * a_2)$

$\rightarrow$ but this result is incorrect without appropriate side-conditions

$\rightarrow$ example: p_1 describes a memory cell and a_2 is an action over that cell

Before looking for a side-condition that will make the result become correct, let's see why we need this result for the frame rule to work.

Towards a frame rule

The shape of the frame rule that we are looking for:

$$\frac{\mathbf{R,G \mid\!- \{p\}\ C\ \{q\} \qquad m\ stable\ under\ R'}}{\mathbf{R{*}R';\ G{*}G' \mid\!- \{p{*}m\}\ C\ \{q{*}m\}}}$$

(ignore G' for now)

→ The intuition of the frame rule is that if we have the proof of the first premise then we redo this proof with a larger set of resources and relies.

→ However, deny-guarantee reasoning involves side conditions of the form p stable under R.

→ So, if we want to redo the proof in a context extended with m resources and R' guarantees, we need to show p∗m stable under R∗R'.

Intuitively, we need to enforce the fact that R' only contains actions that are specific to m (things would go wrong if R' talked about data in p)

Fences on actions

A fence is used to give a precise boundary to an action, so that we can know for sure that an action does not refer to a resource outside this boundary

Definition: an action a is fenced by an invariant I, written I ► a, iff

– I is precise, that is, any state has at most one sub-state satisfying I

– the action a covers the preservation of a state satisfying I, i.e. [I] $\Rightarrow$ a

– I holds over begin and end state of any transition in a, i.e. a $\Rightarrow$ (I~>I)

$\rightarrow$ Typically, a is a rely or a garantee, so we write **I ►R** or **I ►G**

$\rightarrow$ Moreover, we write **I ►{R,G}** for the conjunction of two such facts

Example: $\mathbf{R = (x \rightarrow \text{List } L) \sim> \exists A. (x \rightarrow \text{List } A::L)}$

$\mathbf{G = (x \rightarrow \text{List } L) \sim> \exists B. \exists Q. (L = B::Q * x \rightarrow \text{List } Q)}$

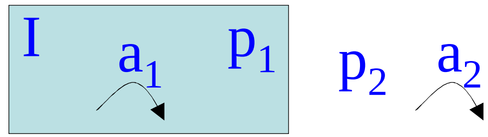
$\mathbf{I = \exists L. x \rightarrow \text{List } L}$

Composition with a fenced action

We have the following (asymmetric!) lemma:

if p_1 stable under a_1 and p_2 stable under a_2 and $\mathbf{p_1 \Rightarrow I}$ and $\mathbf{I} \blacktriangleright \mathbf{a_1}$

then $(p_1 * p_2)$ stable under $(a_1 * a_2)$



Proof that "$\mathbf{p_2}$ is stable under $\mathbf{a_1}$":

$\rightarrow$ the begin and end of transitions in a_1 satisfy I

$\rightarrow$ p_2 is disjoint from p_1, and since p_1 satisfies I, we have p_2 disjoint from I

$\rightarrow$ thus, the action of a_1 (inside I) does not affect p_2 (disjoint from I)

Proof that "$\mathbf{p_1}$ is stable under $\mathbf{a_2}$":

$\rightarrow$ a_1 and a_2 have disjoint begin and end states

$\rightarrow$ the begin and end of transitions in a_1 satisfy I

$\rightarrow$ thus, the transitions in a_2 only affect states that are disjoint from I

$\rightarrow$ since p_1 satisfies I, we therefore conclude that p_1 is not affected by a_2

Back to the frame rule

Recall:

$$\frac{\mathbf{R,G \mid\!- \{p\}\ C\ \{q\} \qquad m\ stable\ under\ R'}}{\mathbf{R{*}R';\ G{*}G' \mid\!- \{p{*}m\}\ C\ \{q{*}m\}}}$$

In order to redo the proof in a context extended with m resources and R' guarantees, we want p stable under R to imply p∗m stable under R∗R'.

According to the lemma we have just seen, it suffices to find an invariant I such that $\mathbf{m \Rightarrow I}$ and $\mathbf{I \blacktriangleright R'}$

→ In other words, we need that the framed resource m satisfies a precise invariant I such that:

– R' contains only transitions whose begin state and end state satisfy I

– and R' contains the identity transition over states satisfying I

Judgment extended with invariants

The precise invariants (e.g. I) now play a role in the reasoning. The judgment needs to be extended to:

$$R; G; I \mid\!- \{p\}\ C\ \{q\}$$

$\rightarrow$ **R** and **G** and **I** are only used to specify the shared state

$\rightarrow$ **p** and **q** specify the whole state

The system enforces two properties:

$\rightarrow$ **R** and **G** are fenced by the invariant **I**, that is, **I ►R** and **I ►G**

$\rightarrow$ the shared state in **p** and **q** is covered by the invariant **I**, so **p ⇒ I*true**

q => I* true

Interestingly, the invariant does not receive any real interpretation in the final soundness theorem. It's only used in the lemmas justifying the correctness of the frame and hide rules.

Frame rule

Frame rule to frame out a shared resource r:

$$\frac{R;\ G;\ I \vdash \{p\}\ C\ \{q\} \quad \mathsf{Sta}(r, R') \quad I' \rhd \{R', G'\} \quad r \Rightarrow I'}{R * R';\ G * G';\ I * I' \vdash \{p * r\}\ C\ \{q * r\}}\ (\text{FR-SHARE})$$

Traditional frame rule for local state:

$$\frac{R;\ G;\ I \vdash \{p\}\ C\ \{q\}}{R;\ G;\ I \vdash \{p * r\}\ C\ \{q * r\}}\ (\text{FR-PRIVATE})$$

General rule that covers both cases:

$$\frac{R;\ G;\ I \vdash \{p\}\ C\ \{q\} \quad \mathsf{Sta}(r, R' * \mathsf{Id}) \quad I' \rhd \{R', G'\} \quad r \Rightarrow I' * \mathsf{true}}{R * R';\ G * G';\ I * I' \vdash \{p * r\}\ C\ \{q * r\}}\ (\text{FRAME})$$

Hide rule

The hiding rule allows to convert a resource from private to shared

$$\frac{R * R';\ G * G';\ I * I' \vdash \{p\}\ C\ \{q\} \qquad I \rhd \{R, G\}}{R;\ G;\ I \vdash \{p\}\ C\ \{q\}}\ (\textsc{Hide})$$

→ The resources from p that are described in I' are shared in the premise, but are private in the conclusion, because R and G do not mention them. (This is because R and G can only mention resources that are covered by the invariant I, and I' is disjoint from I.)

→ Really, this rule should be called differently (e.g., the "share" rule)

Rule for local computations

Rules for reasoning on code that does not involve any shared state:

$$\frac{\{p\}\ C\ \{q\}}{\mathsf{Emp};\ \mathsf{Emp};\ \mathsf{emp} \vdash \{p\}\ C\ \{q\}}\ (\text{ENV})$$

Version combined with the frame rule for shared resources:

$$\frac{\{p\}\ C\ \{q\} \quad \mathsf{Sta}(r, R * \mathsf{Id}) \quad I \triangleright \{R, G\} \quad r \Rightarrow I * \mathsf{true}}{R;\ G;\ I \vdash \{p * r\}\ C\ \{q * r\}}\ (\text{ENV-SHARE})$$

Rule for atomic

Simple version for non-guarded atomic blocs:

$$\frac{\{p\}\ C\ \{q\} \quad (p \sim> q) \Rightarrow G*True \quad (p \vee q) \Rightarrow I*true \quad p,q \text{ stable under } R*Id}{R;\ G;\ I\ \vdash \{p\}\ atomic(C)\ \{q\}}$$

→ In the atomic, all the resources can be considered to be private

→ the premise **p~>q ⇒ G∗True** ensures that the transition on the shared state executed by the command **C** satisfies one of the guarantees, that is, we can decompose **p = ps∗pl** and **q = qs∗ql** such that **(ps~>qs)⇒G** (the latter corresponds to **(p~>q)⊆G** in RGSep)

→ the premise **p ⇒ I*true** ensures that the shared state in the pre-condition **p** satisfies the invariant **I**. The **true** part is used to cover private resources. A similar premise is used for the post-condition **q**.

→ the stability side conditions are used to check that the shared part of **p** and **q** are stable under **R**. The **Id** action is used to cover private resources.

Rule for parallel

Rule for parallel:

$$\frac{R \vee G_2;\, G_1;\, I \vdash \{p_1 * r\}\, C_1\, \{q_1 * r_1\} \quad R \vee G_1;\, G_2;\, I \vdash \{p_2 * r\}\, C_2\, \{q_2 * r_2\} \quad r \vee r_1 \vee r_2 \Rightarrow I \quad I \rhd R}{R;\, G_1 \vee G_2;\, I \vdash \{p_1 * p_2 * r\}\, C_1 \parallel C_2\, \{q_1 * q_2 * (r_1 \wedge r_2)\}} \; (\textsc{Par})$$

→ equivalent to the standard rule, with extra well-formedness premises

Version combined with hiding:

$$\frac{\begin{array}{c} (R \vee G_2) * G_2';\, G_1 * G_1';\, I * I' \vdash \{p_1 * m * r\}\, C_1\, \{q_1 * m_1' * r_1'\} \\ (R \vee G_1) * G_1';\, G_2 * G_2';\, I * I' \vdash \{p_2 * m * r\}\, C_2\, \{q_2 * m_2' * r_2'\} \\ I \rhd \{R, G_1, G_2\} \quad I' \rhd \{G_1', G_2'\} \quad r \vee r_1' \vee r_2' \Rightarrow I \quad m \vee m_1' \vee m_2' \Rightarrow I' \end{array}}{R;\, G_1 \vee G_2;\, I \vdash \{p_1 * p_2 * m * r\}\, C_1 \parallel C_2\, \{q_1 * q_2 * (m_1' \wedge m_2') * (r_1' \wedge r_2')\}} \; (\textsc{Par-Hide})$$

→ for the parent, $\mathbf{p_1}$ and $\mathbf{p_2}$ and **m** are private, and **r** is shared

→ for the left branch, $\mathbf{p_1}$ is private and **m** and **r** are shared

→ for the right branch, $\mathbf{p_2}$ is private and **m** and **r** are shared

→ **I** is the fence for **r** and **I'** is the fence for **m**

Rule of consequence

Rule of consequence:

$$\frac{p' \Rightarrow p \quad R' \Rightarrow R \quad G \Rightarrow G' \quad q \Rightarrow q' \quad R; G; I \vdash \{p\}\, C\, \{q\} \qquad p' \vee q' \Rightarrow I' * \mathsf{true} \quad I' \rhd \{R', G'\}}{R'; G'; I' \vdash \{p'\}\, C\, \{q'\}} \ (\text{CSQ})$$

→ as usual, on a given judgment, we may strengthen **p** or weaken **q**

→ similarly, on a given judgment, we may reduce the set **R** (we have completed a proof assuming the context could be very vicious, but now we only need to assume the context to be a little vicious) or enlarge **G** (we have completed a proof showing that the program only makes a small number of possible transitions, but now we are happy to assume that the program is making a larger number of possible transitions).

→ well-formedness side conditions are also included

Soundness of LRG

Established using a somewhat similar technique as in RGSep.

Note that the conjunction rule is sound in the system.

Open questions

Completeness of LRG for compositional verification: is there a formal definition of "truely compositional concurrent program logic"?

Interest of the rule of conjunction: I have never felt the need for additive conjunction/disjunction while verifying sequential programs. Can we always do without it for verifying concurrent programs?

Relation to CSL: the related work section of LRG conjectures that CSL can be viewed as a special version of LRG. Does this mean that we'll never hear about CSL again?

Relation to RGSep: by avoiding a two-layer logic, LRG seems to improve over RGSep. Yet, LRG imposes a precision requirement on the fences. When does this lead to the need for additional auxiliary variables? Would this really a problem in practice for mechanized proofs?

Bibliography

– A Marriage of Rely/Guarantee and Separation Logic.
  Viktor Vafeiadis, Matthew J. Parkinson. CONCUR 2007.

– Modular fine-grained concurrency verification.
  Viktor Vafeiadis. PhD thesis. Chapter 3.

– Local rely-guarantee reasoning.
  Xinyu Feng. POPL 2009.