A rational development of an ACM implementation

"4-slot" with rely/guarantee conditions, data reification, . . .

Cliff B Jones
(joint work with Ken Pierce)

Computing Science
Newcastle University

QM 2009-01-13

Contents

1. design/understanding via abstraction layers
2. ACMs
   - where to start – our specification
   - splitting atoms (gently) in abstract state
   - retaining less history
   - four-slot representation
3. summary and open problems

Key abstractions

- pre/post-conditions (as in VDM/B/...)
  - design by *sequential* “operation decomposition rules”
  - Floyd/Hoare-like rules (coping with relational post-conditions)
- Rely/Guarantee *thinking*
  - not (just) a specific set of rules
  - show importance of “frames” (cf. Separation Logic)
- abstract objects
  - choice of abstract data objects key for specifications
  - data “reification” (classic-VDM / Nipkow's rule)
  - link with R/G development
- “fiction of atomicity”
  - “splitting (software) atoms safely” [Jon07]
  - cf. database transactions [JLRW05], ...

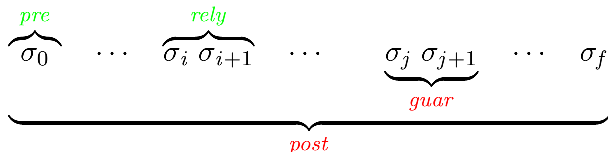
While (operation decomposition) rule

coping with relational predicates

$$\boxed{While\text{-}I}\ \frac{\{P \wedge b\}\ S\ \{P \wedge W\} \qquad P \Rightarrow \delta_l(b)}{\{P\}\ mk\text{-}While(b, S)\ \{P \wedge \neg\, b \wedge W^*\}}$$

Termination ("total correctness") comes for free with W well-founded

An R/G picture



- assumptions $pre/rely$
- commitments $guar/post$
- $rely/guar$ are both transitive and reflexive (cf. zero/multiple steps)
- other versions of R/G rules use "dynamic invariants"
- to achieve "compositionality"
  - accept $\sigma_i \rightarrow \sigma_{i+1}$ might subsequently be refined into multiple steps
  - ... might even be realised on a different representation

One possible R/G rule

remember, message is "R/G thinking"

In the spirit of:
$\{P\}\ S\ \{Q\}$
one can write:
$\{P, R\}\ S\ \{G, Q\}$

$$
\boxed{Par\text{-}I}\ \frac{\begin{array}{l}\{P, R \vee Gr\}\ sl\ \{Gl, Ql\}\\ \{P, R \vee Gl\}\ sr\ \{Gr, Qr\}\\ Gl \vee Gr\ \Rightarrow\ G\\ \overleftarrow{P} \wedge Ql \wedge Qr \wedge (R \vee Gl \vee Gr)^*\ \Rightarrow\ Q\end{array}}{\{P, R\}\ sl \mid\mid sr\ \{G, Q\}}
$$

Interesting link between R/G and data reification

cf. [Jon07]

- in $FINDP$ (find minimum array index s.t. ...)
  - $t \leftarrow min(t, local)$ in two (or n) parallel processes
  - assuming don't want to "lock" t
  - need a representation that helps us to preserve R/G conditions
  - (simple to) represent as t as $min(et, ot)$
- $SIEVE$ (of Eratosthenes — parallel)
  - each of n processes have to remove an element from a set s
  - assuming don't want to "lock" s (big!)
  - need a representation that helps preserve R/G conditions $s \subseteq \overleftarrow{s}$
  - (less obvious) represent s as a bit vector
- Simpson
  - extremely interesting!
  - our claim: this shows the essence of Simpson's contribution

Contents

1 design/understanding via abstraction layers

2 ACMs
- where to start – our specification
- splitting atoms (gently) in abstract state
- retaining less history
- four-slot representation

3 summary and open problems

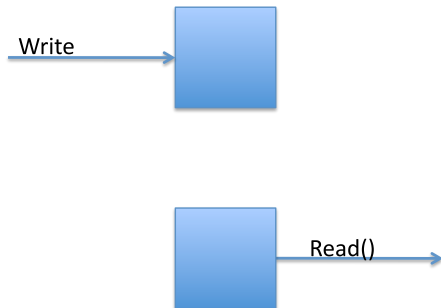
ACMs

for this group, only want to get over key steps of development
details in [JP08]; journal version "in preparation"
then list "open problems" (aka (known) unknowns)

Communication (Atomic)

ACMs

Atomic and (trying for) Asynchronous — but two slots are not enough!



Simpson's algorithm

- 4-slot algorithm
  - ingenious algorithm
  - difficult to prove correct
  - actually, *all* proofs make assumptions
  - (IMHO) different verification methods can give complementary insights
  - but few offer an *explanation*
- several other folk working on this
  - Jean-Raymond Abrial
  - Richard Bornat
  - ...
  - come back to at end
- run through our "rational reconstruction"
  - "explanation" via layers of abstraction
- essential to get the big steps right before detailed proof

Plan (of *explanation*)

- initial abstraction: atomic sub-operations + data abstraction (infinite buffer)
- step 1: interfering operations — but still "fiction of (data) atomicity"
- step 2: data reification ("inadequate representation") — but still "fiction of (data) atomicity"
- code: achieve atomicity with clever representation

Specification (i)

abstract representation holds all written values — phasing

Ideas here:

- start with (data) abstraction of infinite buffer
- use fiction of atomicity both for update of $data\text{-}w$ and to use pre/post on sub operations
- frames (long hand!) and "phasing" to simplify assertions

$$\begin{aligned}\Sigma^a \;::\;& data\text{-}w : Value^* \\ & fresh\text{-}w : \mathbb{N} \\ & hold\text{-}r : \mathbb{N}\end{aligned}$$

$$\textbf{inv}\ (mk\text{-}\Sigma^a(data\text{-}w, fresh\text{-}w, hold\text{-}r)) \triangleq$$
$$fresh\text{-}w, hold\text{-}r \in \{1..\textbf{len}\ data\text{-}w\} \wedge hold\text{-}r \leq fresh\text{-}w$$

$$\sigma_0^a = mk\text{-}\Sigma^a([\mathrm{x}], 1, 1)$$

Specification (ii)

$$\Sigma^a \;::\; data\text{-}w : Value^* \quad fresh\text{-}w : \mathbb{N} \quad hold\text{-}r : \mathbb{N}$$

$$\textbf{inv}\; (mk\text{-}\Sigma^a(data\text{-}w, fresh\text{-}w, hold\text{-}r)) \triangleq fresh\text{-}w, hold\text{-}r \in \{1..\textbf{len}\; data\text{-}w\} \wedge hold\text{-}r \leq fresh\text{-}w$$

while true do
$start\text{-}Write(v{:}\ Value){:}\ data\text{-}w \leftarrow data\text{-}w \curvearrowright [v];$
$commit\text{-}Write(){:}\ fresh\text{-}w \leftarrow \textbf{len}\; data\text{-}w$
od
while true do
$start\text{-}Read(){:}\ hold\text{-}r \leftarrow fresh\text{-}w;$
$end\text{-}Read()\,r{:}\ Value{:}\ r \leftarrow data\text{-}w(i)$ **for some** $i \in \{hold\text{-}r..fresh\text{-}w\}$
od

Examples 1, 2

$$\begin{array}{lll} \textit{start-Write}(\mathrm{y}) & .. & \textit{mk-}\Sigma^a([\mathrm{x},\mathrm{y}],1,1) \\ \textit{commit-Write}() & .. & \textit{mk-}\Sigma^a([\mathrm{x},\mathrm{y}],2,1) \\ \textit{start-Read}() & .. & \textit{mk-}\Sigma^a([\mathrm{x},\mathrm{y}],2,2) \\ \textit{end-Read}() & .. & r = \mathrm{y} \end{array}$$

$$\begin{array}{lll} \textit{start-Write}(\mathrm{y}) & .. & \textit{mk-}\Sigma^a([\mathrm{x},\mathrm{y}],1,1) \\ \textit{start-Read}() & .. & \textit{mk-}\Sigma^a([\mathrm{x},\mathrm{y}],1,1) \\ \textit{end-Read}() & .. & r = \mathrm{x} \\ \textit{commit-Write}() & .. & \textit{mk-}\Sigma^a([\mathrm{x},\mathrm{y}],2,1) \end{array}$$

Example 3

$start\text{-}Read()$	..	$mk\text{-}\Sigma^a([\mathrm{x}], 1, 1)$
$start\text{-}Write(\mathtt{y})$	..	$mk\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}], 1, 1)$
$commit\text{-}Write()$	..	$mk\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}], 2, 1)$
$start\text{-}Write(\mathtt{z})$	..	$mk\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}, \mathrm{z}], 2, 1)$
$commit\text{-}Write()$	..	$mk\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}, \mathrm{z}], 3, 1)$
$end\text{-}Read()$	..	$r \in \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$
$start\text{-}Read()$	..	$mk\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}, \mathrm{z}], 3, 3)$
$end\text{-}Read()$	..	$r = \mathtt{z}$

Specification in terms of four sub-operations ($Write$)

(at this stage) atomic operations — therefore pure pre/post specification

while true do
$start\text{-}Write(v\colon Value)\colon data\text{-}w \leftarrow data\text{-}w \curvearrowright [v];$
$commit\text{-}Write()\colon fresh\text{-}w \leftarrow$ **len** $data\text{-}w$
od
||
⋮

$Write(v\colon Value)$
$start\text{-}Write(v\colon Value)$
wr $data\text{-}w$
post $data\text{-}w = \overleftarrow{data\text{-}w} \curvearrowright [v]$
$commit\text{-}Write(v\colon Value)$
rd $data\text{-}w$
wr $fresh\text{-}w$
pre $data\text{-}w($**len** $data\text{-}w) = v$
post $fresh\text{-}w =$ **len** $data\text{-}w$

Specification in terms of four sub-operations ($Read$)

.
.
.
||
while true do
  $start\text{-}Read()$: $hold\text{-}r \leftarrow fresh\text{-}w$;
  $end\text{-}Read()r$: $Value$: $r \leftarrow data\text{-}w(i)$ **for some** $i \in \{hold\text{-}r..fresh\text{-}w\}$
od

$Read()r$: $Value$
local $hold\text{-}r$: $\mathbb{N}$
$start\text{-}Read()$
  wr $hold\text{-}r$
  rd $fresh\text{-}w$
  post $hold\text{-}r = fresh\text{-}w$
$end\text{-}Read()r$: $Value$
  rd $data\text{-}w, fresh\text{-}w$
  post $\exists i \in \{hold\text{-}r..fresh\text{-}w\} \cdot r = data\text{-}w(i)$

General messages

- note “phasing” prompts objection to “algorithmic” specification
  - “... to yell: what’s been proved?” O’Hearn 2008
- (at this stage) “fiction of atomicity”
  - but single “atomic” variable does not cover all behaviour
- “frames” (for rd/wr access)
  - plus “local”
- data abstraction
- health warning: some slips in [JP08] corrected here

Splitting atoms in Σ^a ($Write$)

idea: allow overlap — therefore rely/guarantee — but phasing still important

$Write(v\colon Value)$

$start\text{-}Write(v\colon Value)$

rd $fresh\text{-}w$

wr $data\text{-}w$

rely $fresh\text{-}w = \overleftharpoon{fresh\text{-}w} \wedge data\text{-}w = \overleftharpoon{data\text{-}w}$

guar $\{1..fresh\text{-}w\} \lhd data\text{-}w = \{1..fresh\text{-}w\} \lhd \overleftharpoon{data\text{-}w}$

post $data\text{-}w = \overleftharpoon{data\text{-}w} \frown [v]$

$commit\text{-}Write(v\colon Value)$

rd $data\text{-}w$

wr $fresh\text{-}w$

pre $data\text{-}w(\textbf{len}\ data\text{-}w) = v$

rely $fresh\text{-}w = \overleftharpoon{fresh\text{-}w} \wedge data\text{-}w = \overleftharpoon{data\text{-}w}$

post $fresh\text{-}w = \textbf{len}\ data\text{-}w$

Splitting atoms in Σ^a $(Read)$

$Read()r\colon Value$

  $start\text{-}Read()$

    rd $fresh\text{-}w$

    wr $hold\text{-}r$

    rely $hold\text{-}r = \overleftarrow{hold\text{-}r}$

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

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

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

    rely $hold\text{-}r = \overleftarrow{hold\text{-}r} \wedge \forall i \in \{hold\text{-}r..\overleftarrow{fresh\text{-}w}\} \cdot data\text{-}w(i) = \overleftarrow{data\text{-}w}(i)$

    post $\exists i \in \{hold\text{-}r..\overleftarrow{fresh\text{-}w}\} \cdot r = \overleftarrow{data\text{-}w}(i)$

General messages

- phasing
  - makes clear $start\text{-}Write$ cannot interfere with $commit\text{-}Write$
  - avoids implications (about phases) in rely conditions
- frames plus phasing significantly simplify R/G assertions
  - cf. $rely\text{-}start\text{-}Write$ on Σ^a above
- rely/guarantee have specified concurrent processes
- ... but still have unachievable atomicity on changes to $data\text{-}w$

design step: Retaining less history

Idea here:

- can retain less values in $data\text{-}w$
- non-deterministic how many
- data reification
- ... but loss of data prevents use of homomorphic retrieve function

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

$$\begin{aligned}\Sigma^i \;::\; & data\text{-}w \colon X \xrightarrow{m} Value \\ & fresh\text{-}w \colon X \\ & hold\text{-}r \colon X \\ & hold\text{-}w \colon X\end{aligned}$$

$$\textbf{inv}\ (mk\text{-}\Sigma^i(data, fresh, hold\text{-}r, hold\text{-}w)) \triangleq \{fresh, hold\text{-}r, hold\text{-}w\} \subseteq \textbf{dom}\ data$$

Relating Σ^i to Σ^a

use Nipkow/Morgan relational rule

$$r(\sigma_1^a, \sigma_1^i) \land post^i(\sigma_1^i, \sigma_2^i) \;\Rightarrow\; \exists \sigma_2^a \in \Sigma^a \cdot post^a(\sigma_1^a, \sigma_2^a) \land r(\sigma_2^a, \sigma_2^i)$$

$$r : \Sigma^a \times \Sigma^i \to \mathbb{B}$$

$$\begin{aligned}
& r(mk\text{-}\Sigma^a(data\text{-}w^a, fresh\text{-}w^a, hold\text{-}r^a), \\
& \qquad mk\text{-}\Sigma^i(data\text{-}w^i, fresh\text{-}w^i, hold\text{-}r^i, hold\text{-}w^i)) \quad \triangle \\
& \quad \textbf{rng}\ data\text{-}w^i \subseteq \textbf{elems}\ data\text{-}w^a \land \\
& \quad data\text{-}w^a(fresh\text{-}w^a) = data\text{-}w^i(fresh\text{-}w^i) \land \\
& \quad data\text{-}w^a(hold\text{-}r^a) = data\text{-}w^i(hold\text{-}r^i)
\end{aligned}$$

Specifications of the sub-operations on Σ^i

still concurrent so use rely/guarantee again

$Write(v: Value)$
  local $hold\text{-}w: X$
  $start\text{-}Write(v: Value)$
    rd $hold\text{-}r, fresh\text{-}w$
    wr $data\text{-}w, hold\text{-}w$
    rely $fresh\text{-}w = \overleftarrow{fresh\text{-}w} \land data\text{-}w = \overleftarrow{data\text{-}w}$
    guar $\{\overleftarrow{hold\text{-}r}, hold\text{-}r\} \lhd data\text{-}w = \{\overleftarrow{hold\text{-}r}, hold\text{-}r\} \lhd \overleftarrow{data\text{-}w}$
    post $hold\text{-}w \in (X - \{fresh\text{-}w, \overleftarrow{hold\text{-}r}, hold\text{-}r\}) \land data\text{-}w = \overleftarrow{data\text{-}w} \dagger \{hold\text{-}w \mapsto v\}$
  $end\text{-}Write(v: Value)$
    rd $data\text{-}w, hold\text{-}w$
    wr $fresh\text{-}w$
    pre $data\text{-}w(hold\text{-}w) = v$
    rely $fresh\text{-}w = \overleftarrow{fresh\text{-}w} \land data\text{-}w = \overleftarrow{data\text{-}w}$
    post $fresh\text{-}w = hold\text{-}w$

Specifications of the sub-operations on Σ^i

$Read()r\colon Value$
  $start\text{-}Read()$
    rd $fresh\text{-}w$
    wr $hold\text{-}r$
    rely $hold\text{-}r = \overleftarrow{hold\text{-}r}$
    post $hold\text{-}r \in \{\overleftarrow{fresh\text{-}w}, fresh\text{-}w\}$
  $end\text{-}Read()r\colon Value$
    rd $hold\text{-}r, data\text{-}w$
    rely $hold\text{-}r = \overleftarrow{hold\text{-}r} \wedge data\text{-}w(hold\text{-}r) = \overleftarrow{data\text{-}w}(hold\text{-}r)$
    post $r = data\text{-}w(hold\text{-}r)$

General messages

- simpler R/G because of read/write frames
- data reification
  - (potentially) reducing data held
  - use of (VDM's) other reification rule
- still have "bold" atomicity assumptions
  - couldn't update $data\text{-}w$ atomically on any reasonable machine
- still work to be done
- role of data reification in achieving rely conditions
  - Simpson's 4-slot representation crucial

The four-slot representation

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

$$\Sigma^r \;::\; data\text{-}w : P \times S \xrightarrow{m} Value$$
$$pair\text{-}w : P$$
$$pair\text{-}r : P$$
$$slot\text{-}w : P \xrightarrow{m} S$$
$$wp\text{-}w : P$$
$$ws\text{-}w : S$$
$$rs\text{-}r : S$$

where (key assumptions about granularity (ρ)):

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

$P = S$
$\mathbf{card}\ P = 2$
$\rho(i) \neq i$

Connection Σ^r with Σ^i

Σ^i	represented in Σ^r by
$data\text{-}w^i$	$data\text{-}w^r$
$fresh\text{-}w^i$	$(pair\text{-}w^r, slot\text{-}w^r(pair\text{-}w^r))$
$hold\text{-}r^i$	$(pair\text{-}r^r, slot\text{-}w^r(pair\text{-}r^r))$
$hold\text{-}w^i$	$(wp\text{-}w^r, wp\text{-}s^r)$

Specifications of the sub-operations on Σ^r

(of course) with rely/guarantee

$Write(v: Value)$
  local $wp\text{-}w: P$
  local $ws\text{-}w: S$
  $start\text{-}Write(v: Value)$
    rd $pair\text{-}r, slot\text{-}w$
    wr $data\text{-}w$
    rely $slot\text{-}w = \overleftarrow{slot\text{-}w} \land data\text{-}w = \overleftarrow{data\text{-}w}$
    guar $\{(\overleftarrow{pair\text{-}r}, slot\text{-}w(\overleftarrow{pair\text{-}r})), (pair\text{-}r, slot\text{-}w(pair\text{-}r))\} \lhd data\text{-}w =$
      $\{(\overleftarrow{pair\text{-}r}, slot\text{-}w(\overleftarrow{pair\text{-}r})), (pair\text{-}r, slot\text{-}w(pair\text{-}r))\} \lhd \overleftarrow{data\text{-}w}$
    post $wp\text{-}w = \rho(\overleftarrow{pair\text{-}r}) \land ws\text{-}w = \rho(slot\text{-}w(wp\text{-}w)) \land data\text{-}w(wp\text{-}w, ws\text{-}w) = v$
  $end\text{-}Write()$
    wr $pair\text{-}w, slot\text{-}w$
    rely $pair\text{-}w = \overleftarrow{pair\text{-}w} \land slot\text{-}w = \overleftarrow{slot\text{-}w}$
    guar $slot\text{-}w(pair\text{-}r) = \overleftarrow{slot\text{-}w}(pair\text{-}r)$
    post $slot\text{-}w(wp\text{-}w) = ws\text{-}w \land pair\text{-}w = wp\text{-}w$

Specifications of the sub-operations on Σ^r

$Read()r\colon Value$
local $rs\text{-}r\colon S$
$start\text{-}Read()$
rd $pair\text{-}w, slot\text{-}w$
wr $pair\text{-}r$
rely $slot\text{-}w(pair\text{-}r) = \overleftarrow{slot\text{-}w}(pair\text{-}r) \wedge pair\text{-}r = \overleftarrow{pair\text{-}r}$
post $pair\text{-}r = \overleftarrow{pair\text{-}w} \wedge rs\text{-}r = \overleftarrow{slot\text{-}w}(pair\text{-}r)$
$end\text{-}Read()r\colon Value$
rd $pair\text{-}r, data\text{-}w$
rely $pair\text{-}r = \overleftarrow{pair\text{-}r} \wedge data\text{-}w(pair\text{-}r, rs\text{-}r) = \overleftarrow{data\text{-}w}(pair\text{-}r, rs\text{-}r)$
post $r = data\text{-}w(pair\text{-}r, rs\text{-}r)$

Code (finally!)

which satisfies guarantee conditions (as well as post)

$Write(v: Value)$
local $wp\text{-}w: P$
local $ws\text{-}w: S$
$wp\text{-}w \leftarrow \rho(pair\text{-}r);$
$ws\text{-}w \leftarrow \rho(slot\text{-}w(wp\text{-}w));$
$data\text{-}w(wp\text{-}w, ws\text{-}w) \leftarrow v;$
$slot\text{-}w(wp\text{-}w) \leftarrow ws\text{-}w;$
$pair\text{-}w \leftarrow wp\text{-}w$

$Read()r: Value$
local $rs\text{-}r: S$
$pair\text{-}r \leftarrow pair\text{-}w;$
$rs\text{-}r \leftarrow slot\text{-}w(pair\text{-}r);$
$r \leftarrow data\text{-}w(pair\text{-}r, rs\text{-}r)$

Contents

1 design/understanding via abstraction layers

2 ACMs
- where to start – our specification
- splitting atoms (gently) in abstract state
- retaining less history
- four-slot representation

3 summary and open problems

Comparisons

- Henderson's thesis (JSF/CBJ supervision)
  - use "shrinking sequence" in specification
  - different approaches (including CSP/FDR) highlight facets
  - up to, including "meta-stability" of control bits
- event refinement (Abrial)
  - f/g maps to "avoid" algorithmic specification
  - non-deterministic order of events, virtual "instruction counter"
  - refine one event to many: all but one "refines skip"
- Separation Logic (Bornat, Parkinson, Vafeiadis, O'Hearn)
  - "frame" defined by alphabet of assertions
  - notation certainly more compact (than rd/wr frames here)
  - expected it to be much better on 4-slot because of "ownership"
  - in fact, (IMHO) doesn't offer intuition

Summary

- all here probably accept "refinement from abstractions"
  - hope to have shown: aids comprehension
  - have not satisfied Tony's wish for new algorithm(s)
  - "splitting atoms" – a new/old formal addition
- subsidiary points
  - rely/guarantee "thinking"
  - remember frame descriptions
  - combination with data reification
  - link with "phasing"
  - "auxiliary variables" + Nipkow's rule
  - ...
- further technical issues
  - expressiveness of R/G (thanks to Viktor Vafeiadis)
  - other technical details
  - Ken has proof details (almost) done
  - actually, reification after R/G was nowhere justified!

References

C. B. Jones, D. Lomet, A. Romanovsky, and G. Weikum.
The atomicity manifesto.
Journal of Universal Computer Science, 11(5):636–650, 2005.

C. B. Jones.
Splitting atoms safely.
Theoretical Computer Science, 357:109–119, 2007.

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.

Viktor Vafeiadis.
Modular fine-grained concurrency verification.
PhD thesis, University of Cambridge, 2007.