The Laws: Summary

• What are they?
• What do they mean?
• Are they useful?
• Are they true?
• Are they beautiful?
1. The Laws

are algebraic equations like

\[ 2pxq + qxq \leq (p+q) \times (p+q) \]
Variables \( p, q, r, \ldots \)

- stand for specifications/designs/programs describing all behaviours of a computer that are desired/planned/actual when the program is executed
- a single behaviour is recorded as a set of events, occurring inside or near a computer system while it is executing a program
Three operators

- then ; sequential composition
- with || concurrent composition
- skip | does nothing
Their intended meaning

- then ; sequential composition
- with || concurrent composition
- skip | does nothing
- $p; q$ describes the behaviour resulting from execution of $p$ till completion followed by execution of $q$
- $p||q$ describes their concurrent execution $p$ and $q$ start together and finish together
Five Axioms

• assoc \[ p;(q;r) = (p;q);r \] (also ||)
• comm \[ p||q = q||p \]
• unit \[ p||l = p = l||p \] (also ;)
Reversibility

- assoc \( p;(q;r) = (p;q);r \) (also \( || \))
- comm \( p||q = q||p \)
- unit \( p||I = p = I||p \) (also \( ; \))

- swapping the order of operands of \( ; \) (or of \( || \)) translates each axiom into itself.
- and each proof into a swapped proof.
Duality

• Metatheorem: (theorems for free)
  When a theorem is translated by reversing the operands of all \( \llcorner \)s the result is also a theorem.
• (same for all \( \lrcorner \)s)

• Analogy: many laws of physics remain true when the direction of time is reversed.
Refinement: $p \Rightarrow q$

- means every execution described by $p$ is also described by $q$
- in other words,
  - program $p$ is more predictable and more controllable than program $q$
  - program $p$ meets spec $q$
  - design $p$ conforms to design $q$
Axiom

• $\Rightarrow$ is a partial order
  – reflexive $p \Rightarrow p$
  – transitive if $p \Rightarrow q$ & $q \Rightarrow r$ then $p \Rightarrow r$
• swapping the operands of $\Rightarrow$
  translates each axiom into itself
• justifies duality by order reversal
  – if $p \Rightarrow q$ is a theorem proved from these axioms
    so is $q \Rightarrow p$
• (later axioms will violate this duality)
Monotonicity

• Definition: an operator \( \bullet \) is monotonic if

\[
\begin{align*}
p \Rightarrow q & \Rightarrow p \bullet r \Rightarrow q \bullet r \\
\quad \& r \bullet p & \Rightarrow r \bullet q
\end{align*}
\]

• Axioms: ; and \( \mid \mid \) are monotonic

• In a theorem, we can replace any subterm of a term on the left (right) of \( \Rightarrow \) by one that is more (less) refined
Monotonicity

• Metatheorem:
Let $p \Rightarrow q$ be a theorem
Let $F$ be a formula containing $p$.
Let $F'$ be a modification of $F$ that just replaces an occurrence of $p$ by $q$

Then $F \Rightarrow F'$ is also a theorem
Exchange Axiom

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

• LHS describes certain interleavings of RHS
  – those where the two RHS ;s are synchronised
• implemented by interleaving \(p\) with \(q\)
• followed by an interleaving of \(q\) with \(q'\)
Exchange Axiom

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

• Theorem (frame): \((p||q); q' \Rightarrow p||(q;q')\)
  
  – Proof: substitute I for \(p'\) in exchange axiom

• Theorem: \(p;q' \Rightarrow p||q'\)
  
  – Proof: substitute I for \(q\)

• This axiom is self-dual by time-reversal
  
  – but not by order-reversal
2. Applications

to Hoare logic
and to Milner transitions
The laws are useful

• for proof of correctness of programs/designs
  – by means of Hoare logic
  – (extended by concurrent separation logic)
  – to describe the structure of proofs

• for design/proof of implementations
  – using Milner transitions
  – (extended by sequential composition)
  – to describe the steps of execution.
The Hoare triple

• Definition: \( \{p\} \ q \ \{r\} = p; q \Rightarrow r \)
  – If \( p \) describes what has happened so far,
  – and then \( q \) is executed to completion,
  – the overall execution will satisfy \( r \).

• Example: \( p \) and \( r \) may be ‘assertions’,
  – describing all executions that leave the machine
    in a state satisfying a given Boolean predicate.
The rule of composition

• Definition: \( \{p\} q \{r\} = p;q \Rightarrow r \)

• Theorem:
  \[
  \{p\} q \{s\} \underbrace{\{s\} q'} \{r\} \\
  \{p\} q;q' \{r\}
  \]
Proof

• Definition: \( \{p\} q \{r\} = p; q \Rightarrow r \)

• expanding the definition:

\[
\begin{align*}
p; q & \Rightarrow s \quad \text{s;q’} \Rightarrow r \\
p; q; q’ & \Rightarrow r
\end{align*}
\]

because ; is monotonic and associative
The rule of consequence

• Theorem

\[ p' \Rightarrow p \quad \{p\} \quad q \quad \{r\} \quad r \Rightarrow r' \]
\[ \{p'\} \quad q \quad \{r'\} \]

Proof: monotonicity and transitivity
Modularity rule for $||$

• in concurrent separation logic

\[
\begin{align*}
\{p\}q\{r\} & \quad \{p'\}q'\{r'\} \\
\{p||p'\} (q||q') & \{r||r'\}
\end{align*}
\]

– permits modular proof of concurrent programs.

• it is \textit{equivalent} to the exchange law
Modularity rule implies Exchange law

- By reflexivity: $p; q \Rightarrow p; q$ and $p'; q' \Rightarrow p'; q'$
- take these as antecedents of modularity rule
  - replacing $r, r'$ by $p; q$ and $p; q'$,
- After the same substitution, the conclusion of the modularity rule gives
  
  $$(p || p') ; (q || q') \Rightarrow (p; q) \ | \ | (p'; q')$$
  - which is the Exchange law
Exchange law implies modularity

• Assume: $p;q \Rightarrow r$ and $p';q' \Rightarrow r'$
• monotonicity of $||$ gives
  $$(p;q) || (p';q') \Rightarrow r || r'$$
• the Exchange law says
  $$(p || p') ; (q || q') \Rightarrow (p;q) || (p';q')$$
• by transitivity:
  $$(p || p') ; (q || q') \Rightarrow r || r'$$
which is the conclusion of the modularity rule
Frame Rules

\[
\{p\} q \{r\}
\]
\[
\{p \mid \mid f\} q \{r \mid \mid f\}
\]
– adapts a triple to a concurrent environment \( f \)
– proof: from frame theorem

\[
\underline{\{p\} q \{r\}}
\]
\[
\{f ; p\} q \{f ; r\}
\]
– proof: mon, assoc of ;
The Milner triple: \( r \rightarrow q \rightarrow p \)

- defined as \( q;p \rightarrow r \)
  - (the time reversal of \( \{p\} q \{r\} \))
- \( r \) may be executed by first executing \( q \)
  - with \( p \) as continuation for later execution.
  - maybe there are other ways of executing \( r \)

- Tautology: \( (q ; p) \rightarrow q \rightarrow p \) (CCS)
- Proof: from reflexivity: \( q;p \rightarrow q;p \)
Technical Objection

• 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.
• These restrictions are useful in application.
• And so is their removal in theory
  – (provided that the axioms are still consistent).
Sequential composition

\[\{p\} q' \{s\} \implies \{s\} q \{r\}\]
\[\{p\} q'; q \{r\}\]

\[r \rightarrow q -> s \quad \rightarrow s \rightarrow q' -> p\]
\[r \rightarrow (q; q') -> p\]

Proof: by time-reversal of the Hoare rule
Concurrenty in CCS

\[ r -\sigma \rightarrow q \quad r' -\sigma' \rightarrow q' \]
\[ (r \parallel r') - (p \parallel p') \rightarrow (q \parallel q') \]

Proof: by time-reversal of the modularity rule

• In Milner’s CCS, the rule is applied only if \( p \) and \( p' \) are synchronised, e.g., input and output on the same channel.
Frame Rules

\[
\begin{align*}
  &\quad r \rightarrow q \rightarrow p \\
  \Rightarrow \\
  (r || f) \rightarrow q \rightarrow (p || f)
\end{align*}
\]

– a step q possible for a single thread r is still possible when r is executed concurrently with f

\[
\begin{align*}
  &\quad r \rightarrow q \rightarrow p \\
  \Rightarrow \\
  (r; f) \rightarrow (p; f)
\end{align*}
\]

– operational definition of ;
The internal step

• $r \rightarrow p = \text{def. } p \rightarrow r$
  – (the order reversal of refinement)

• implementation may make a refinement step
  – reducing the range of subsequent behaviours
Rule of consequence

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

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

- Each rule is the dual of the other 
  - by order reversal and time reversal
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.

• Deductive and operational semantics are mutually consistent, provided the laws are true
3. The laws are true of a realistic mathematical model of real program behaviour
Behaviours

• are sets of events
  – occurring in and around a computer
  – that is executing a program

• Let $\textbf{Ev}$ be the set of all occurrences
  – of all such events
  – that ever were, or ever could be
Happens before (\(\rightarrow\))

• Let \(e, f, g \in \text{Ev}\) (sets of event occurrences).
• \(e \rightarrow f\) is intended to mean (your choice of):
  – “the occurrence \(e\) is an immediate and necessary cause of the occurrence \(f\)”
  – “the occurrence \(f\) directly depends (depended) on the occurrence \(e\)”
  – “\(e\) happens before \(f\)” “\(f\) happens after \(e\)”
Examples: software

• $n^{th}$ output $\rightarrow$ $n^{th}$ input (on a reliable channel)
• $n^{th}$ V (acquire) $\rightarrow$ $n^{th}$ P (release) (on an exclusion semaphore)
• $n^{th}$ assignment $\rightarrow$ read of the $n^{th}$ value assigned (to a variable in memory)
• read of $n^{th}$ value $\rightarrow$ $(n + 1)^{st}$ assignment (in strong memory)
Precedes/follows

• Define \( \leq \) as \((\rightarrow)^*\)
  
  – the reflexive transitive closure of \(\rightarrow\)
  
  – Define \(\geq\) as \(\leq^\circ\) (the converse of \(\leq\))

• Examples:
  
  – allocation of a resource \(\leq\) every use of it
  
  – disposal of a resource \(\geq\) every use of it
Interpretations

• \( e \leq f \quad \& \quad f \leq e \) means
  – e and f are (parts of) the same atomic action

• not \( e \leq f \quad \& \quad \text{not } f \leq e \) means
  – e and f are independent of each other
  – their executions may overlap in time,
  – or one may complete before the other starts
Cartesian product

• Let $p, q, r \subseteq Ev$
  
  – behaviours are sets of event occurrences

• Define $p \times q = \{(e,f) \mid e \in p \& f \in q\}$
  
  – the Cartesian product

• Theorem: $p \times (q \cup r) = p \times q \cup p \times r$
  
  $(q \cup r) \times p = q \times p \cup r \times p$
Composition

• Let \( p, q, r \subseteq Ev \) (behaviours)
• Let \( seq \subseteq Ev \times Ev \) (arbitrary relation)
• Define \( p; q = p \cup q \) if \( p \times q \subseteq seq \)
  & \( p, q \) are defined
  – and is undefined otherwise
• Define \( p \sqsubseteq q \) as \( p = q \) or \( p \) is undefined

Theorem: \( ; \) is monotonic wrto \( \leq \)
Theorem: \((p \; q) \; r = p \; (q \; r)\)

- **Proof:** when they are both defined, each side is equal to \((p \cup q \cup r)\).

- We still need to prove that LHS is defined iff ant RHS is defined.
Theorem:  \((p \ ; q) \ ; r = p \ ; (q \ ; r)\)

LHS is defined  \text{iff} \quad (\text{by definition of } ;)

\[ p \times q \subseteq \text{seq} \ \& \ \ (p \cup q) \times r \subseteq \text{seq} \]

iff  \[ p \times q \subseteq \text{seq} \ \& \ \ p \times r \subseteq \text{seq} \ \& \ \ q \times r \subseteq \text{seq} \ (*) \]

iff  \[ p \times (q \cup r) \subseteq \text{seq} \ \& \ \ q \times r \subseteq \text{seq} \ (*) \]

iff  RHS is defined

*(by \times \text{ distrib } \cup)*
Sequential composition (strong)

• Define \( \text{seq} = \leq \)

• Then \( p;q \) is (strong) sequential composition

• means that \( p \) must finish before \( q \) starts

  – every event in \( p \) comes before every event in \( p \)

• Example: \( \text{Ev} \) is \( \text{NN} < \leq \text{is numerical} <\)

  – \{1, 7, 19\} ; \{21, 32\} = \{1, 7, 19, 21, 32\}

  – \{1, 7, 19\} ; \{19, 32\} is undefined
Sequential composition (weak)

• Define $\text{seq} = \text{not} \geq$

• Then $p; q$ is (weak) sequential composition

• means that $p$ can finish before $q$ starts
  – no event in $q$ comes before any event in $p$
  – but $q$ can often start before end of $p$,

  provided the exchanged events are independent.
Concurrent Composition

Define \( \text{par} = \text{Ev} \times \text{Ev} \)

Note: \( \text{seq} \subseteq \text{par} = \text{par}^\circ \) (converse)

Theorem: \( \text{pxq} \subseteq \text{par} \)

Define \( p \mid\mid q = p \cup q \)

Theorem: \( \mid\mid \) is associative and commutative.

and satisfies exchange law with ; (weak)
Examples

- Example: $E_B$ is $\mathbb{N}$

- $\{1, 7, 19\} \cup \{21, 32\} = \{1, 7, 19, 21, 32\}$
- $\{1, 7, 19\} \cup \{19, 32\}$ is undefined
- $\{1, 7, 19\} \cup \{3, 10, 32\} = \{1, 3, 7, 10, 19, 32\}$
\[(q \parallel q') \land (r \parallel r') \Rightarrow (q ; r) \parallel (q' ; r')\]

- Proof: when LHS is defined, it equals RHS
  \[q \cup r \cup q' \cup r'\]
\[(q \parallel q') \; ; \; (r \parallel r') \implies (q \; ; \; r) \parallel (q' \; ; \; r')\]

LHS defined \iff \(q \times q' \subseteq \text{par} \; \& \; \; r \times r' \subseteq \text{par}\)

\& \( (q \cup q') \times (r \cup r') \subseteq \text{seq}\)

implies \(q' \times r' \cup q \times r \cup q' \times r \cup q \times r' \subseteq \text{seq}\)

implies \(q' \times r' \subseteq \text{seq} \; \& \; q \times r \subseteq \text{seq}\)

\& \( (q \cup r) \times (q' \cup r') \subseteq \text{par}\)

implies RHS defined.
4. The laws are useful
Tools for Software Engineering

Verification       Compilation

Testing
based on semantics

Verification
deductive (Hoare)

Compilation
operational (Milner)

Testing
denotational (Scott)
Laws prove consistency

- Verification deductive
- Compilation operational
- Testing denotational
5. Conclusion
The Laws

• The laws are useful
  – they shorten formulae, theorems, proofs
  – they prove consistency of proof rules
    with the implementation

• The laws are true
  – of specifications, designs, products
  – hardware/software/the real world

• The laws are beautiful
Isaac Newton

Communication with Richard Gregory (1694)

“Our specious algebra [the infinitesimal calculus] is fit enough to find out [is ok as a heuristic], 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.
Examples: hardware

• a rising edge $\rightarrow$ next falling edge on same wire

• a rising edge $\rightarrow$ rising edge on another wire
Example: Petri nets

\[ e \rightarrow f' \quad \& \quad f' \rightarrow \quad g \]

\[ e \rightarrow f \quad \& \quad f \rightarrow \quad g \]
Message sequence chart

app    sql    net
Additional operators

• $p \lor q$ describes all traces of $p$ and all of $q$
  – describes options in design
  – choice (non-determinism) in execution

• $p \land q$ describes all traces of both $p$ and $q$
  – conjunction of requirements (aspects) in design
  – lock-step concurrency in execution
Axioms

• \( \lor \) is the disjunction and \( \land \) is the conjunction of a Boolean Algebra (even with negation).

• Axiom: \( ; \) and \( || \) distribute through \( \lor \)
  – which validates reasoning by cases
  – and implementation by non-deterministic selection
Choice

• \( \{p\} q \{r\} \quad \{p\} q' \{r\} \)
  \( \{p\} (q \lor q') \{r\} \)
  – both choices must be correct
  – proof: distribution of \( ; \) through \( \lor \)

\[ \begin{array}{c}
   r \Rightarrow p \\
  \hline
  (r \lor r') \Rightarrow p \\
\end{array} \]

– only one of the alternatives is executed
– proof: \( r \Rightarrow r \lor r' \)
## Axioms proved from calculi

<table>
<thead>
<tr>
<th>from Hoare</th>
<th>from Milner</th>
</tr>
</thead>
<tbody>
<tr>
<td>• $p ; (q \vee r) \Rightarrow p ; q \vee p ; r$</td>
<td>• $(p \vee q) ; r \Rightarrow (p ; r) \vee (q ; r)$</td>
</tr>
<tr>
<td>• $p ; r \vee q ; r \Rightarrow (p \vee q) ; r$</td>
<td>• $p ; q \vee p ; r \Rightarrow p ; (q \vee r)$</td>
</tr>
</tbody>
</table>

| from both                  |
|-----------------|-----------------|
| • $p ; (q ; r) \Rightarrow (p ; q) ; r$ | • $(p ; q) ; r \Rightarrow p ; (q ; r)$ |
| • exchange law  |