Concurrency, Rely/Guarantee and Separation Logic

Cliff Jones

Newcastle University

Overture Workshop
2014-06-21

Expressive power must be a “good thing”

I beg to differ!

- (decidable) type systems
- $\{pre\}\ S\ \{post\}$
- data abstraction (which is a sort of *leitmotiv*)
    - benefit of making clear what can *not* be discussed!
- “power” can beget intractability

One message: start with concepts

cf. "to a man with a hammer, every problem is a nail"

- e.g. concept input/output relation of a program
- Hoare logic based on specifications
  - $\{p\}\ S\ \{q\}$
  - pre/post say less than implementation
  - but "extend the vocabulary" (using $\wedge / \neg$)
  - easier to show "satisfaction of specification"
  - … than equivalence of two programs
- post conditions are relations!

An important concept: separation

- separation = zero interference/visibility(!)
  - question: control reads (as well as writes)?
- in the case of normal (stack) variables …
  - just separating alphabets
  - cf. VDM rd/wr frames
  - new R/G presentation allows $x: c$

Separation Logic

- basic idea is simple
  - to prove things about $S_1 \parallel S_2$
  - would like to conjoin their pre/post conditions
- history
  - [Hoa75] tackles parallelism with “stack” variables
  - [Rey02] covers“Separation Logic” for “heap” variables
  - Concurrent Separation Logic — Peter O’Hearn [O’H07]
- “heap” variables harder than normal (stack) variables
  - SL designed for this case
  - could “bend” R/G with $s \lhd heap$ etc.
  - . . . see below on using abstraction
- SL origin = bottom-up code analysis
  - heap variables
  - probably avoid SL for stack variables!

A key SL proof rule

"Separating conjunction" – $P * Q$ (only if P and Q are separate)

$$\boxed{SL}\ \frac{\{P_1\}\ s_1\ \{Q_1\} \quad \{P_2\}\ s_2\ \{Q_2\}}{\{P_1 * P_2\}\ s_1 \parallel s_2\ \{Q_1 * Q_2\}}$$

Example

$$\begin{array}{l} \quad\{x \mapsto \_ * y \mapsto \_\} \\ [x] \leftarrow 3 \parallel [y] \leftarrow 4 \\ \quad\{x \mapsto 3 * y \mapsto 4\} \end{array}$$

SL's "frame rule"

$$\boxed{\mathit{SL\text{-}frame}}\ \frac{\{P\}\ s\ \{Q\}}{\{P * R\}\ s\ \{Q * R\}}$$

Reynold's example [Rey02] reconsidered

The following program (!) performs an in-place reversal of a list:

$j := \textbf{nil};\ \textbf{while}\ i \neq \textbf{nil do}$
$(k := [i + \mathit{1}]; [i + \mathit{1}] := j; j := i; i := k).$

(Here the notation $[e]$ denotes the contents of the storage at address e.)

The post condition itself only has to require that some variable, say s, is changed so that

$\exists \alpha, \beta \cdot \mathit{list}(\alpha, i) * \mathit{list}(\beta, j)$

Re-do Reynold's example with "Separation as an abstraction"!?

$r, s \colon [r' = rev(s)]$

s and r are *assumed* to be distinct variables
that they are separate is a (useful and) natural abstraction

It is straightforward to "posit & prove":

$$
\begin{array}{l}
r \leftarrow [\,];\\
\textbf{while } s \neq [\,] \textbf{ do}\\
\quad r, s \colon [r' = [\textbf{hd}\, s] \curvearrowright r \wedge s' = \textbf{tl}\, s]\\
\qquad \{rev(s') \curvearrowright r' = rev(s) \curvearrowright r\}\\
\textbf{od}
\end{array}
$$

Step 2: reify r, s onto John's linked list

$$Heap = \mathbb{N} \xrightarrow{m} (X \times \left[\mathbb{N}\right])$$

$$\begin{aligned} Rep \ :: \ & h : Heap \\ & i : \mathbb{N} \\ & j : \mathbb{N} \end{aligned}$$

$$coll : \left[\mathbb{N}\right] \times Heap \to X^*$$

$$retr : Rep \to (X^* \times X^*)$$

$$retr(mk\text{-}Rep(h, i, j)) \quad \triangleq \quad (coll(i, h), coll(j, h))$$

Following this line

- separation is a (useful) abstraction
  - reification obligation is to preserve the abstraction
  - the invariant on *Rep* can use * (or a simple predicate)
- this differs from standard view of SL
  - what form would/will SL take in this view?
  - trying more (complicated) concurrent examples
- we are now working on concurrent DOM trees

SL extensions

- basic idea works well for “disjoint concurrency”
  - e.g. parallel merge sort
- many extensions — see [Par10]
  - “Next 700 Separation Logics”
- magic wand (fits algebraic view)
- fractional permissions Boyland
- (most papers) limit to “partial correctness”
- (concurrent) abstract predicates

An important concept: ownership

CSL

- Interesting examples involve “ownership”
- processes/threads can “exchange” ownership

$$[10] \leftarrow x \parallel y \leftarrow [10]$$

- … given appropriate locking – reason about passing value
- could code ownership swapping in R/G!
- actually comes back to “what is ownership?”
- one attempt to demarcate scopes of SL and R/G
  a promising dichotomy – [O’H07]
  - use SL if proving (data) race freedom
  - use R/G for “racy” programs

Issue: interference

- how to express (constraints on) interference
- R/G background:
  - VDM
  - post conditions are relations (over Σ)
  - (total) correctness
  - “posit and prove” style of development
  - importance of data abstraction/reification
  - compositional development
  - didn’t handle concurrency
- Owicki/Gries

Rely/Guarantee "thinking"

- basic idea is simple
  - acknowledge "interference"
- rely conditions
  - record assumptions the designer can make
  - cf. pre conditions
- guarantee conditions
  - requirements on running code
  - cf. post conditions
- (see below: interplay with data abstraction)

$$\underbrace{\overbrace{\sigma_0}^{pre} \quad \cdots \quad \overbrace{\sigma_i \;\; \sigma_{i+1}}^{rely} \quad \cdots \quad \underbrace{\sigma_j \;\; \sigma_{j+1}}_{guar} \quad \cdots \quad \sigma_f}_{post}$$

NB: $rely$, $guar$ (and $post$) conditions are relations

One form of R/G rule

$$\boxed{\parallel\text{-}I}\ \frac{\begin{array}{l}\{P, R \vee G_2\}\ s_1\ \{G_1, Q_1\}\\ \{P, R \vee G_1\}\ s_2\ \{G_2, Q_2\}\end{array}}{\{P, R\}\ s_1 \parallel s_2\ \{G_1 \vee G_2, Q_1 \wedge Q_2 \wedge (R \vee G_1 \vee G_2)^*\}}$$

A more algebraic presentation of R/G

"pulling R/G apart"

- abandon 5-tuple: $\{p, r\}\ S\ \{g, q\}$
- present in "refinement calculus" style
  - specifications: $[p, q]$ (special case: $[q]$)
  - **rely** $r \cdot c$
  - **guar** $r \cdot c$
  - x: c rather than VDM rd/wr framing

(Some) Laws

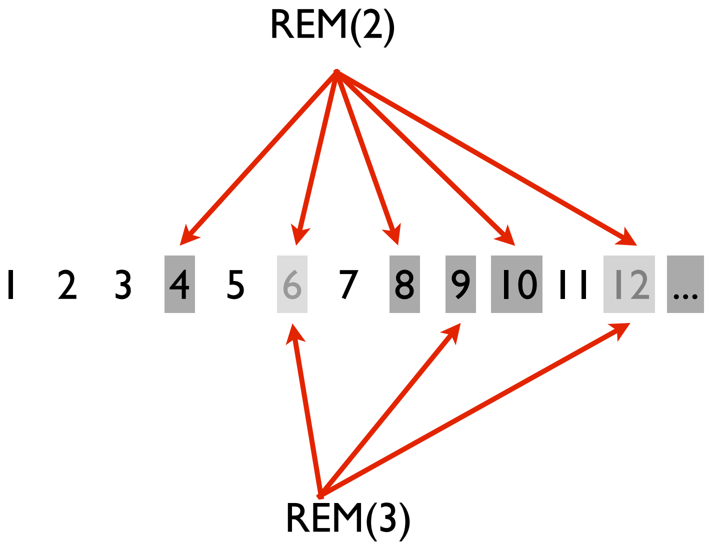
Nested-G: $(\textbf{guar}\ g_1 \cdot (\textbf{guar}\ g_2 \cdot c)) = (\textbf{guar}\ g_1 \wedge g_2 \cdot c)$

Intro-G: $c \sqsubseteq (\textbf{guar}\ g \cdot c)$

Trading-G-Q: $(\textbf{guar}\ g \cdot [g^* \wedge q]) = (\textbf{guar}\ g \cdot [q])$

Intro-multi-Par: $[\wedge_i q_i] \ \sqsubseteq \|_i\ (\textbf{guar}\ gr \cdot (\textbf{rely}\ gr \cdot [q_i]))$

Example: Prime sieve



Refinement calculus style development

Set s initially contains all natural numbers up to some n, C is the set of all composite numbers

$$[s' = s - C] = [s' \subseteq s \wedge s - s' \subseteq C \wedge s' \cap C = \{\,\}]$$

$\sqsubseteq$ by Intro-G

$$\textbf{guar}\ s' \subseteq s \wedge s - s' \subseteq C \cdot [s' \subseteq s \wedge s - s' \subseteq C \wedge s' \cap C = \{\,\}]$$

$=$ by Trading-G-Q ($s - s' \subseteq C$ is transitive)

$$\textbf{guar}\ s' \subseteq s \wedge s - s' \subseteq C \cdot [s' \cap C = \{\,\}]$$

$=$ by set theory

$$\textbf{guar}\ s' \subseteq s \wedge s - s' \subseteq C \cdot [\wedge_i s' \cap c_i = \{\,\}]$$

$\sqsubseteq$ by Intro-multi-Par

$$\textbf{guar}\ s' \subseteq s \wedge s - s' \subseteq C \cdot (\|_i\ \textbf{guar}\ s' \subseteq s \cdot \textbf{rely}\ s' \subseteq s \cdot [s' \cap c_i = \{\,\}])$$

$=$ Distribute-G

$$\textbf{guar}\ s' \subseteq s \wedge s - s' \subseteq C \cdot \textbf{guar}\ s' \subseteq s \cdot (\|_i\ \textbf{rely}\ s' \subseteq s \cdot [s' \cap c_i = \{\,\}])$$

$=$ Nested-G

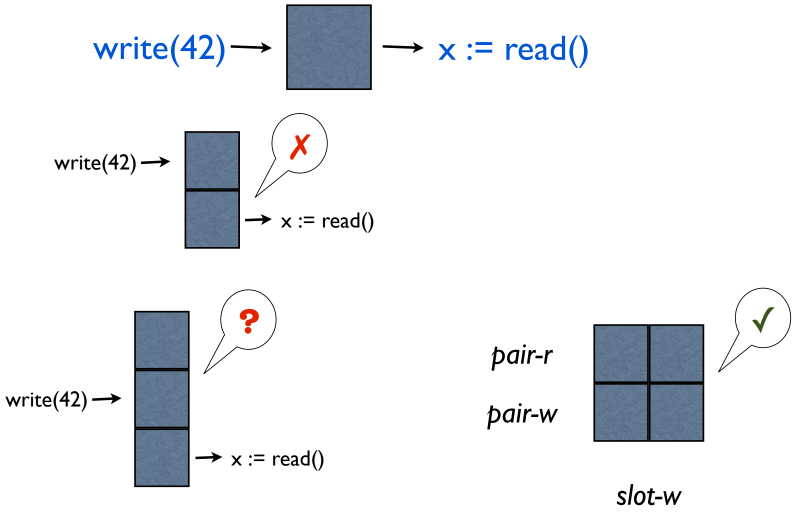
$$\textbf{guar}\ s - s' \subseteq C \wedge s' \subseteq s \cdot (\|_i\ \textbf{rely}\ s' \subseteq s \cdot [s' \cap c_i = \{\,\}])$$

Another look at Peter's "dichotomy"

Using Simpson's non-blocking "4-slot" algorithm

- Asynchronous Communication Mechanisms
  - one reader/writer
  - "lock free"
  - never read corrupt data (i.e. whist being written)
  - always read "most recently written"
- there are several algorithms, specifically ...
  - there are several proofs of Simpson's 4-slot algorithm

Hugo Simpson's 4-slot idea



Doubts about that neat dichotomy

- essence of 4-slot idea is race freedom on slots
- argue in terms of exchanging ownership (of slots)
- 4 (of many) papers on Simpson's 4-slot algorithm
  - R/G Jones & Pierce
  - SL Bornat & Amjad
- Richard Bornat [BA10]
  - uses R/G as well . . . and serialisability!
  - SL *not* used for ownership
  - Wang & Wang do — but no freshness proof
- Jones & Pierce use R/G for race *freedom*
  - . . . at an abstract level
  - introduced "possible values" concept (below)
- have a new specification (using "possible values")

Strategic messages

- start with the concepts/challenges
  - not with your pet notation
- abstraction, abstraction, abstraction
- identify issues/concepts in question
  - e.g. interference, separation
  - *then* select an apposite specific notation/approach
- reversing this order frequently ...
  - bends an approach to do things that aren't natural
  - encrypts real step
- "Ghost variables" a way to cheat on expressiveness

A (minor?) concept: possible values

- arose in Jones/Pierce work on 4-slot
- our first attempt (ABZ 2008) had an interesting flaw
  - $\mathit{hold}\text{-}r = \overleftarrow{\mathit{fresh}\text{-}w} \vee \mathit{hold}\text{-}r = \mathit{fresh}\text{-}w$
  - but Write could actually change $\mathit{fresh}\text{-}w$ many times
- actually need to say:
  - READ can set $\mathit{hold}\text{-}r$ (only) to any value set in $\mathit{fresh}\text{-}w$
  - I prefer to avoid “ghost variables” (longer story)
  - $\mathit{hold}\text{-}r \in \widehat{\mathit{fresh}\text{-}w}$
- found a variety of other uses
- + link to Hayes’ work on
  “non-deterministic expression evaluation” (TCJ paper)

New project: "Taming Concurrency"

EPSRC (UK) funded

- "pull R/G and SL apart" — two papers submitted
  - CS-TR-1394 (short)
  - CS-TR-1395 (long)
- figure out what they express well
  - try for a *semantic* combination
  - ... which might look like neither!
- (UK) project twinned with Australian (ARC) project
  - "Understanding concurrent programmes using rely-guarantee thinking"
  - led by Ian Hayes

References

Richard Bornat and Hasan Amjad.
Inter-process buffers in separation logic with rely-guarantee.
Formal Aspects of Computing, 22(6):735–772, 2010.

C.A.R. Hoare.
Parallel programming: An axiomatic approach.
Computer Languages, 1(2):151–160, June 1975.

P. W. O'Hearn.
Resources, concurrency and local reasoning.
Theoretical Computer Science, 375(1-3):271–307, May 2007.

Matthew Parkinson.
The next 700 separation logics.
volume 6217 of *LNCS*, pages 169–182. Springer, 2010.

John Reynolds.
A logic for shared mutable data structures.
In Gordon Plotkin, editor, *LICS 2002*. IEEE Computer Society Press, July 2002.