Correct Concurrent Programs via Rely-Guarantee Refinement

I. Hayes, L. Meinicke, R. Colvin, K. Solin, K. Winter

School of ITEE, The University of Queensland, Brisbane, Australia
Oracle Labs, Brisbane, Australia

Sydney November 2015

Correctness w.r.t. a specification

Hoare Logic

$$\{p\}\ c\ \{q\}$$

Refinement Calculus

$$\llparenthesis\ q\ \rrparenthesis \sqsubseteq \ldots \sqsubseteq prog$$

Rely-Guarantee

$$\{p, r\}\ c\ \{q, g\}$$

Rely-Guarantee Refinement Calculus

Refinement into Concurrent Programs

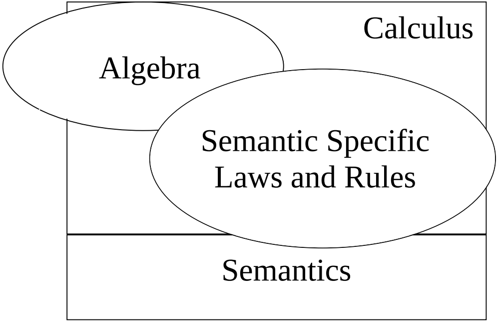
$$\llparenthesis\ q_1 \cap q_2\ \rrparenthesis$$

$$\sqsubseteq$$

$$(\textbf{rely}\ r_1) \Cap \llparenthesis\ q_1\ \rrparenthesis \Cap (\textbf{guar}\ g_1) \parallel (\textbf{rely}\ r_2) \Cap \llparenthesis\ q_2\ \rrparenthesis \Cap (\textbf{guar}\ g_2)$$

$$\text{if}\quad \begin{array}{l} g_2 \Rrightarrow r_1 \\ g_1 \Rrightarrow r_2 \end{array}$$

Rely-Guarantee Refinement Calculus – Structure



Semantic model

Aczel Traces

- states $\sigma \in \Sigma$, predicates $p \subseteq \Sigma$
- program steps $\boldsymbol{\pi}(\sigma, \sigma')$
- environment steps $\boldsymbol{\epsilon}(\sigma, \sigma')$
- trace $[\ \boldsymbol{\pi}(\sigma_0, \sigma_1),\ \boldsymbol{\epsilon}(\sigma_1, \sigma_2),\ \boldsymbol{\epsilon}(\sigma_2, \sigma_3),\ \boldsymbol{\pi}(\sigma_3, \sigma_4),\ \ldots]$
- relations $q, r, g \subseteq \Sigma \times \Sigma$

  $\boldsymbol{\pi}(g) = \{(\sigma, \sigma') \in g \cdot \boldsymbol{\pi}(\sigma, \sigma')\}$

  $\boldsymbol{\epsilon}(r) \ = \{(\sigma, \sigma') \in r \cdot \boldsymbol{\epsilon}(\sigma, \sigma')\}$

Hierarchy of Algebras

Kleene Algebra

$$(C, \; + \; , \; \cdot \; , \; * \; , \; 0, \; 1)$$

↓ *Propositional Dynamic Logic, Hoare Logic*

Kleene Algebra with Test [D. Kozen]

$$(C, \; + \; , \; \cdot \; , \; * \; , \; - \; , \; 0, \; 1)$$

↓ *partial correctness*

⇢ ~~*a 0 = 0*~~ *total correctness*

Omega Algebra [E. Cohen]

$$(C, \; + \; , \; \cdot \; , \; * \; , \; \infty \; , \; 0, \; 1)$$

Refinement Algebra [J. v Wright]

$$(C, \; + \; , \; \cdot \; , \; * \; , \; \omega \; , \; 0, \; 1)$$

Refinement Algebras

Refinement Algebra
[J. v Wright]

$$(C,\ \sqcap\ ,\ ;\ ,\ \ast\ ,\ \omega\ ,\ \top,\ skip)$$

$(C, \sqcap, \sqcup, \top, \bot)$
as complete lattice

Rely-Guarantee Refinement Algebra
[I. Hayes]

$$(C,\ \sqcap\ ,\ ;\ ,\ \parallel\ , \Cap,\ \top,\ nil,\ skip)$$

$$c^{*} = \nu x \cdot nil \sqcap c; x$$
$$c^{\omega} = \mu x \cdot nil \sqcap c; x$$

Special operators

- strict conjunction: $c \Cap d$

  $(\pi(r_1); c_1) \Cap (\pi(r_2); c_2) \;=\; \pi(r_1 \cap r_2); (c_1 \Cap c_2)$

  $(\epsilon(r_1); c_1) \Cap (\epsilon(r_2); c_2) \;=\; \epsilon(r_1 \cap r_2); (c_1 \Cap c_2)$

  $c \Cap \mathrm{abort} = \mathrm{abort}$

  e.g., $(\textbf{rely}\ r) \Cap c$ and $(\textbf{guar}\ g) \Cap c$

- parallel composition: $c \parallel d$

  $(\pi(r_1); c_1) \parallel (\epsilon(r_2); c_2) \;=\; \pi(r_1 \cap r_2); (c_1 \parallel c_2)$

  $(\epsilon(r_1); c_1) \parallel (\epsilon(r_2); c_2) \;=\; \epsilon(r_1 \cap r_2); (c_1 \parallel c_2)$

Concurrent Prime number sieve

S the set of numbers, C the set of all composite numbers.

$$\llparenthesis\ S' = S - C\ \rrparenthesis$$

$=$ by set theory

$$\llparenthesis\ S' \cap C = \emptyset\ \wedge\ S - S' \subseteq C\ \wedge\ S' \subseteq S\ \rrparenthesis$$

$\sqsubseteq$ introducing guarantee

$$(\textbf{guar}\ S - S' \subseteq C\ \wedge\ S' \subseteq S) \Cap \llparenthesis\ S' \cap C = \emptyset\ \wedge\ S - S' \subseteq C \wedge S' \subseteq S\ \rrparenthesis$$

$=$ trading guarantee with specification as relation is transitive

$$(\textbf{guar}\ S - S' \subseteq C \wedge S' \subseteq S)\ \Cap\ \llparenthesis\ S' \cap C = \emptyset\ \rrparenthesis$$

$\sqsubseteq$ assume $C = \bigcup_i c_i$

$$(\textbf{guar}\ S - S' \subseteq C \wedge S' \subseteq S)\ \Cap\ \llparenthesis\ \forall i \cdot S' \cap c_i = \emptyset\ \rrparenthesis$$

Prime number sieve (cont)

$$(\textbf{guar}\ S - S' \subseteq C \wedge S' \subseteq S) \Cap ⦇ \forall i \cdot S' \cap c_i = \emptyset ⦈$$

$\sqsubseteq$ Law *introduce-parallel (for n processes)*

$$(\textbf{guar}\ S - S' \subseteq C \wedge S' \subseteq S) \Cap$$
$$\|_i ((\textbf{guar}\ S' \subseteq S) \Cap (\textbf{rely}\ \ S' \subseteq S) \Cap ⦇ S' \cap c_i = \emptyset ⦈)$$

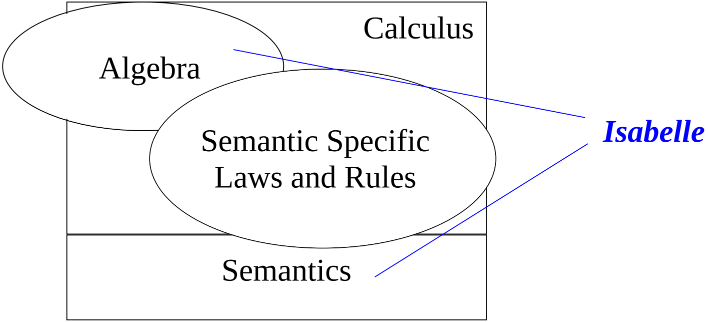
$\sqsubseteq$ Law *distribute guarantee over* $\|$

$$(\textbf{guar}\ S - S' \subseteq C \wedge S' \subseteq S) \Cap (\textbf{guar}\ S' \subseteq S) \Cap$$
$$\|_i ((\textbf{rely}\ \ S' \subseteq S) \Cap ⦇ S' \cap c_i = \emptyset ⦈)$$

$\sqsubseteq$ Law *nested-guarantee*

$$(\textbf{guar}\ S - S' \subseteq C \wedge S' \subseteq S) \Cap$$
$$\|_i ((\textbf{rely}\ \ S' \subseteq S) \Cap ⦇ S' \cap c_i = \emptyset ⦈)$$

Tool support

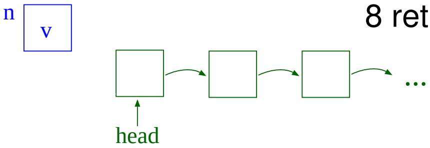


Future directions: Applications

- Verification of algorithms with fine-grained synchronisation e.g., non-blocking data structures

```
push(v : T):
n:= new(Node);
n.val := v;
repeat
    first:= head;
    n.next := first;
until CAS(head,first,n)
return
```

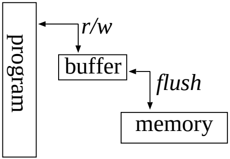
```
pop(): T:
repeat
    first:= head;
    if first = null
        return empty;
    fn := first.next;
    v := first.val
until CAS(head,first,fn);
return v
```



Future directions: Applications

- Verification of algorithms with fine-grained synchronisation e.g., non-blocking data structures
- Verification of algorithms in Weak Memory Models

p	**q**
initially: x = y = 0	
x := 1	y := 1
z := y	w := x
observable: z = w = 0	



- include buffer and flushes in target program
- allow for potential reordering of operations

Future directions: Applications

- Verification of algorithms with fine-grained synchronisation e.g., non-blocking data structures
- Verification of algorithms in Weak Memory Models
- “Rely-Guarantee Compilation” ?