Algebra for Program Transformation

Tony Hoare
13 Feb 2017
Summary of lecture series

• Lecture 1: Geometry for Program Testing
• Lecture 2: Algebra for Program Transformation (this lecture)
• Lecture 3: Logic for Program Description.

• The Geometry is a model of the Algebra
• The Rules of Logic are derived from the Algebra
• The Axioms of the Logic are derived from the Geometry.
Summary of Lecture

• Introduction: background and motivation

• The Algebra of Program Transformation Rules
  • The operators: sequential (;) and concurrent (|) composition
  • The basic judgement is a comparison \( \leq \) between terms.

• The Laws of Programming
  • The axioms are associativity of operators, which share a unit
  • and their mutual distributivity by an “interchange” law

• A causal model
  • with actions and causal links between them
1. Introduction

Applications, principles, pioneers and precursors
Applications (1)

• Compilation of programs
  • from high-level language programs to machine code
  • or from domain-specific to general purpose languages

• Optimisation
  • of object code or source code
    of general purpose or application specific languages
Applications (2)

• Refactoring
  • to improve program structure for further evolution while preserving its current functionality

• Top-down development
  • of designs from specifications by stepwise decomposition
  • and by generation of delivered code
Goal

• An algebraic proof of a new transformation rule provides all the steps necessary to carry out the transformation, which is thereby correct by construction.

• Like a construction in a Euclidian proof:
  e.g., Proposition 1. To construct a equilateral triangle on a given base line
Pioneers

• Al Khwarizmi solved quadratic equations by algebraic reasoning

• Leibniz held that every mathematical proof can be expressed as an easily checkable symbolic calculation

• He formulated the rule of substitution of identity:
  
  If two terms have been proved equal, the truth of any statement containing one of the terms is preserved after its substitution by the other term
Precursors

• Church’s rule of beta-reduction (substituting parameters by arguments) is the basis of functional programming.

• Kleene’s Regular Expressions for Finite State Machines is the basis of distributed and interactive programming.

• Tarski’s Relational Calculus is the basis for sequential imperative programming.
2. Algebraic Transformation

What laws do we need?
And what do we need them for?
A transformation rule

• makes a term better for some purpose
• is based on a comparison $p \leq q$ between $p$ and $q$,
  • according to a some property (previously agreed)
  • eg. $p$ is cheaper, faster, prettier, lower, higher, ... than $q$
  • $p$ “is below” $q$ (q “is above” $p$) according to the agreed property
• $p \leq q$ justifies transformations between $p$ and $q$
  • either (leftward) if you choose to lower $q$ to $p$
  • or (rightward) if you choose to raise $p$ to $q$
Axiom: \( \leq \) is a preorder

- \( \leq \) is transitive: if \( p \leq q \) and \( q \leq r \) then \( p \leq r \)
  - justifies step-by-step transformation between \( r \) and \( p \)
  - with \( q \) as the intermediate step

- \( \leq \) is reflexive: \( p \leq p \)
  - justifies leaving \( p \) unchanged.

- \( \leq \) is **not** antisymmetric, so not usually a partial order
  - \( p \leq q \) & \( q \leq p \) does not necessarily imply \( p = q \)
Equivalence: \( p \equiv q \) (equality modulo \( \equiv \) )

• is defined as \( p \leq q \) & \( q \leq p \)
• \( p \) and \( q \) possess the relevant property equally
• \( \equiv \) is transitive, reflexive and symmetric: \( p \equiv q \) implies \( q \equiv p \)
• but they may differ in many other irrelevant respects
  • eg. current position, past history, future, ...
  • race, gender, age, ...
• \( \equiv \) is more interesting than equality
  • (I am the only person who equals me; but in the eye of the law, I am equivalent to all my fellow citizens)
Rule of Precongruence

• $p \leq q$ implies that $\ldots(p)\underline{\ldots} \leq \ldots(q)\underline{\ldots}$
  • where the two sides of the inequation are identical, except for the interchange of $(p)$ on the left with $(q)$ on the right
  • compare the Leibniz rule for equality, which has $=$ in place of $\leq$
• It allows transformation internally on separate parts of a term.
• The rule is also known as locality, compositionality, monotonicity, isotony, covariance, referential transparency, ...

because it has been reinvented many times
3. Laws of programming

association, unit, interchange
A Program

- is defined by (or as) the set of all its traces of execution,
- wherever or whenever it is executed,
- starting in any state, and with any combination of values input to it,
- and with any resolution of its internal non-determinism.
- Examples: with program-valued variables $P, Q, R, P', Q', \ldots$
  
  $P;Q, P|Q, (P;Q);R, (P;Q)|(P';Q'), (P|Q);(P'|Q')$
A Trace

- is a program whose execution always produces the same trace
- It contains no non-determinism, no conditionals, no loops, ...
- It consists of a set of basic actions which execute basic commands, grouped sequentially by (;) or concurrently by (|
- Examples: with trace-valued variables p, q, r, p’, q’, ...

  p;q, p\mid q, (p;q)r, (p;q)\mathbin{|}(p’;q’), (p\mid q);(p’\mid q’)
Informal semantics of traces

- \( p;q \)  
  - \( p \) then \( q \)  
  - sequential composition
  - \( q \) starts when \( p \) has finished

- \( p|q \)  
  - \( p \) during \( q \)  
  - concurrent composition
  - \( p \) and \( q \) start together, run together and finish together

- [  ]  
  - does nothing
Basic Axioms

• Let $o$ stand for either operator $(;)$ or $(||)$. Then the unit laws are

\[
[ ] o p \equiv p \equiv p o [ ]
\]

justifying removal or insertion anywhere in the term of

\[
[ ] o \quad \text{or of} \quad o [ ]
\]

\[
p o (q o r) \equiv (p o q) o r
\]

justifying removal or insertion of matched pairs of brackets anywhere

• Note these laws are self-dual. They are the same laws when read backward (except that the free variables have been permuted)
Axiom: \( \circ \) is monotonic

- If \( p \leq q \) then \( r \circ p \leq r \circ q \) and \( p \circ r \leq q \circ r \) (monotonicity of \( \circ \))

Theorem: any term containing only monotonic operators is monotonic in all its free variables

Proof: by transitivity and induction on the structure of ___( )___

- Corollary: \( \leq \) is a precongruence wrto operators ; and \( | \)

So \( p \leq q \) justifies the transformation (in either direction) between

.....(p)___ and .....(q)___
The Concurrency Principle

• A concurrent trace may at any time have more concurrent branches (threads) than there are concurrent processors to execute them.

• It can be executed by partial (or total) interleaving of the actions of the branches (i.e., by multiprogramming)

• The place where to interleave is often chosen at run time by a multiprogramming scheduler, using interrupts

• We need a law which justifies this, ... the Interchange Axiom.
Axiom: Interchange

• \((p|q) ; (p'|q') \leq (p;p') | (q;q')\)

The lhs is an interleaving of the rhs, in which the two semicolons on the right are synchronised at the point of the semicolon on the left.

• This axiom is unchanged in meaning when \(;\) is interchanged with \(|\) and \(\leq\) with \(\geq\). The above axiom translates to
  \[(p;q)|(p';q') \geq (p|p') ; (q|q')\]
  as in Boolean algebra \((p \text{ n} q) \text{ u} (p' \text{ n} q') \Rightarrow (p \text{ u} p') \text{ n} (q \text{ u} q')\)

• The unit and associative axioms are self-dual in the same way.
Simpler laws are proved from interchange

\((p|q);(p'|q') \leq (p;p') | (q;q')\) by the unit laws

\[
\begin{align*}
(p|q);q' &\leq p|(q;q') & p' = [] \\
p;(p'|q') &\leq (p;p')|q' & q = [] \\
q;(p'|q') &\leq p'|(q;q') & p = [] \\
(p|q);p' &\leq (p;p')|q & q' = [] \\
p;q' &\leq p|q' & p' = q = [] \\
q;p' &\leq p'|q & q' = p = []
\end{align*}
\]
Example of Interleaving

\[
abcd \mid xyzw \quad \text{rhs of interchange}
\]
\[
(a;bcd) \mid (xy;zw) \quad \text{associativity}
\]
\[
(a\mid xy) ; (bcd\mid zw) \quad \text{interchange}
\]
\[
(a\mid x;y) ; (b;cd\mid zw) \quad \text{associativity, interchange}
\]
\[
(a\mid x) ; y ; (b\mid zw) ; cd \quad \text{interchange (twice)}
\]
\[
xayzbwcd \quad \text{similarly}
\]
Completeness of Interchange

• If $x$ is any interleaving of $p$ with $q$, then $x \leq p \unicode{x2215} q$ can be proved from the laws given so far.

Proof: write a functional program, using only clauses derivable from the laws of this talk, that will test whether an arbitrary string $x$ is an interleaving of a given term or not. Check that it always terminates.
A Causal Model of the Algebra

A trace is identified by its actions and the causal relationships between them.
Causal dependency

- Let a, b be basic actions of the same trace p
- Let (a -> b) be a causal link by which b depends on a
  The dependency delays performance of b until after a has occurred.

- each link of p has an action of p at one end or the other
- internal links have an act of p at both ends
- input links have only the dependent end in p
- output links have only the independent end in p
Formal definitions

• \( \text{acts}([ ]) = \{ \} \)
• \( \text{acts}(p \circ q) = \text{acts}(p) + \text{acts}(q) \)
each action of \( p \circ q \) is performed by only one of \( p \) or \( q \)

• \( \text{links}([ ]) = \{ \} \)
• \( \text{links}(p \circ q) = \text{links}(p) \cup \text{links}(q) \)
links(p) \( \cup \) links(q) are the links that cross between \( p \) and \( q \).
Each such link is an input link of one operand, an output link of the other, and an internal link of the result.
Sequential links

• $\text{seq}(p)$ is the set of internal links that cross a (;) within $p$.
• $\text{seq}([ ]) = \{ \}$
• $\text{seq}(p;q) = \text{seq}(p) + \text{seq}(q) + \{(a \rightarrow b) \mid a \in \text{acts}(p) \& b \in \text{acts}(q)\}$

Note: no act of $p$ causally depends on an act of $q$, which would violate our requirement that $p$ ends before $q$ begins.

• $\text{seq}(p|q) = \text{seq}(p) + \text{seq}(q)$

This says that there are no causal links between traces that are executed concurrently.
\( p \leq q \) means \( p \) is more sequential than \( q \)

- This formal definition includes the decision that the only interesting properties of a trace are its \emph{acts} and its \emph{links} and its \emph{seq}.
- \( p \leq q \) means \( \text{acts}(p) = \text{acts}(q) \) \& \( \text{links}(p) = \text{links}(q) \)
  \[ \quad \text{\& seq}(q) \subseteq \text{seq}(p) \]
  Note: more sequential, because \( p \) has more sequential links than \( q \)
- Example: \( (p|q) ; (p'|q') \leq (p;p')|(q;q') \)
  Even though the lhs has less semicolons than the rhs, it has more links crossing between the sequentially composed operands, ie, the links between \( p \) and \( q' \) and between \( q \) and \( p' \)
Conclusion

• All our axioms are valid in this model
It is isomorphic to Grabowski’s pomset model

• Unfortunately the model prohibits any sharing or communication between the concurrent components of a program.

• A similar more recent model allows communications, but forbids deadlocks.
The Next Lecture

• will show that the Rules of Hoare Logic can be derived from our algebra, together with O’Hearn’s concurrency and frame laws.
• and will also derive the Transition Rules of Milner’s CCS, expressed as a Plotkin Structured Operational Semantics.
• The algebraic axioms can also be derived from each of the above, thereby unifying these rival theories by simple duality
Anybody against Algebra?
Isaac Newton (1642-1726)

Communication with Richard Gregory (1694)

“Our specious [falsely convincing] algebra [the infinitesimal calculus] is fit enough to find out [has some heuristic value], but entirely unfit to consign to writing and commit to posterity [it cannot and must not be published].”  (with translation to Modern English)
The method of postulation has many advantages. They are the same as the advantages of theft over honest toil.

Introduction to Mathematical Philosophy.
Gottfried Leibniz (1646-1716)

• calculemus