Refining Rely/Guarantee: a (more) algebraic presentation

Cliff Jones

Newcastle University
Joint work with Ian Hayes and Rob Colvin (Queensland)

WG 1.9
Vienna
2014-07-14
Basic Rely/Guarantee (R/G) idea (presupposed)

face interference (in specifications and design process)

- **assumptions** \(pre/rely\)
- **commitments** \(guar/post\)
- can debate specific form of R/G conditions
- many variants/applications — cf. HJJ [HJJ03, JHJ07]
Our aim is to pull apart R/G (and maybe SL) looking at the issues they cover (rather than the notation *per se*).

- instead of a fixed 5-tuple: \( \{P, R\} \rightarrow S \rightarrow \{G, Q\} \)
- separate the concepts
- one presentation in “refinement calculus” style \([P, Q]\)
- also allow “framing” as in \(s' = s - C\)
- but reservation on refinement calculus presentation below
- allow **guar** \(G \circ c\), **rely** \(R \circ c\)
- see [HJC14] (replaces [HJC13])
- preliminary work on SL [JHC14] (replaces [Jon12])
Examples

\[ \text{guar } x < x' \bullet [x' = x + 1] \sqsubseteq x := x + 1 \]

\[ \text{guar } x < x' \bullet [x' = x + 2] \sqsubseteq x := x + 1; x := x + 1 \]

\[(\text{rely } x = x' \bullet [x' = x + 1])\]

\[(\text{rely } x < x' \bullet [x + 1 \leq x'])\]

\[ [q_0 \land q_1] \sqsubseteq (\text{guar } g_0 \bullet (\text{rely } g_1 \bullet [q_0])) \parallel (\text{guar } g_1 \bullet (\text{rely } g_0 \bullet [q_1])) \]
Some intuitive Laws

\[(\text{guar true } \bullet c) = c\]

**Nested-G:**
\[(\text{guar } g_1 \bullet (\text{guar } g_2 \bullet c)) = (\text{guar } g_1 \land g_2 \bullet c)\]

**Intro-G:**
\[c \sqsubseteq (\text{guar } g \bullet c)\]

**Trading-G-Q:**
\[(\text{guar } g \bullet [g^* \land q]) = (\text{guar } g \bullet [q])\]
\[\text{guar } g \bullet (c; d) = (\text{guar } g \bullet c); (\text{guar } g \bullet d)\]
\[\text{guar } g \bullet (c || d) = (\text{guar } g \bullet c) || (\text{guar } g \bullet d)\]
The (actually “a”) key Law

Intro-multi-Par: \[ \land_i[q_i] \subseteq \|_i (\text{guar } gr \circ (\text{rely } gr \circ [q_i])) \]

This is symmetric (in \( gr \)) — many cases are not.

Other variants include rules for two operands to \( \| \)
Example: Prime sieve

REM(2)

REM(3)

Refining R/G Cliff Jones [7]
Example: Prime sieve

illustrates pattern of splitting $Q$ to (weaker) $Q$ and $R$

\[ SIEVE \]
\[ \text{wr } s: X\text{-set} \]
\[ \text{post } s' = s - C \]

\[ C = \bigcup \{ c_i | 2 \leq i \leq \lfloor \sqrt{n} \rfloor \} \]
\[ c_i = \{ i \star j | 2 \leq j \land (i \star j) \leq n \} \]

\[ SIEVE \text{ is satisfied by} \]
\[ \text{do } i = 2 \text{ to } \lfloor \sqrt{n} \rfloor \text{ REM}(i) \]

\[ REM(i) \]
\[ \text{post } s' = s - c_i \]
Example: Concurrent prime sieve

... as a conjuring trick (with rabbits)

\( REM(i) \)
\( \text{post } s' = s - c_i \)

\( SIEVE \) is satisfied by \( \bigcap_{i=2}^{\lfloor \sqrt{n} \rfloor} REM(i) \)

\( REM(i) \)
\( \text{rely } s' \subseteq s \)
\( \text{guar } s - s' \subseteq c_i \land s' \subseteq s \)
\( \text{post } s' \cap c_i = \{ \} \)
Refinement calculus style development

Set $s$ initially contains all (?) natural numbers up to some $n$

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

- by Intro-G
  $$\text{guar } s' \subseteq s \land s - s' \subseteq C \land s' \cap C = \{\} \bullet$$
  $$[s' \subseteq s \land s - s' \subseteq C \land s' \cap C = \{\}]$$

- by Trading-G-Q
  $$\text{guar } s' \subseteq s \land s - s' \subseteq C \bullet [s' \cap C = \{\}]$$

- by Intro-muti-Par
  $$\text{guar } s' \subseteq s \land s - s' \subseteq C \bullet$$
  $$\left(\|i \text{ guar } s' \subseteq s \bullet \text{ rely } s' \subseteq s \bullet [s' \cap c_i = \{\}]\right)$$

- by Nested-G
  $$\text{guar } s - s' \subseteq C \land s' \subseteq s \bullet (\|i \text{ rely } s' \subseteq s \bullet [s' \cap c_i = \{\}])$$
Reservations

RC is very pretty, but industrial specs are not one-liners
Possible values

- aversion to “history” (aka “ghost”) variables [Jon10]
- “possible values” might offer a new concept in specifications
- we needed something like post: \( x = y \lor x = y' \)
  - ... but multiple changes to \( y \) possible!
- enter \( \hat{y} \)
possible values

\{P\}x \leftarrow y\{x \in \widehat{y}\}
remember \widehat{y} is a set

The original one (in developing Simpson’s 4-slot):

\textit{post-start-Read: hold-r} \in \textit{fresh-w}
SIEVE again

- a useful check at the beginning of $REM(i)$ is whether $i \in s$
- but only of use if the “threads” are launched in sequence
- a better check might be to test $i \in s$ frequently
- but the specification here could be delicate
  
  $\text{rely-REM} \triangleq i \notin s \Rightarrow \text{multiples of } i \text{ will be deleted}$
- but with posvals:
  
  $\text{post-REM} \triangleq (\forall pos \in \hat{s} \cdot i \in pos) \Rightarrow s' \cap c_i = \{\}$
- remember $\text{guar-REM}$
possible values: good uses

\[
y \leftarrow 1; \\
(y \leftarrow 3) \parallel x \leftarrow y \parallel (y \leftarrow 4)
\]

\[x \leftarrow y \text{ could have a rely}\]

\[\text{rely: } \hat{y} \subseteq \{1, 3, 4\}\]

\[\begin{align*}
\text{pre: } & \text{is-odd}(y) \\
\text{rely: } & y \neq \bar{y} \Rightarrow \text{is-odd}(y) \\
\text{or: } & \\
\text{rely: } & \forall v \in \hat{y} \cdot \text{is-odd}(v) \\
\text{rely: } & p((\hat{y}, z))
\end{align*}\]
“Towards” reasoning about posvals

\{true\} \textbf{while} \ y \neq 0 \ \textbf{do} \ x \leftarrow x + 1 \ \textbf{od} \ \{0 \in \hat{y}\}
\{true\} \ l \leftarrow [v] \ \bar{\land} \ l \ \{\exists s \in \hat{l} \cdot \text{hd}\ s = v\}

or:
\textbf{guar} \ x \neq \check{x} \ \Rightarrow \ x = y \cdot C \ \textbf{satisfies} \ [x \in \hat{y} \lor x = \check{x}] 

with $x$ owned:
(\textbf{if} \ y = 7 \ \textbf{then} \ x \leftarrow \text{false}) \ \textbf{satisfies} \ [x = \text{true}, \ 7 \notin \hat{y} \ \Rightarrow \ x = \text{true}]
FINDP

• classic problem from Owicki’s thesis
• illustrates preservation of a property (if it holds)
• \textbf{guar-inv} \; p \cdot c \triangleq (\textbf{guar}(p \Rightarrow p') \cdot c)
• (in both the sequential and concurrent development)
• repeats experience that data abstraction/reification intimately linked to R/G
• and . . .
**FINDP:** [HJC14] goes through development of with:

\[ satp(v, t) \triangleq t \in \text{dom}(v) \land p(v(t)) \]

\[ notp(v, s, t) \triangleq (\forall i \in s \cdot i < t \Rightarrow \neg p(v(i))) \]

\[ t: \textbf{rely} \ id(\{v, t\}) \bullet [(t' = \text{len}(v) + 1 \lor satp(v, t')) \land notp(v, \text{dom}(v), t')] \]

\[ \sqsubseteq \]

\begin{verbatim}
var ot, et •
  ot := len(v) + 1 ;
  et := len(v) + 1 ;
  ( var oc •
    oc := 1 ;
    while oc < ot \land oc < et do
      if p(v(oc)) then ot := oc
      else oc := oc + 2
    t := min(ot, et)
  )

var ec •
  ec := 2 ;
  while ec < ot \land ec < et do
    if p(v(ec)) then et := ec
    else ec := ec + 2
\end{verbatim}

NB tests: use shared variables
are not assumed to be executed atomically
Other on-going work

- semantics (difficult)
- even more abstract R/G — invite Ian to describe (cf. CKAs)
- data abstraction/reification is everywhere — working on best style/fit
- review some of the older extensions to R/G
- “separation as an abstraction” — [JHC14]
Where are we heading?

- R/G has spawned a lot of ideas
- 2 new projects (EPSRC, ARC)
- aim: (“pull apart” R/G and SL) start from issues
  - separation
  - ownership
  - interference
  - progress
  - do once (cf. Linearisability (vs. splitting atoms))
- don’t take position:
  “my notation (aka “hammer”) solves every problem”
- balance expressive strength/weakness against tractability
Ian J. Hayes, Cliff B. Jones, and Robert J. Colvin.
Reasoning about concurrent programs: Refining rely-guarantee thinking.

Ian J. Hayes, Cliff B. Jones, and Robert J. Colvin.
Laws and semantics for rely-guarantee refinement.

Ian Hayes, Michael Jackson, and Cliff Jones.
Determining the specification of a control system from that of its environment.

Balancing expressiveness in formal approaches to concurrency.

Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson.
Deriving specifications for systems that are connected to the physical world.

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

Cliff B. Jones.
Abstraction as a unifying link for formal approaches to concurrency.