Microsoft Research Asia

Verified Software Workshop and Summer School 2012

Workshop: August 23–24, 2012
Summer School: August 27–31, 2012
Venue
East China Normal University
Shanghai, China

Tony Hoare
Sir
Microsoft Research
Cambridge

Unifying Theories of Programming

with Stephan van Staden
Goals

• Unify two classical Theories of Programming
• Extend them both
  – to combine sequencing with concurrency
• Generalise them both
  – to apply to specifications, designs, contracts
  – as well as programs and assertions
1. The classical calculi

• The Hoare calculus
  – for proving correctness of sequential programs
• The Milner calculus
  – for defining correct implementation of CCS
Unexpected delight

- The definitions of the basic (triple) judgement of one calculus is the reverse of the definition for the other calculus.
- Each rule in one calculus is the dual of a corresponding rule in the other calculus
  - either by reversal
  - or by lattice duality
Hoare triple: \{p\} q \{r\}

• **defined as** \( p ; q \Rightarrow r \)
  – If you have already done \( p \), then doing \( q \)
    will achieve the overall objective \( r \)

• **example:** \{.. (x+1 \leq n)} x:= x + 1 {..( x \leq n)}

• where \( .. b \) (finally \( b \)) describes all executions that end in a state
  satisfying a single-state predicate \( b \).
Milner triple: $r - q \rightarrow p$

- defined as $q;p \rightarrow r$
  - $r$ may be executed by first executing $q$ and then $p$.
  - maybe there are other ways of executing $r$
- Tautology: $(q ; r) \rightarrow q \rightarrow r$ (as in CCS)
  - Proof: $q;r \rightarrow q;r$
- Internal step: $r \rightarrow p =_{\text{def.}} p \rightarrow r$
  - the lattice-dual of refinement
  - the step may reduce the range of subsequent choice
Technical Objections

• Originally, Hoare restricted \( q \) to be a program, and \( p, r \) to be state descriptions.
• Originally, Milner restricted \( p \) and \( r \) to be programs, and \( q \) to be an atomic action.
• Answer: The purpose of generalisation is to remove restrictions.
• The axioms are satisfied by a realistic model of the behaviour of real programs.
  – as shown in the previous lecture
Theorems

• Using the definitions of their basic judgments, we will now prove the remaining rules of both calculi from the axioms of our single algebra.
Rule of consequence

- $p \Rightarrow p' \quad \{p'\} \quad q \quad \{r'\} \quad r' \Rightarrow r$
  
  $\{p\} \quad q \quad \{r\}$

- $r \Rightarrow r' \quad r' \Rightarrow q \Rightarrow p' \quad p' \Rightarrow p$
  
  $r \Rightarrow q \Rightarrow p$

- Proof: ; is monotonic, $\Rightarrow$ is transitive.
Sequential composition

{p} q {s} \begin{array}{c}
\hline
{s} q' {r}
\end{array}
{p} q; q' {r}

\begin{array}{c}
\hline
r -q-> s \quad s -q'-> p
\end{array}
r -q; q'-> p

Proof: by associativity of ;
Sequential frame rule

\[ r \rightarrow_{q} p \] (operational def of ;)
\[ (r;f) \rightarrow_{q} (p;f) \]

- \( r;f \) has same first action as \( r \),
  and then behaves like \( p;f \)

\[ \{p\} q \{r\} \]
\[ \{f;p\} q \{f;r\} \]

Proof: mon, assoc of ;
Choice

- \{p\} q \{r\} \quad \{p\} q' \{r\}
  \{p\} (q \lor q') \{r\}
  – both choices must be correct

- r -q->p \quad r -q'-> p
  r -(q \lor q')-> p

Proof: \quad \text{distributes through } \lor
Choice and conjunction

\[ r \neg q \rightarrow p \]

\[(r \lor r') \neg q \rightarrow p\]
you need execute only one of the alternatives

\[ \{p\} q \{r\} \]

\[ \{p \land p'\} q \{r\} \]

Proof: monotonicity of ;

Note: the lattice duality
Concurrent Frame Rule

\[
\{p\} q \{r\}
\{p || f\} q \{r || f\}
\]
– adapts a rule to a wider environment \( f \)
– Proof: by frame theorem of my earlier lecture

\[
\frac{r \rightarrow p}{(r || f) \rightarrow (p || f)}
\]
– a step possible for a single thread \( r \) is still possible when \( r \) is executed concurrently with \( f \)
Modular Concurrency

• \{p\} q \{r\} \{p'\} q' \{r'\}
  \{p \parallel p'\} (q \parallel q') \{r \parallel r'\}

  – permits modular proof of concurrent programs.
  – equivalent to exchange law
Concurrent in CCS

\[ r -p-> q \quad r' -p'-> q' \]
\[ (r || r') -(p || p')-\quad (q || q') \]

– provided \( p || p' = \tau \)
– where \( \tau \) is the unobserved atomic transition, which occurs (in CCS) when \( p \) and \( p' \) are an input and an output on the same channel.
Axioms proved from calculi

from Hoare

• \( p ; (q \lor r) \Rightarrow p ; q \lor p ; r \)
• \( p ; r \lor q ; r \Rightarrow (p \lor q) ; r \)

from Milner

• \( (p \lor q) ; r \Rightarrow (p ; r) \lor (q ; r) \)
• \( p ; q \lor p ; r \Rightarrow p ; (q \lor r) \)

from both

• \( p ; (q ; r) \Rightarrow (p ; q) ; r \)
• \( (p ; q) ; r \Rightarrow p ; (q ; r) \)
• exchange law
Message

• Both the Hoare and Milner rules are derived from the same algebra of programming.

• The algebra is simpler than each of the calculi,

• and stronger than both of them combined.
2. Program Specifications

examples
refinement
unification
Specs

• Our variables (p, q, r, ...) stand for computer programs, designs, contracts, assertions, specifications,...
  – they all describe what happens inside/around a computer that executes a given program.
• The program itself is the most precise description
  – with all the excruciating detail.
• The user specification is the most abstract
  – describing only interactions with environment.
• Designs come in between.
Example specs

• Postcondition:
  – execution ends with array \( A \) sorted

• Conditional correctness:
  – if execution ends, it ends with \( A \) sorted

• Precondition:
  – execution starts with \( x \) even

• Program: \( x := x+1 \)
  – the final value of \( x \) is one greater than the initial
More examples of specs

• Safety:
  – There are no buffer overflows

• Termination:
  – execution is finite (ie., always ends)

• Liveness:
  – no infinite internal activity (livelock)

• Fairness:
  – all waiting is bounded

• Probability:
  – the ratio of heads to tails tends to 1 with time
Also

• Security
  – low programs do not access high variables
• Separation
  – threads do not assign to shared variables
• Communication
  – outputs on channel \texttt{c} are in alphabetical order
• Resource control
  – there are no space leaks
• Responsiveness
  – interval between request and response is $<1\text{sec}$
If $P$ and $Q$ are Specs

then so are

1. $P \lor Q$
2. $P \land Q$
3. $P \parallel Q$
   - describes concurrent execution of $P$ and $Q$
4. $P;Q$
   - describes sequential execution of $P$ and $Q$
Refinement: $P \implies Q$

- means that $P$ logically implies $Q$.
- If $P$ and $Q$ are specs,
  
  $P$ describes more design decisions than $Q$.
- If $P$ is a program and $Q$ is its spec,
  
  $P$ is correct, because it satisfies $Q$.
- If $P$ and $Q$ are both programs,
  
  $P$ is more predictable and controllable than $Q$ (but not necessarily faster)
Ideal of verified development

1. Formalise the user specification $S$
2. construct a design $D$ so that $D \Rightarrow S$
3. replace $S$ by $D$
4. repeat from 2...
5. until $S$ is a program
6. deliver the program

transitivity of $\Rightarrow$ ensures correctness
Verified decomposition step

1. start with a design \( D \)
2. decide to implement it by \( D1 \bullet D2 \)
   – where \( \bullet \) is a programming operator
3. prove that \( D1 \bullet D2 \Rightarrow D \)
4. implement \( D1 \), delivering program \( P1 \)
   \( \| \) implement \( D2 \), delivering program \( P2 \)
5. now deliver \( P1 \bullet P2 \)
monotonicity of \( \bullet \) ensures validity
Advantages of unification

• Same laws apply
  – for programs, designs, requirements
  – for many forms of correctness

• Tools based on the laws serve many purposes
  – and communicate by sound interfaces

• Scientific controversy is resolved
  – and engineers confidently apply the science
Unification

is the goal of every branch of pure science, because it increases credibility of theory

Diversification

is needed for each application, e.g.,

– Hoare logic: for proofs of correctness,
– Milner logic: for implementation and testing,

and to exploit faster algorithms
In praise of algebra

• Powerful
  – as we have just seen

• Familiar
  – properties are the same as high-school algebra
  – they are reused many times

• Simple
  – pairs vs. triples
  – equations vs. inductive rules
In praise of Algebra

• Flexible
  – extensible
  – modular
  – reusable
• Simplifies proofs
  – for humans,
  – and for computers
• Elegant
Isaac Newton

Communication with Richard Gregory (1694)

“Our specious algebra [the infinitesimal calculus] is fit enough to find out, but entirely unfit to consign to writing and commit to posterity.”
Bertrand Russell

• The method of postulation has many advantages. They are the same as the advantages of theft over honest toil.

Introduction to Mathematical Philosophy.
Gottfried Leibnitz

• Calculemus.