Exploring limits of Rely-Guarantee approach

Cliff Jones
(joint work with with Nisansala Yatapanage)

Newcastle University

WG 2.3 Providence, RI
2018-05-07

Example: GC

NB: it is the underlying points, not the example, that matter

- concurrent Garbage Collection: Ben Ari [BA84]
- an ‘Owicki/Gries’ proof by Jan van de Snepscheut [vdS87]
- illustrates:
  - asymmetric processes (and therefore R/G):
    $Collector \mid\mid Mutator$
  - three $Mutator$ options: $Redirect/Malloc/Zap$
- highlights a limitation of R/G
  - lets us explore options

A (second) data reification introduces marking

$$\Sigma_2 \;::\; roots\colon Addr\text{-}\mathbf{set}$$
$$hp\colon Heap$$
$$free\colon Addr\text{-}\mathbf{set}$$
$$marked\colon Addr\text{-}\mathbf{set}$$

where

$$inv\text{-}\Sigma_2(mk\text{-}\Sigma_2(roots, hp, free, marked)) \quad \triangleq \quad \cdots$$

$$Heap = Addr \xrightarrow{m} Node$$

$$Node = \left[Addr\right]^*$$

Mark trivial version (non-interfering)

$Collector \triangleq (Unmark; Mark; Sweep)$

$Mark$
wr $marked$
rd $hp, roots, free$
pre true
rely $marked' = marked \land free' = free \land hp' = hp$
guar true
post $marked' = (free \cup reach(roots, hp))$

transition from sequential to interfering shows an R/G ‘pattern’

Code for *Mark* (all three versions)

$Mark \triangleq$
repeat
$mc \leftarrow$ **card** $marked$;
$Propagate$
until card $marked = mc$

$Propagate$:
$consid \leftarrow \{\,\}$;
do while $consid \neq Addr$
let $x \in (Addr - consid)$ **in**
if $x \in marked$ **then** $Mark\text{-}kids(x)$ **else skip**;
$consid \leftarrow consid \cup \{x\}$
od

proofs of (sequential) loops are straightforward — but differ

Concurrent GC

Collector has to enlist *Mutator* to help with marking process!

now:

Unmark
wr *marked*
rd *roots*, *free*
pre true
rely $free' \subseteq free$
guar $marked' \subseteq marked$
post $\forall a \in (Addr - (roots \cup free)) \cdot \exists m \in \overleftarrow{marked} \cdot a \notin m$

simplification: upper bound of marking (lower of GC) ignored for now

Possible values

- in the presence of interference
  $\{P\}x \leftarrow y\{x' = y \vee x' = y'\}$
  is not enough — instead we now write:
  $\{P\}x \leftarrow y\{x' \in \widehat{y}\}$
- concept identified in [JP11]
  - development of Simpson's '4-slot' algorithm for ACMs
  - (asymmetric $WRITE \mid\mid READ$)
- possible values ('posvals') extends expressive power
  - avoids the 'need' for some ghost variables
  - [JH16] has a rather clear specification of ACMs
    $r \leftarrow READ()$: $r \in \widehat{buf}$
    $WRITE(x)$: $buf' \neq buf \Rightarrow buf' = x$

Mark simplified version (atomic)

Pretend $Mutator(Redirect)$ can make change/mark atomically

$$< hp(a), marked \leftarrow hp(a) \dagger \{i \mapsto b\}, marked \cup \{b\} >$$

code shown for brevity – paper has R/G specification

$Mark$
wr $marked$
rd $roots, hp, free$
pre true
rely $marked \subseteq marked' \wedge$
$\forall (a,i) \in \textbf{dom}\, hp \cdot hp'(a,i) \neq hp(a,i) \;\Rightarrow\; hp'(a,i) \in marked'$
guar $marked \subseteq marked'$
post $reach(roots, hp') \subseteq marked'$

slightly more complicated but previous proof *strategy* works

with exactly the same code — different R/G

Concurrent version (finally)

- the *Mutator* cannot redirect/mark atomically
  $hp(a) \leftarrow hp(a) \dagger \{i \mapsto b\};\ marked \leftarrow marked \cup \{b\}$
- I have worked on 'fiction of atomicity/atomicity refinement' but I don't think that's the right approach here
- the essence of the correctness is a 'three state' argument:
  Mutator can change $hp(a,i)$ to point to b
  ... then go to sleep (before marking b)
  if the *Collector* has passed a, *post-Propagate* could fail
  however there must be another pointer (say $hp(c,j)$) to b
  if b not marked by *Collector*, c is still to be handled
  BUT what if the *Mutator* destroys the $hp(c,j)$ link first?
  ... to do so, the *Mutator* must have marked b — phew!
- such a three state argument is not expressible as rely

Various ways to undermine compositionality

- is R/G expressively weak? intentionally so!
- minimal information about the (concurrent) environment
  - R/G facilitates this
  - interesting examples: asymmetric concurrent processes
- in Owicki/Gries [Owi75] one reasons about the final code!
- 'ghost' variables dent/destroy compositionality
  - in the extreme, they can dictate the entire environment code e.g. $(PC = 1 \Rightarrow x = 5) \wedge (PC = 2 \Rightarrow x = 6) \wedge \cdots$
  - in many cases, such 'auxiliary' variables can be avoided often with an apposite abstraction
  - if I have to use them, I want a test (cf. homomorphic rule vs. 'biased' specifications)

Concurrent version: alternative (i)

concede a ghost variable! (as in the SETTA paper)

tbm: $\left[Addr\right]$

$< hp(a), tbm \leftarrow hp(a) \dagger \{i \mapsto b\}, b >$;
$< marked, tbm \leftarrow marked \cup \{tbm\}, \textbf{nil} >$

but we do have a test!

$$rely\text{-}Collector_c : \Sigma_2 \times \Sigma_2 \to \mathbb{B}$$

$$\begin{array}{l} rely\text{-}Collector_c(\sigma, \sigma') \quad \triangleq \\ \quad free' \subseteq free \wedge marked \subseteq marked' \wedge \\ \quad (\forall (a, i) \in hp \cdot \\ \qquad hp'(a, i) \neq hp(a, i) \wedge hp'(a, i) \in Addr \Rightarrow \\ \qquad\qquad hp'(a, i) \in marked' \vee tbm' = hp'(a, i)) \wedge \\ \quad (tbm \neq \textbf{nil} \wedge tbm' \neq tbm \;\Rightarrow\; tbm \in marked' \wedge tbm' = \textbf{nil}) \end{array}$$

Concurrent version: more alternatives

- reject the restrictions of R/G and use Temporal Logic?
  - I worry that RGITL [STER11] is too expressive!
  - slippery slope to arguing about the combined code
- could specify that the *Mutator* preserves *can-be-completed*
- also damages compositionality! the designer of *Mutator* ...

(My) current preferred alternative

again, code as abbreviation for R/G specifications

$mr\text{-}1$: $< hp(a) \leftarrow hp(a) \dagger \{i \mapsto b\} >$
$mr\text{-}2$: $< marked \leftarrow marked \cup \{b\} >$

two roles for tbm in $rely\text{-}Collector_c$: remember value + mark point

Useful Lemma:

$$tbm \neq \textbf{nil} \;\; \Rightarrow \;\; \exists \{(a,i),(b,j)\} \subseteq \textbf{dom}\, hp \cdot (a,i) \neq (b,j) \wedge hp(a,i) = hp(b,j) = tbm$$

So, at the point between $mr\text{-}1/mr\text{-}2$: rely on second path
but this is a ‘local’ rely — cf. [JH16]

Clearly this argument also limits compositionality — less so?

Compositional development for concurrent designs

- compositionality is difficult to achieve for concurrency
  - see [dR01]
  - interference is the essence of concurrency
  - (even with process algebras!)
- recording interference (e.g. Rely/Guarantee)
  - with symmetric processes (e.g. $\mathit{SIEVE} = ||_i \mathit{REM}(i)$)
  - more interesting with asymmetric processes: different R/Gs
  - e.g. *4-slot*, *QREL* (aka 'union/find'), *GC*

Conclusions

- compositionality = good engineering
  - ‘compositionality’ wrt a notation?
- R/G can’t specify GC fully compositionally
  - but the (related) versions of R/Gs are nice!
  - ‘$Collector/Mutator$ were designed together’ Simon Doherty
- adding ‘ghost variables’
  - the 3-state argument looks like a clear test
- the jury is still out on the least damaging solution
  - actually, I think all candidate workarounds tell us something
- (for me) examples are essential!
  - N.B. two ways of using $\widehat{y}$
- comments?

Mordechai Ben-Ari.
Algorithms for on-the-fly garbage collection.
ACM Transactions on Programming Languages ans Systems, 6(3):333–344, 1984.

W.-P. de Roever.
Concurrency Verification: Introduction to Compositional and Noncompositional Methods.
Cambridge University Press, 2001.

Cliff B. Jones and Ian J. Hayes.
Possible values: Exploring a concept for concurrency.
Journal of Logical and Algebraic Methods in Programming, 2016.

Cliff B. Jones and Ken G. Pierce.
Elucidating concurrent algorithms via layers of abstraction and reification.
Formal Aspects of Computing, 23(3):289–306, 2011.

S. Owicki.
Axiomatic Proof Techniques for Parallel Programs.
PhD thesis, Department of Computer Science, Cornell University, 1975.

G. Schellhorn, B. Tofan, G. Ernst, and W. Reif.
Interleaved programs and rely-guarantee reasoning with ITL.
In *Temporal Representation and Reasoning (TIME), 2011 Eighteenth International Symposium on*, pages 99–106, 2011.

Jan LA van de Snepscheut.
"Algorithms for on-the-fly garbage collection" revisited.
Information Processing Letters, 24(4):211–216, 1987.

(Concurrent) garbage collection: abstract specification

Σ_0 :: $busy: Addr$-**set**
$free: Addr$-**set**

where

$$inv\text{-}\Sigma_0(mk\text{-}\Sigma_0(busy, free)) \quad \triangleq \quad busy \cap free = \{\,\}$$

the invariant sets an upper bound for GC

$GC: Collector^*$

Collector
wr *free*
rd *busy*
pre true
...
post $(Addr - busy) \subseteq \bigcup \overleftarrow{free}$ lower bound for GC

First data reification — gets us to:

$$\Sigma_1 \;::\; roots : Addr\text{-}\mathbf{set}$$
$$hp : Heap$$
$$free : Addr\text{-}\mathbf{set}$$

where

$$inv\text{-}\Sigma_1(mk\text{-}\Sigma_1(roots, hp, free)) \quad \triangleq$$
$$\mathbf{dom}\; hp = Addr \wedge$$
$$free \cap reach(roots, hp) = \{\,\}$$

upper bound for GC

$$Heap = Addr \xrightarrow{m} Node$$

$$Node = Addr^*$$

Simplification: here **nil** is ignored