Separating the concerns of rely and guarantee in reasoning about concurrent programs

Ian J. Hayes
The University of Queensland
visiting Newcastle University until Christmas
work with Cliff Jones, Rob Colvin and Larissa Meinicke

BCS-FACS — 16 September 2015
My commitments

**Guarantee** to explain rely and guarantee (separately)

**Postcondition** finish on time

assuming

**Precondition** start on time

**Rely** audience asks (all the right) questions
Compositional proof/development method

Abstract specification of a component

\[
\{ x \leq y \} \ m := y \ \{ m' = \max(x, y) \} \\
\]

Refinement calculus

\[
\{ x \leq y \} ; \ [m' = \max(x, y)] \sqsubseteq m := y \\
\]

Separates specification into two commands

- a precondition assumption \( \{ x \leq y \} \) and
- a postcondition \([m' = \max(x, y)]\)
Concurrent development

- Compositional development
  - Separate development of parallel components
  - Specifying a component in isolation

- Specifying assumptions about interference
  - Cliff’s rely condition - a binary relation
  - an abstraction of the environment of the process
Rely-guarantee specification of a component

- $s$ is a set of natural numbers
- $C$ is the set of all composite (non-prime) numbers
- Jones quintuple

$$\{s = 2 \ldots N, \ s' \subseteq s\}$$

$\text{rem\_mult}(i)$

$$\{s' \subseteq s \land s - s' \subseteq C, \ s' \cap \text{mult}(i) = \emptyset\}$$

- Using four separate commands

$$\{s = 2 \ldots N\} \cap \langle s' \subseteq s \land s - s' \subseteq C\rangle^\omega \cap \langle s' \cap \text{mult}(i) = \emptyset\rangle^\omega \cap \langle s' \subseteq s\rangle^*$$

$$\sqsubseteq \text{rem\_mult}(i)$$

- What does all this mean?
- Why bother?
Guarantees

Motivation: Carroll Morgan’s invariant command \((\text{inv } i \bullet c)\)

First attempt was a command \((\text{guar } g \bullet c)\) behaves as \(c\) but

- every atomic program step must satisfy the guarantee \(g\),
- unless \(c\) aborts, at which point \((\text{guar } g \bullet c)\) aborts.

\[
\langle g \rangle^\omega \quad \pi(g) \quad \varepsilon \quad \pi(g) \quad \varepsilon \quad - \\
\cap \quad = \quad \cap \quad = \quad \cap \quad = \quad \cap \quad = \quad \cap \quad = \quad \cap
\]

\[
c \quad \pi(\sigma_0, \sigma_1) \quad \varepsilon(\sigma_1, \sigma_2) \quad \pi(\sigma_2, \sigma_3) \quad \varepsilon(\sigma_3, \sigma_4) \quad \pi(\sigma_4, \bot)
\]

Second attempt using atomic steps and weak conjunction

\[
\langle g \rangle \quad \text{performs a single atomic step satisfying } g \text{ and allows any environment steps}
\]

\[
d^\omega \quad \text{repeats command } d \text{ zero or more times, possibly infinitely many times}
\]
Weak conjunction

Weak conjunction “$c \sqcap d$” synchronises execution of $c$ and $d$

- performs a step $\pi(\sigma, \sigma')$ only if both $c$ and $d$ can
- performs a step $\epsilon(\sigma, \sigma')$ only if both $c$ and $d$ can
- terminates only if both $c$ and $d$ can terminate
- but aborts if either $c$ or $d$ can abort

Strong conjunction does not have the last property

- aborts only if both $c$ and $d$ abort

Weak conjunction can be used with any pair of processes

- not just guarantee processes $\langle g \rangle^\omega$

\(^1\)a.k.a. strict conjunction
Properties of weak conjunction

Nice algebraic properties similar to conjunction

\[
\begin{align*}
    c_0 \sqcap (c_1 \sqcap c_2) &= (c_0 \sqcap c_1) \sqcap c_2 & \text{– associative} \\
    c_0 \sqcap c_1 &= c_1 \sqcap c_0 & \text{– commutative} \\
    c \sqcap c &= c & \text{– idempotent} \\
    c \sqcap \text{chaos} &= c & \text{– identity} \\
    c \sqcap \text{abort} &= \text{abort} & \text{– abort strict (weak)} \\
    c_0 \sqsubseteq c_1 &\Rightarrow c_0 \sqcap d \sqsubseteq c_1 \sqcap d & \text{– monotonicity}
\end{align*}
\]

where

\[
\text{chaos} = \langle \text{true}\rangle^\omega
\]

i.e. any behaviour except abort
Laws for guarantees

- Strengthen guarantee
  \((g_2 \Rightarrow g_1) \Rightarrow \langle g_1 \rangle^\omega \sqsubseteq \langle g_2 \rangle^\omega\)

- Combining guarantees
  \(\langle g_1 \rangle^\omega \cap \langle g_2 \rangle^\omega = \langle g_1 \land g_2 \rangle^\omega\)

- Trading
  \([g^* \land q] \sqsubseteq \langle g \rangle^\omega \cap [q] \)

Proof sketch

\([g^* \land q] = [g^*] \cap [q] \sqsubseteq \langle g \rangle^* \cap [q] = \langle g \rangle^\omega \cap [q]\)
Prime number sieve

- $s$ is a set of natural numbers and $C$ is all composites

\[ s' = s - C \]

= by set theory

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

= the relation is reflexive and transitive

\[ (s' \subseteq s \land s - s' \subseteq C)^* \land s' \cap C = \{\} \]

\[ \sqsubseteq \text{ by Trading} \]

\[ (s' \subseteq s \land s - s' \subseteq C)^\omega \sqcap [s' \cap C = \{\}] \]

- Reflexive

\[ s' = s \Rightarrow s' \subseteq s \land s - s' \subseteq C \]

and transitive

\[ s' \subseteq s'' \land s'' \subseteq s \Rightarrow s' \subseteq s \]

\[ s - s'' \subseteq C \land s'' - s' \subseteq C \Rightarrow s - s' \subseteq C \]
In general, if \( i \sqsubseteq i \parallel i \),

\[
\begin{align*}
  i \cap (c_0 \parallel c_1) &\sqsubseteq (i \parallel i) \cap (c_0 \parallel c_1) \\
  &\sqsubseteq (i \cap c_0) \parallel (i \cap c_1)
\end{align*}
\]

For relational guarantees \( \langle g \rangle^\omega = \langle g \rangle^\omega \parallel \langle g \rangle^\omega \) and hence

\[
\langle g \rangle^\omega \cap (c_0 \parallel c_1) \sqsubseteq (\langle g \rangle^\omega \cap c_0) \parallel (\langle g \rangle^\omega \cap c_1)
\]
First attempt was command \((\text{rely } r \circ c)\)
- implements \(c\) under interference satisfying \(r\)
  \[
  \{ \text{term}(c, \text{id}) \} \cap c \sqsubseteq_{\text{id}} (\text{rely } r \circ c) \parallel \langle r \rangle^* \quad (1)
  \]
- \(\langle r \rangle^*\) represents finite interference satisfying \(r\)
- but assumes that \(c\) terminates providing its environment changes nothing (satisfies the identity relation \(\text{id}\))
- but the rely context \((\text{id} \text{ above})\) within \(c\) needs handling

Too complicated!

But (1) contains an interesting idea

Second attempt is a rely quotient operator
- \(c \sqsubseteq (c // \langle r \rangle^*) \parallel \langle r \rangle^*\) \quad (2)
- \(\langle r \rangle^*\) can be replaced by any process \(i\)
  \[
  c \sqsubseteq (c // i) \parallel i \quad (3)
  \]
Properties of rely quotient \((c // i)\)

**Motivating property**

\[
\begin{align*}
  c & \leq (c/i) \times i \quad \text{arithmetic} \\
  c & \subseteq (c // i) \parallel i
\end{align*}
\]

**Galois connection arithmetic analogy**

\[
\begin{align*}
  c/i & \leq d \iff c \leq d \times i \quad \text{arithmetic} \\
  c // i & \subseteq d \iff c \subseteq d \parallel i
\end{align*}
\]
Properties of rely quotient

- **Monotonicity**
  \[ c_1 \leq c_2 \Rightarrow (c_1 / i) \leq (c_2 / i) \quad \text{arithmetic} \]
  \[ c_1 \sqsubseteq c_2 \Rightarrow (c_1 \parallel i) \sqsubseteq (c_2 \parallel i) \]

- **Weaken rely**
  \[ i_2 \leq i_1 \Rightarrow (c / i_1) \leq (c / i_2) \quad \text{arithmetic} \]
  \[ i_2 \sqsubseteq i_1 \Rightarrow (c \parallel i) \sqsubseteq (c \parallel i_2) \]
  \[ (r_1 \Rightarrow r_2) \Rightarrow (c \parallel \langle r_1 \rangle^*) \sqsubseteq (c \parallel \langle r_2 \rangle^*) \]

- **Nested relies**
  \[ (c / i_1) / i_2 = c / (i_1 \ast i_2) \quad \text{arithmetic} \]
  \[ (c \parallel i_1) \parallel i_2 = c \parallel (i_1 \parallel i_2) \]
  \[ (c \parallel \langle r_1 \rangle^*) \parallel \langle r_2 \rangle^* = c \parallel (\langle r_1 \rangle^* \parallel \langle r_2 \rangle^*) \]
  \[ = c \parallel \langle r_1 \lor r_2 \rangle^* \]
Parallel introduction

- Weak conjunction and parallel interchange axiom
  \[(c_1 \parallel d_1) \cap (c_2 \parallel d_2) \subseteq (c_1 \cap c_2) \parallel (d_1 \cap d_2)\]

- Parallel introduction
  \[c \subseteq (c // i) \parallel i\]
  \[d \subseteq j \parallel (d // j)\]

- Hence
  \[c \cap d \subseteq ((c // i) \parallel i) \cap (j \parallel (d // j))\]

- Using the interchange axiom
  \[c \cap d \subseteq (j \cap (c // i)) \parallel (i \cap (d // j))\] (4)

- Relational rely-guarantee
  \[\left[ q_1 \land q_2 \right] = \left[ q_1 \right] \cap \left[ q_2 \right] \]
  \[\subseteq (\langle r \rangle^* \cap ([q_1] // \langle g \rangle^*)) \parallel (\langle g \rangle^* \cap ([q_2] // \langle r \rangle^*))\]
Parallel introduction for a finite set of processes

\[ [\forall k \in T \bullet q_k] \subseteq \parallel_{k \in T} \langle r \rangle^* \cap ([q_k] // \langle r \rangle^*) \]

Apply to prime number sieve
Composites within \( s \) are multiples of 2, 3, 4, 5, \( \ldots \sqrt{N} \)
Use a process for each that removes all its multiples

Processes interfere

\[ [s' \cap C = \{\}] \]
\[ = \text{as } s \subseteq 2 \ldots N, s \cap C \subseteq \bigcup_{i \in 2 \ldots \lfloor \sqrt{N} \rfloor} \text{mults}(i) \]
\[ [\forall i \in 2 \ldots \lfloor \sqrt{N} \rfloor \bullet s' \cap \text{mults}(i) = \{\}] \]
\[ \subseteq \text{by Law introduce-parallel (for } n \text{ processes)} \]
\[ \parallel_{i \in 2 \ldots \lfloor \sqrt{N} \rfloor} \langle s' \subseteq s \rangle^* \cap ([s' \cap \text{mults}(i) = \{\}] // \langle s' \subseteq s \rangle^*) \]
Conclusions

Separating concerns of rely and guarantee

- Properties of each can be considered separately
- Generalisations of both relies and guarantees
- Nice algebraic properties for reasoning
- Much simpler proofs of laws, e.g. parallel introduction
- Much easier to formalise (in Isabelle)
For the future

- Handling non-terminating processes with interference $\langle r \rangle^\omega$
- Handling progress properties other than termination
- Process-based operators applicable to other models
  - Event-based models as used in process algebras
  - Hybrid true-concurrency models
Thanks for listening
This research was supported in part by Australian Research Council Grants DP130102901.