

Rely/guarantee for race-free reasoning

Joey wanted: “possible values in R/G”

Cliff B Jones

Computing Science
Newcastle University

NCW 2009-11-23

Contents

Intro

R/G + SL

Possible values

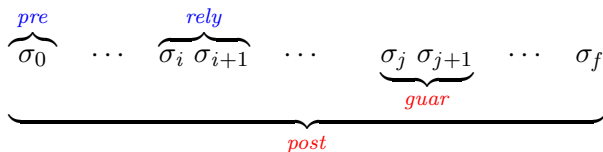
Ghost variables

Conclusions

Overview

- I'll be brief!?
- ... because:
 - it's about Simpson's 4-slot (yet) again
 - not all of you heard the London Concurrency talk in January
 - there's a (in fact two) paper(s) [Jon09, JP09]
- famous last words (in January) re [JP08]
 - "Ken has proof details (almost) done"

R/G on a slide



- there are (several sets of) proof rules
- typical R/G conditions:
 - x unchanged (but see “framing”)
 - $s \subseteq s'$
 - $flag$ implies ...
- compositional
 - can be destroyed by ghost (aka auxiliary) variables
- expressive weakness

Contents

Intro

R/G + SL

Possible values

Ghost variables

Conclusions

R/G for (abstract) race free design

- MFPS 2005
- JCR's insightful
 - “R/G for reasoning about races;
 - SL for proving no races”
- but
 - here: “race” is on the abstract variable
 - mutual *data* exclusion in concrete

Our work on Simpson's ACM implementation

- proposed $\Sigma^a / \Sigma^i / \Sigma^r$ split in [JP08]
- I confess that I tried very hard *not* to read the code!
- ... doing proofs in more detail showed a flaw (in fact two)
 - I then thought I'd have to concede a ghost variable (or two)
 - (ghost variables as a signal that abstraction found wanting?)
- I now have a development without auxiliary variables
 - submitted to a journal — see [JP09]
 - interesting (abstract) mutual exclusion argument
 - new notation: “possible values” \widehat{read}
 - significantly simpler than our previous attempt
- several others working on “clear explanations”

Plan (of *explanation*)

- Σ^a initial abstraction: interfering sub-operations + data abstraction (infinite memory)
- Σ^i data reification (“inadequate representation”) — but still “fiction of (data) atomicity”
- Σ^r code: achieve atomicity with clever representation

Specification

Ideas here:

- start with (data) abstraction of infinite buffer
- use fiction of atomicity for update of $data-w$

$$\Sigma^a ::= data-w: Value^*$$

$$fresh-w: \mathbb{N}_1$$

$$hold-r: \mathbb{N}_1$$

$$\mathbf{inv} (mk-\Sigma^a(d-w, fr-w, ho-r)) \triangleq 1 \leq ho-r \leq fr-w \leq \mathbf{len} d-w$$

Design step 1: reusing cells without clashing

Idea here:

- can retain less values in *data-w*
- non-deterministic how many
- data reification
- ... but less data prevents homomorphic retrieve function

X is an arbitrary set ($X = \mathbb{N}$ would allow no change from previous)

$\Sigma^i :: data-w: X \xrightarrow{m} Value$

fresh-w: X

hold-r: X

hold-w: X

inv ($mk\text{-}\Sigma^i(d-w, fr-w, ho-r, ho-w)$) $\triangleq \{fr-w, ho-r, ho-w\} \subseteq \mathbf{dom} d-w$

Specifications of *Write* sub-operations on Σ^i

Write(v : Value)

owns *wr* $data-w, fresh-w, hold-w$

start-Write(v : Value)

wr $data-w, hold-w$

rd $hold-r, fresh-w$

rely $hold-r \neq \overline{hold-r} \Rightarrow hold-r = \overline{fresh-w}$

guar $\overline{hold-r} \triangleleft data-w = \overline{hold-r} \triangleleft \overline{data-w}$

post $hold-w \notin \{\overline{hold-r}, \overline{fresh-w}\} \wedge data-w = \overline{data-w} \dagger \{hold-w \mapsto v\}$

commit-Write()

wr $fresh-w$

rd $hold-w$

post $fresh-w = hold-w$

Specifications of *Read* sub-operations on Σ^i

*Read()**r*: Value

owns **wr** *hold-r*

start-Read()

wr *hold-r*

rd *fresh-w*

guar $\overline{\text{hold-r} \neq \text{hold-r}} \Rightarrow \text{hold-r} = \text{fresh-w}$

post $\overline{\text{hold-r} \in \text{fresh-w}}$

*end-Read()**r*: Value

rd *data-w, hold-r*

rely $\overline{\text{data-w}(\text{hold-r}) = \text{data-w}(\text{hold-r})}$

post $r = \text{data-w}(\text{hold-r})$

NB

- mutual exclusion (at abstract level)
- R/G in spite of no races!
- code inherits the exclusion
- this use of R/G helps postpone/structure decisions
 - cf. data abstraction + non-determinacy
- still have bold atomicity assumptions
 - couldn't update *data-w* atomically on any reasonable machine
 - still work to be done
- role of data reification in achieving rely conditions
 - while assuming atomicity only at the bit level!
 - Simpson's 4-slot representation crucial

Design step 2: the four-slot representation

idea (Simpson's inspiration): explain as a choice for index set X

$$\Sigma^r :: \text{data-}w: P \times S \xrightarrow{m} [\text{Value}]$$

$$\text{pair-}w: P$$

$$\text{pair-}r: P$$

$$\text{slot-}w: P \xrightarrow{m} S$$

$$\text{wp-}w: P$$

$$\text{ws-}w: S$$

$$\text{rs-}r: S$$

$$\mathbf{inv} (mk\text{-}\Sigma^r(\text{data-}w, \text{pair-}w, \text{pair-}r, \text{slot-}w, \text{wp-}w, \text{ws-}w, \text{rs-}r)) \triangleq$$

$$\mathbf{card\ dom} \text{ data-}w = 4 \wedge$$

$$\mathbf{card\ dom} \text{ slot-}w = 2$$

Where ($\mathbf{card} P = \mathbf{card} S = 2$):

$$P, S = \mathbf{token\text{-}set}$$

$$\rho(i) \neq i$$

Contents

Intro

R/G + SL

Possible values

Ghost variables

Conclusions

For Joey: “Possible values”

- fixes problems in original proofs
- *fresh-w* avoids the “need” for a ghost variable
 - actually, that on Σ^a would be free
 - on Σ^i would have to be added
- “natural” (in the eye of (this) beholder)
- having come up with the concept, other uses emerging

Specification

$$\Sigma^a :: \text{data-}w: \text{Value}^*$$

$$\text{fresh-}w: \mathbb{N}_1$$

$$\text{hold-}r: \mathbb{N}_1$$

$$\text{inv } (mk\text{-}\Sigma^a(d\text{-}w, fr\text{-}w, ho\text{-}r)) \triangleq 1 \leq ho\text{-}r \leq fr\text{-}w \leq \text{len } d\text{-}w$$

$$\text{Read}()r: \text{Value}$$

owns wr $hold\text{-}r$

$$\text{start-Read}()$$

wr $hold\text{-}r$

rd $fresh\text{-}w$

guar $fresh\text{-}w \leq \overline{fresh\text{-}w}$

post $hold\text{-}r \in \overline{fresh\text{-}w}$

$$\text{end-Read}()r: \text{Value}$$

rd $data\text{-}w, hold\text{-}r$

rely $data\text{-}w(hold\text{-}r) = \overline{data\text{-}w(hold\text{-}r)}$

post $r = data\text{-}w(hold\text{-}r)$

Design step 1: reusing cells without clashing

$$\Sigma^i :: \text{data-}w: X \xrightarrow{m} \text{Value}$$

$$\text{fresh-}w: X$$

$$\text{hold-}r: X$$

$$\text{hold-}w: X$$

$$\mathbf{inv} (mk\text{-}\Sigma^i(d\text{-}w, fr\text{-}w, ho\text{-}r, ho\text{-}w)) \triangleq \{fr\text{-}w, ho\text{-}r, ho\text{-}w\} \subseteq \mathbf{dom} d\text{-}w$$

$$\text{Read}()r: \text{Value}$$

owns **wr** *hold-r*

start-Read()

wr *hold-r*

rd *fresh-w*

guar $\text{hold-}r \neq \overleftarrow{\text{hold-}r} \Rightarrow \text{hold-}r = \text{fresh-}w$

post $\text{hold-}r \in \overline{\text{fresh-}w}$

$$\text{end-Read}()r: \text{Value}$$

rd *data-w, hold-r*

rely $\text{data-}w(\text{hold-}r) = \overleftarrow{\text{data-}w(\text{hold-}r)}$

post $r = \text{data-}w(\text{hold-}r)$

Contents

Intro

R/G + SL

Possible values

Ghost variables

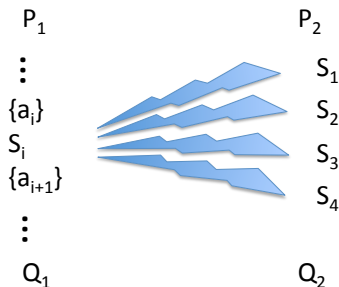
Conclusions

Auxiliary variables

- R/G conditions split development by recording interference
 - expressive “weakness” of R/G
 - ... use of a single relation limits what one can say
- Ken [Pie09] uses auxiliary variables instead of “possvals”
- ... and for explicit mutual data exclusion argument
- prompts question: are they needed?
- proving soundness of R/G rules
 - joint paper with Joey Coleman: [CJ07]
 - language with nested parallel construct + fine granularity
 - specific form of R also useful in our proof

Are R/G conditions the/a culprit (wrt ghost variables)?

cf. Owicki/Gries (or even Ashcroft and Manna)



My (current) position (on auxiliary variables)

- Auxiliary variables
 - are (sometimes) justified
 - ... to distinguish *where* in execution an assertion is true
 - i.e. permits s_l to make assertions specific to phases of s_r
 - but should be used carefully (lest compositionality compromised)
- I see above use as an extension of “phasing”
- does this lead to new proof rule(s)?
 - doubt process algebra fits
 - nor TL (as we know it)
- for now, I'll use “ghost variables”
 - ... only if I must

Contents

Intro

R/G + SL

Possible values

Ghost variables

Conclusions

Summary

- R/G for non-racey code — facilitates abstraction
- “possible values” (vs. ghost variables)
- ghost variables may be harmful
- it would be really nice to have a (“bias” like) test!
- subsidiary points
 - avoid atomicity *assumptions*
 - ... see the forthcoming paper (TR of [Jon09])

References



J. W. Coleman and C. B. Jones.

A structural proof of the soundness of rely/guarantee rules.

Journal of Logic and Computation, 17(4):807–841, 2007.



Cliff B. Jones.

The role of auxiliary variables in the formal development of concurrent programs.

In Cliff Jones, Bill Roscoe, and Ken Wood, editors, *Reflections on the work of C. A. R. Hoare*, page submitted. Springer, 2009.



Cliff B. Jones and Ken G. Pierce.

Splitting atoms with rely/guarantee conditions coupled with data reification.

In *ABZ2008*, volume LNCS 5238, pages 360–377, 2008.



Cliff B. Jones and Ken G. Pierce.

Elucidating concurrent algorithms via layers of abstraction and reification.

Technical Report CS-TR-1166, School of Computing Science, Newcastle University, 2009.



Ken Pierce.

Enhancing the Useability of Rely-Guarantee Conditions for Atomicity Refinement.

PhD thesis, University of Newcastle upon Tyne, submitted 2009.