Formal Calculation Unifying Engineering Theories Beyond Software

Raymond Boute, INTEC, Ghent University Marktoberdorf 2004/08/13

Overview

1. Introduction: motivation and approach
2. The formalism, part A: language
3. The formalism, part B: formal rules
4. Examples I: Systems Theory
5. Examples II: Computing Science
6. Examples III: Common Aspects
7. Conclusions — A formalism for Electrical and Computer engineering

1 Introduction

1.1 Motivation: rift between engineering theories

- Parnas:

> Professional engineers can often be distinguished from other designers by the engineers' ability to use mathematical models to describe and analyze their products.

- Observation: rift in practice
  - In classical engineering (electrical, mechanical, civil): established *de facto*
  - In software "engineering": mathematical models rarely used (occasionally in critical systems under the name "Formal Methods")
- Causes
  - Different degree of preparation,
  - Divergent mathematical methodology and style

- Methodology rift mirrors $\boxed{\text{style breach}}$ throughout mathematics
  - In long-standing areas of mathematics (algebra, analysis, etc.):
    style of calculation essentially formal ("letting symbols do the work")

    Examples:

From: Blahut / data compacting

$$
\begin{aligned}
& \frac{1}{n} \sum_{\mathbf{x}} p^n(\mathbf{x}|\theta) l_n(\mathbf{x}) \\
& \quad \leq \quad \frac{1}{n} \sum_{\mathbf{x}} p^n(\mathbf{x}|\theta)[1 - \log q^n(\mathbf{x})] \\
& \quad = \quad \frac{1}{n} + \frac{1}{n} L(\mathbf{p}^n; \mathbf{q}^n) + H_n(\theta) \\
& \quad = \quad \frac{1}{n} + \frac{1}{n} d(\mathbf{p}^n, \mathcal{G}) + H_n(\theta) \\
& \quad \leq \quad \frac{2}{n} + H_n(\theta)
\end{aligned}
$$

From: Bracewell / transforms

$$
\begin{aligned}
F(s) &= \int_{-\infty}^{+\infty} e^{-|x|} e^{-i2\pi xs} dx \\
&= 2 \int_{0}^{+\infty} e^{-x} \cos 2\pi xs \, dx \\
&= 2 \ \mathrm{Re} \int_{0}^{+\infty} e^{-x} e^{i2\pi xs} dx \\
&= 2 \ \mathrm{Re} \frac{-1}{i2\pi s - 1} \\
&= \frac{2}{4\pi^2 s^2 + 1}.
\end{aligned}
$$

– Major defect: supporting logical arguments highly informal

> "The notation of elementary school arithmetic, which nowadays everyone takes for granted, took centuries to develop. There was an intermediate stage called *syncopation*, using abbreviations for the words for addition, square, root, *etc.* For example Rafael Bombelli (*c.* 1560) would write
>
> R. c. L. 2 p. di m. 11 L for our $3\sqrt{2 + 11i}$.
>
> Many professional mathematicians to this day use the quantifiers ($\forall$,$\exists$) in a similar fashion,
>
> $\exists \delta > 0$ s.t. $|f(x) - f(x_0)| < \epsilon$ if $|x - x_0| < \delta$, for all $\epsilon > 0$,
>
> in spite of the efforts of [Frege, Peano, Russell] [...]. Even now, mathematics students are expected to learn complicated (ϵ-δ)-proofs in analysis with no help in understanding the logical structure of the arguments. Examiners fully deserve the garbage that they get in return."
>
> (P. Taylor, "Practical Foundations of Mathematics")

– Similar situation in Computing Science: even in formal areas (semantics), style of theory development is similar to analysis texts.

1.2 Principle: formal calculation

- Mathematical styles
  - "formal" = manipulating expressions on the basis of their *form*
  - "informal" = manipulating expressions on the basis of their *meaning*
- Advantages of formality
  - Usual arguments: precision, reliability of design etc. well-known
  - Equally (or more) important: *guidance in expression manipulation*
    Calculations guided by the shape of the formulas
  - For engineering theories: leads to common style and methodology
    Abstraction from subject-specific idioms

$\boxed{\text{UT FACIANT OPUS SIGNA}}$

"Let the symbols do the work"
(Maxim of the conferences on *Mathematics of Program Construction*)

Provides help in *thinking*: acquiring feeling for the *shape* of formulas
$\rightarrow$ an additional kind of / added dimension to intuition!

1.3 Realization: Functional Mathematics (Funmath)

- Unifying formalism for continuous and discrete mathematics
  - Formalism = notation (language) + formal manipulation rules
- Characteristics
  - Principle: functions as first-class objects and basis for unification
  - Language: very simple (4 constructs only)

    Synthesizes common notations, without their defects

    Synthesizes new useful forms of expression, in particular: "point-free", e.g. $square = times \circ duplicate$ versus $square\, x = x\, times\, x$
  - Formal rules: *calculational*

2 The formalism, part A: language

2.1 Rationale: the need for defect-free notation

Examples of defects in mathematical conventions

Examples A: defects in often-used conventions relevant to systems theory

- Ellipsis, i.e., dots $(\ldots)$ as in $a_0 + a_1 + \cdots + a_n$

  Common use violates Leibniz's principle (substitution of equals for equals)

  Example: $a_i = i^2$ and $n = 7$ yields $0 + 1 + \cdots + 49$ (probably not intended!)

- Summation sign $\sum$ not as well-understood as often assumed.

  Example: error in *Mathematica*: $\sum_{i=1}^{n} \sum_{j=i}^{m} 1 \ = \ \frac{n \cdot (2 \cdot m - n + 1)}{2}$

  Taking $n := 3$ and $m := 1$ yields 0 instead of the correct sum 1.

- Confusing function application with the function itself

  Example: $y(t) = x(t) * h(t)$ where $*$ is convolution.

  Causes incorrect instantiation, e.g., $y(t - \tau) = x(t - \tau) * h(t - \tau)$

Examples B: ambiguities in conventions for sets

- Patterns typical in mathematical writing:
  (assuming logical expression p, arbitrary expression p

Patterns	$\{x \in X \mid p\}$	and	$\{e \mid x \in X\}$
Examples	$\{m \in \mathbb{Z} \mid m < n\}$	and	$\{n \cdot m \mid m \in \mathbb{Z}\}$

  The usual tacit convention is that $\in$ binds x. This seems innocuous, BUT

- Ambiguity is revealed in case p or e is itself of the form $y \in Y$.
  Example: let $Even := \{2 \cdot m \mid m \in \mathbb{Z}\}$ in

Patterns	$\{x \in X \mid p\}$	and	$\{e \mid x \in X\}$
Examples	$\{n \in \mathbb{Z} \mid n \in Even\}$	and	$\{n \in Even \mid n \in \mathbb{Z}\}$

  Both examples match *both patterns*, thereby illustrating the ambiguity.

- Worse: such defects *prohibit even the formulation of calculation rules*!
  Formal calculation with set expressions rare/nonexistent in the literature.

Underlying cause: overloading relational operator $\in$ for binding of a dummy.
This poor convention is ubiquitous (not only for sets), as in $\forall\, x \in \mathbb{R} \,.\, x^2 \geq 0$.

2.2 Funmath language design

Basis: *function* $(=$ *domain* $+$ *mapping*)
Language syntax : 4 constructs: identifier, application, abstraction, tupling

0. Identifier: any symbol or string except a few keywords.

   Identifiers are *introduced* by *bindings*

   - General form: $\boxed{i : X \wedge p}$, read "i in X satisfying p"
     Here i is the (tuple of) identifier(s), X a set and p a proposition.
     Optional: *filter* $\wedge\, p$ (or $\textbf{with}\ p$), e.g., $\boxed{n : \mathbb{N}}$ is same as $\boxed{n : \mathbb{Z} \wedge n \geq 0}$
     Identifiers from i should not appear in expression X.
   - Identifiers can be
     variables: in an *abstraction* of the form $\mathit{binding} \,.\, \mathit{expression}$
     constants: declared by a *definition* of the form $\textbf{def}\ \mathit{binding}$

   Well-established symbols, such as $\mathbb{B}$, $\Rightarrow$, $\mathbb{R}$, $+$, serve as predefined constants.

1. Function application:

- Default form: $\boxed{f\,x}$ for function f and argument e
- Other affix conventions: by dashes in the binding, e.g., $— \star —$ for infix.
- Role of parentheses: *never* used as operators.

  Only for parsing (overruling/emphasizing affix conventions/precedence).

  Precedence rules for making parentheses optional are the usual ones.

  If f is a function-valued function, $\boxed{f\,x\,y \text{ stands for } (f\,x)\,y}$
- Special application forms for any infix operator $\star$
  - *Partial application* is of the form $a\star$ or $\star b$, and is defined by

    $$\boxed{(a\star)\,b = a \star b = (\star\,b)\,a}$$

  - *Variadic application* is of the form $a \star b \star c$ etc., *always* defined by

    $$\boxed{a \star b \star c = F\,(a, b, c)}$$

    for a suitably defined *elastic extension* F of $\star$.

2. Abstraction:

- General form: $\boxed{b \,.\, e}$ where

  b is a binding and

  e an expression, extending after "." as far as parentheses permit.

  Intuitive meaning: $v : X \wedge p \,.\, e$ denotes a *function*

  Domain $=$ the set of v in X satisfying p;

  Mapping: maps v to e.
- Trivial example (constant functions): if v not free in e, we define $^\bullet$ by $\boxed{X^\bullet e = v : X \,.\, e}$. Example: $(\mathbb{Z}^\bullet 3)\,7 = 3$.
- Syntactic sugar: $\boxed{e \mid b \text{ stands for } b \,.\, e}$ and $\boxed{v : X \mid p \text{ stands for } v : X \wedge p \,.\, v}$.
- We shall see how abstractions help synthesizing familiar expressions such as $\boxed{\sum i : 0 \mathinner{..} n \,.\, q^i}$ and $\boxed{\{m \cdot n \mid m : \mathbb{Z}\}}$ and $\boxed{\{m : \mathbb{Z} \mid m < n\}}$.

3. Tupling:

- 1-dimensional form: $\boxed{e, e', e''}$ (any length)
  Intuitive meaning: function with
  Domain: $\mathcal{D}\,(e, e', e'') = \{0, 1, 2\}$
  Mapping: $(e, e', e'')\,0 = e$ and $(e, e', e'')\,1 = e'$ and $(e, e', e'')\,2 = e''$.
- Parentheses are *not* part of tupling: as optional in (m, n) as in $(m + n)$.
- The empty tuple is ε and for singleton tuples we define τ with $\tau\, e = 0 \mapsto e$.
- Matrices are 2-dimensional tuples.

Legend: here we used two special cases of $^\bullet$:

defining ε by $\varepsilon := \emptyset^\bullet\, e$ (any e) for the *empty function*

defining $\mapsto$ by $d \mapsto e = \iota\, d^\bullet\, e$ for *one-point functions*.

3 The formalism, part B: formal rules

3.1 Rules for equational and calculational reasoning

- Calculational reasoning: Generalizes the usual chaining of calculation steps to

$$\begin{array}{llll} e_0 & R_0 \,\langle \text{Justification}_0 \rangle & e_1 & \\ & R_1 \,\langle \text{Justification}_1 \rangle & e_2 & \text{etc.} \end{array}$$

  where R_i, R_{i+1} are mutually transitive, e.g., $=$, $\leq$ (arithmetic), $\equiv$, $\Rightarrow$ (logic).

- General inference rule: For any theorem p,

  INSTANTIATION: from p, infer $p[^v_e$.

  Note: $[^v_e$ or $[v := e]$ expresses substitution of e for v, for instance,
  $(x + y = y + x)[x, y := 3, z + 1]$ stands for $3 + (z + 1) = (z + 1) + 3$.

- Equational reasoning: basic rules are reflexivity, symmetry, transitivity and

  LEIBNIZ'S PRINCIPLE: from $e = e'$, infer $d[^v_e = d[^v_{e'}$

3.2 Rules for calculating with propositions and sets

- Proposition calculus Usual propositional operators $\neg$, $\equiv$, $\Rightarrow$, $\wedge$, $\vee$. Notes:
  - For practical use, an extensive set of rules is needed (see e.g. Gries)
  - Note: $\equiv$ is associative, $\Rightarrow$ is not. We read $p \Rightarrow q \Rightarrow r$ as $p \Rightarrow (q \Rightarrow r)$.
  - Binary algebra is embedded in arithmetic. Logic constants are 0 and 1.
  - Leibniz's principle can be rewritten $e = e' \Rightarrow d[^v_e = d[^v_{e'}$.
- Calculating with sets The basic operator is $\in$.
  - The rules are derived ones (set calculus from proposition calculus), e.g.,

Set intersection $\cap$ is defined by	$x \in X \cap Y \;\equiv\; x \in X \wedge x \in Y$
Cartesian product $\times$ is defined by	$x, y \in X \times Y \;\equiv\; x \in X \wedge y \in Y$
After defining $\{$—$\}$, we can prove	$y \in \{x : X \mid p\} \;\equiv\; y \in X \wedge p[^x_y$

  - *Set equality* is defined via

Leibniz's principle: $X = Y \Rightarrow (x \in X \;\equiv\; x \in Y)$, and the converse:
Extensionality: from $x \in X \;\equiv\; x \in Y$ (with new x), infer $X = Y$.

3.3 Rules for calculating with functions and generic functionals

- General rules for functions

  – *Equality* is defined (taking domains into account) via

Leibniz's principle	$f = g \Rightarrow \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, f \cap \mathcal{D}\, g \Rightarrow f\, x = g\, x)$
Extensionality	$\dfrac{p \Rightarrow \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, f \cap \mathcal{D}\, g \Rightarrow f\, x = g\, x)}{p \Rightarrow f = g}$

  – Abstraction encapsulates substitution. Formal axioms:

Domain axiom:	$d \in \mathcal{D}\,(v : X \wedge\!\!\!. \; p \,.\, e) \;\equiv\; d \in X \wedge p[^v_d$
Mapping axiom:	$d \in \mathcal{D}\,(v : X \wedge\!\!\!. \; p \,.\, e) \Rightarrow (v : X \wedge\!\!\!. \; p \,.\, e)\, d = e[^v_d$

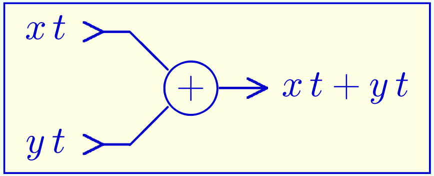
  Equality is characterized via function equality (exercise).

- Generic functionals
  - Goals:

    (a) Removing restrictions in common functionals from mathematics.

    Example: composition $f \circ g$; common definition requires $\mathcal{R}\, g \subseteq \mathcal{D}\, f$

    (b) Making often-used implicit functionals from systems theory explicit.



    Usual notations: $(x + y)\, t = x\, t + y\, t$ (overloading $+$)

    or: $(x \oplus y)\, t = x\, t + y\, t$ (special symbol)

  - Design principle: defining the domain of the result function in such a way that the image definition does not involve out-of-domain applications.

    This applies to goal (a), goal (b) and new designs (discussed next).

– Design illustrating goal (a): *composition* ($\circ$)
For *any* functions f, g,

$$\boxed{f \circ g \;=\; x : \mathcal{D}\, g \wedge g\, x \in \mathcal{D}\, f \,.\, f\,(g\, x)}$$

Observation: $\mathcal{D}\,(f \circ g) = \{x : \mathcal{D}\, g \mid g\, x \in \mathcal{D}\, f\}$.

– Design illustrating goal (b): *(Duplex) direct extension* ($\widehat{\ }$)
For any functions $\star$ (infix), f, g,

$$\boxed{f \mathbin{\widehat{\star}} g \;=\; x : \mathcal{D}\, f \cap \mathcal{D}\, g \wedge (f\, x, g\, x) \in \mathcal{D}\,(\star) \,.\, f\, x \star g\, x}$$

Example: given $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{Z} \to \mathbb{C}$ we get $\mathcal{D}\,(f \mathbin{\widehat{+}} g) = \mathbb{N}$.
Often we need *half direct extension*: for function f, any e,

$$\boxed{f \mathbin{\overleftharpoon{\star}} e = f \mathbin{\widehat{\star}} (\mathcal{D}\, f \,{}^\bullet\, e) \quad \text{and} \quad e \mathbin{\overrightharpoon{\star}} f = (\mathcal{D}\, f \,{}^\bullet\, e) \mathbin{\widehat{\star}} f}$$

Typical algebraic property: $x \mathbin{\overrightharpoon{\star}} f = (x \star) \circ f$
Simplex direct extension ($\overline{\ }$) is defined by

$$\boxed{\overline{f}\, g = f \circ g}$$

- Generic functionals (continued:) some other important generic functionals

  – *Function merge* ($\dot{\cup}$) is defined in 2 parts to fit the line:

$$\boxed{\begin{array}{lcl} x \in \mathcal{D}\,(f \mathbin{\dot{\cup}} g) & \equiv & x \in \mathcal{D}\,f \cup \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \\ x \in \mathcal{D}\,(f \mathbin{\dot{\cup}} g) & \Rightarrow & (f \mathbin{\dot{\cup}} g)\,x = (x \in \mathcal{D}\,f)\,?\,f\,x \mathbin{\nmid} g\,x \end{array}}$$

  – *Filtering* ($\downarrow$) introduces/eliminates arguments: (here P is a predicate)

$$\boxed{f \downarrow P \;\; = \;\; x : \mathcal{D}\,f \cap \mathcal{D}\,P \mathbin{\underset{\cdot}{\wedge}} P\,x\,.\,f\,x}$$

  A particularization is the familiar *restriction* ($\rceil$): $f \rceil X \;=\; f \downarrow (X^{\bullet}\,1)$.

  We extend $\downarrow$ to sets: $x \in (X \downarrow P) \equiv x \in X \cap \mathcal{D}\,P \wedge P\,x$.

  Writing a_b for $a \downarrow b$ and using partial application, this yields formal rules for useful shorthands like $f_{<n}$ and $\mathbb{Z}_{>0}$.

  – *Function compatibility* (©) is a relation on functions:

$$\boxed{f \mathbin{\copyright} g \;\; \equiv \;\; f \rceil \mathcal{D}\,g = g \rceil \mathcal{D}\,f}$$

  Algebraic property: $f = g \equiv \mathcal{D}\,f = \mathcal{D}\,g \wedge f \mathbin{\copyright} g$.

3.4 Rules for calculating with predicates and quantifiers

Goal: formally calculating with quantifiers as fluently as with derivatives/integrals.
Practical use requires a large collection of calculation rules.
Here only give the axioms and most important derived rules.

- Axioms and forms of expression
  - Basic axioms: *quantifiers* ($\forall$, $\exists$) are predicates on predicates defined by

$$\boxed{\forall\, P \equiv P = \mathcal{D}\, P^{\bullet} 1 \quad \text{and} \quad \exists\, P \equiv P \neq \mathcal{D}\, P^{\bullet} 0}$$

  - Forms of expression
    Taking for P an abstraction yields familiar forms like $\forall\, x : \mathbb{R} \,.\, x \geq 0$.
    Taking for P a pair p, q of boolean expressions yields $\forall\, (p, q) \equiv p \wedge q$.
    So $\forall$ is an elastic extension of $\wedge$, and we define $p \wedge q \wedge r \equiv \forall\, (p, q, r)$

- Derived rules

Relating $\forall/\exists$ by *duality* (or *generalized De Morgan's law*)

$\boxed{\neg\forall P = \exists(\overline{\neg}\, P) \text{ or, in pointwise form, } \neg(\forall\, v : S\,.\,p) \equiv \exists\, v : S\,.\,\neg\, p}$

Distributivity rules (each has a dual, not stated here):

Name of the rule	Point-free form	Letting $P := v : S\,.\,p$ with $v \notin \varphi q$
Distributivity $\vee/\forall$	$q \vee \forall P \equiv \forall(q \overset{\rightharpoonup}{\vee} P)$	$q \vee \forall(v : S\,.\,p) \equiv \forall(v : S\,.\,q \vee p)$
L(eft)-distrib. $\Rightarrow/\forall$	$q \Rightarrow \forall P \equiv \forall(q \overset{\rightharpoonup}{\Rightarrow} P)$	$q \Rightarrow \forall(v : S\,.\,p) \equiv \forall(v : S\,.\,q \Rightarrow p)$
R(ight)-distr. $\Rightarrow/\exists$	$\exists P \Rightarrow q \equiv \forall(P \overset{\leftharpoonup}{\Rightarrow} q)$	$\exists(v : S\,.\,p) \Rightarrow q \equiv \forall(v : S\,.\,p \Rightarrow q)$
P(seudo)-dist. $\wedge/\forall$	$q \wedge \forall P \equiv \forall(q \overset{\rightharpoonup}{\wedge} P)$	$q \wedge \forall(v : S\,.\,p) \equiv \forall(v : S\,.\,q \wedge p)$

Note: $\wedge/\forall$ assumes $\mathcal{D}\, P \neq \emptyset$. The general form is $(p \wedge \forall P) \vee \mathcal{D}\, P = \emptyset \equiv \forall(p \overset{\rightharpoonup}{\wedge} P)$

As in algebra, the nomenclature is very helpful for familiarization and use.

Distributivity $\vee/\forall$ generalizes $q \vee (r \wedge s) \equiv (q \vee r) \wedge (q \vee s)$

L(eft)-distrib. $\Rightarrow/\forall$ generalizes $q \Rightarrow (r \wedge s) \equiv (q \Rightarrow r) \wedge (q \Rightarrow s)$

R(ight)-distr. $\Rightarrow/\exists$ generalizes $(r \vee s) \Rightarrow q \equiv (r \Rightarrow q) \wedge (s \Rightarrow q)$

P(seudo)-dist. $\wedge/\forall$ generalizes $q \wedge (r \wedge s) \equiv (q \wedge r) \wedge (q \wedge s)$

- Derived rules (continued)

  Some additional laws

Name	Point-free form	Letting $P := v : S \,.\, p$ with $v \notin \varphi q$
Distrib. $\forall/\wedge$	$\forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\,P \wedge \forall\,Q$	$\forall\,(v : S \,.\, p \wedge q) \equiv \forall\,(v : S \,.\, p) \wedge \forall\,(v : S \,.\, q)$
One-point rule	$\forall\,P_{=e} \equiv e \in \mathcal{D}\,P \Rightarrow P\,e$	$\forall\,(v : S \,.\, v = e \Rightarrow p) \equiv e \in S \Rightarrow p[^v_e$
Trading $\forall$	$\forall\,P_Q \equiv \forall\,(Q \mathbin{\widehat{\Rightarrow}} P)$	$\forall\,(v : S \wedge q \,.\, p) \equiv \forall\,(v : S \,.\, q \Rightarrow p)$
Transp./Swap	$\forall\,(\forall \circ R) = \forall\,(\forall \circ R^{\mathsf{T}})$	$\forall\,(v : S \,.\, \forall\,w : T \,.\, p) \equiv \forall\,(w : T \,.\, \forall\,v : S \,.\, p)$

Note: $\forall/\wedge$ assumes $\mathcal{D}\,P = \mathcal{D}\,Q$. Without this condition, $\forall\,P \wedge \forall\,Q \Rightarrow \forall\,(P \mathbin{\widehat{\wedge}} Q)$.

Just one derivation example:

$$
\begin{array}{lll}
& \forall\,P \wedge \forall\,Q & \\
\equiv & \langle \text{Def. } \forall \rangle & P = \mathcal{D}\,P^{\bullet}\,1 \wedge Q = \mathcal{D}\,Q^{\bullet}\,1 \\
\Rightarrow & \langle \text{Leibniz} \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\,(\mathcal{D}\,P^{\bullet}\,1 \mathbin{\widehat{\wedge}} \mathcal{D}\,Q^{\bullet}\,1) \\
\equiv & \langle \text{Def. } \widehat{\ } \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\,x : \mathcal{D}\,P \cap \mathcal{D}\,Q \,.\, (\mathcal{D}\,P^{\bullet}\,1)\,x \wedge (\mathcal{D}\,Q^{\bullet}\,1)\,x \\
\equiv & \langle \text{Def. } {}^{\bullet}) \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\,x : \mathcal{D}\,P \cap \mathcal{D}\,Q \,.\, 1 \wedge 1 \\
\equiv & \langle \forall\,(X^{\bullet}\,1) \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q)
\end{array}
$$

3.5 Wrapping up the package for functions

- Function range We define the range operator $\mathcal{R}$ by

$$\boxed{e \in \mathcal{R}\, f \equiv \exists\, x : \mathcal{D}\, f \,.\, f\, x = e \ .}$$

Consequence: $\boxed{\forall\, P \Rightarrow \forall\, (P \circ f)}$ and $\boxed{\mathcal{D}\, P \subseteq \mathcal{R}\, f \Rightarrow (\forall\, (P \circ f) \equiv \forall\, P)}$

pointwise form: $\boxed{\forall\, (y : \mathcal{R}\, f \,.\, p) \equiv \forall\, (x : \mathcal{D}\, f \,.\, p[^y_{f\,x})}$ ("dummy change").

The familiar *function arrow* ($\rightarrow$) $\boxed{f \in X \rightarrow Y \equiv \mathcal{D}\, f = X \wedge \mathcal{R}\, f \subseteq Y}$

- Set comprehension

Basis: we define $\{—\}$ as *fully interchangeable* with $\mathcal{R}$.

Consequence: defect-free set notation:

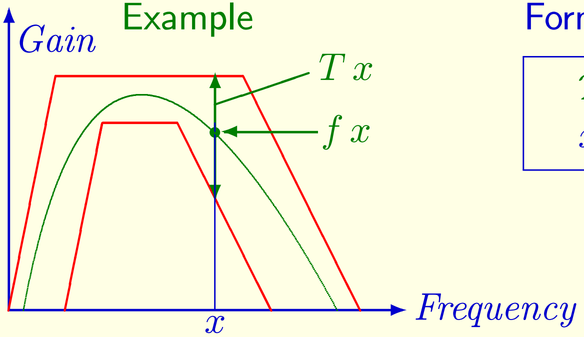
  - Expressions like $\{2, 3, 5\}$ and $\{2 \cdot m \mid m : \mathbb{Z}\}$ have familiar form & meaning
  - All desired calculation rules follow from predicate calculus via $\mathcal{R}$.
  - In particular, we can prove $e \in \{v : X \mid p\} \equiv e \in X \wedge p[^v_e$ (exercise).

3.6 Designing a generic operator from the function tolerance pparadigm

- Tolerances for functions: formalizing a convention in communications:

  A *tolerance function* T specifies for every domain value x the set $T\,x$ of allowable function values. Note: $\mathcal{D}\,T$ also taken as the domain specification.

  Example: radio frequency filter characteristic and its formalization



Formalization

$$\mathcal{D}\,f = \mathcal{D}\,T$$
$$x \in \mathcal{D}\,f \cap \mathcal{D}\,T \Rightarrow f\,x \in T\,x$$

- *Generalized Functional Cartesian Product* $\times$: for *any* family T of sets,

$$\boxed{\text{Definition:} \quad \times T \;=\; \{f : \mathcal{D}\,T \to \bigcup T \mid \forall\, x : \mathcal{D}\,f \cap \mathcal{D}\,T\,.\,f\,x \in T\,x\}}$$

Consequences: $\boxed{\times(X, Y) = X \times Y}$ and $\boxed{\times(X^{\bullet}\,Y) = X \to Y}$

4 Examples I: Systems Theory

4.1 Analysis: calculation replacing syncopation — an example

$$\mathbf{def}\ \mathsf{ad} : (\mathbb{R} \to \mathbb{B}) \to (\mathbb{R} \to \mathbb{B})\ \mathbf{with}\ \mathsf{ad}\, P\, v \equiv \forall\, \epsilon : \mathbb{R}_{>0} \,.\, \exists\, x : \mathbb{R}_P \,.\, |x - v| < \epsilon$$
$$\mathbf{def}\ \mathsf{open} : (\mathbb{R} \to \mathbb{B}) \to \mathbb{B}\ \mathbf{with}$$
$$\mathsf{open}\, P \equiv \forall\, v : \mathbb{R}_P \,.\, \exists\, \epsilon : \mathbb{R}_{>0} \,.\, \forall\, x : \mathbb{R} \,.\, |x - v| < \epsilon \Rightarrow P\, x$$
$$\mathbf{def}\ \mathsf{closed} : (\mathbb{R} \to \mathbb{B}) \to \mathbb{B}\ \mathbf{with}\ \mathsf{closed}\, P \equiv \mathsf{open}\, (\overline{\neg}\, P)$$

Example: proving the *closure property* $\boxed{\mathsf{closed}\, P \equiv \mathsf{ad}\, P = P}$.

$$\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{closed}\, P} \\
\equiv & \langle \text{Definit. closed} \rangle & \mathsf{open}\, (\overline{\neg}\, P) \\
\equiv & \langle \text{Definit. open} \rangle & \forall\, v : \mathbb{R}_{\overline{\neg} P} \,.\, \exists\, \epsilon : \mathbb{R}_{>0} \,.\, \forall\, x : \mathbb{R} \,.\, |x - v| < \epsilon \Rightarrow \overline{\neg}\, P\, x \\
\equiv & \langle \text{Trading sub } \forall \rangle & \forall\, v : \mathbb{R} \,.\, \overline{\neg}\, P\, v \Rightarrow \exists\, \epsilon : \mathbb{R}_{>0} \,.\, \forall\, x : \mathbb{R} \,.\, |x - v| < \epsilon \Rightarrow \overline{\neg}\, P\, x \\
\equiv & \langle \text{Contrapositive} \rangle & \forall\, v : \mathbb{R} \,.\, \overline{\neg}\, \exists\, (\epsilon : \mathbb{R}_{>0} \,.\, \forall\, x : \mathbb{R} \,.\, P\, x \Rightarrow \neg\, (|x - v| < \epsilon)) \Rightarrow P\, v \\
\equiv & \langle \text{Duality, twice} \rangle & \forall\, v : \mathbb{R} \,.\, \forall\, (\epsilon : \mathbb{R}_{>0} \,.\, \exists\, x : \mathbb{R} \,.\, P\, x \wedge |x - v| < \epsilon) \Rightarrow P\, v \\
\equiv & \langle \text{Definition ad} \rangle & \forall\, v : \mathbb{R} \,.\, \mathsf{ad}\, P\, v \Rightarrow P\, v \\
\equiv & \langle P\, v \Rightarrow \mathsf{ad}\, P\, v \rangle & \forall\, v : \mathbb{R} \,.\, \mathsf{ad}\, P\, v \equiv P\, v \ \text{(proving } P\, v \Rightarrow \mathsf{ad}\, P\, v \text{ is near-trivial)}
\end{array}$$

4.2 Transform methods

- Emphasis: formally correct use of functionals

  Avoiding common defective notations like $\mathcal{F}\{f(t)\}$ and writing $\mathcal{F}\, f\, \omega$ instead

  $$\boxed{\begin{array}{lcl} \mathcal{F}\, f\, \omega & = & \int_{-\infty}^{+\infty} e^{-j\cdot\omega\cdot t}\cdot f\, t\cdot \mathsf{d}t \\ \mathcal{F}'\, g\, t & = & \frac{1}{2\cdot\pi}\cdot\int_{-\infty}^{+\infty} e^{j\cdot\omega\cdot t}\cdot g\,\omega\cdot \mathsf{d}\omega \end{array}}$$

  Clear and unambiguous bindings allow formal calculation.

- Example: formalizing Laplace transforms via Fourier transforms.

  Auxiliary function: $\boxed{\ell_{\_} : \mathbb{R} \to \mathbb{R} \to \mathbb{R} \text{ with } \ell_\sigma\, t = (t < 0)\,?\,0 \nmid e^{-\sigma\cdot t}}$

  We define the Laplace-transform $\mathcal{L}\, f$ of a function f by:

  $$\boxed{\mathcal{L}\, f\,(\sigma + j\cdot\omega) \;=\; \mathcal{F}\,(\ell_\sigma \mathbin{\widehat{\cdot}} f)\,\omega}$$

  for real σ and ω, with σ such that $\ell_\sigma \mathbin{\widehat{\cdot}} f$ has a Fourier transform.

  With $s := \sigma + j\cdot\omega$ we obtain

  $$\boxed{\mathcal{L}\, f\, s = \int_0^{+\infty} f\, t\cdot e^{-s\cdot t}\cdot \mathsf{d}t\ .}$$

- Calculation example: the inverse Laplace transform

Specification of $\mathcal{L}'$: $\boxed{\mathcal{L}'\,(\mathcal{L}\,f)\,t = f\,t \text{ for all } t \geq 0}$
(weakened where $\ell_\sigma \mathbin{\widehat{\cdot}} f$ is discontinous).

Calculation of an explicit expression: For t as specified,

$$
\begin{array}{lcll}
\mathcal{L}'\,(\mathcal{L}\,f)\,t & = & \langle\text{Specification}\rangle & f\,t \\
 & = & \langle e^{\sigma\cdot t}\cdot\ell_\sigma\,t = 1\rangle & e^{\sigma\cdot t}\cdot\ell_\sigma\,t\cdot f\,t \\
 & = & \langle\text{Definition }\widehat{\ }\rangle & e^{\sigma\cdot t}\cdot(\ell_\sigma \mathbin{\widehat{\cdot}} f)\,t \\
 & = & \langle\text{Weakened}\rangle & e^{\sigma\cdot t}\cdot\mathcal{F}'\,(\mathcal{F}\,(\ell_\sigma \mathbin{\widehat{\cdot}} f))\,t \\
 & = & \langle\text{Definition }\mathcal{F}'\rangle & e^{\sigma\cdot t}\cdot\frac{1}{2\cdot\pi}\cdot\int_{-\infty}^{+\infty}\mathcal{F}\,(\ell_\sigma \mathbin{\widehat{\cdot}} f)\,\omega\cdot e^{j\cdot\omega\cdot t}\cdot\mathrm{d}\omega \\
 & = & \langle\text{Definition }\mathcal{L}\rangle & e^{\sigma\cdot t}\cdot\frac{1}{2\cdot\pi}\cdot\int_{-\infty}^{+\infty}\mathcal{L}\,f\,(\sigma+j\cdot\omega)\cdot e^{j\cdot\omega\cdot t}\cdot\mathrm{d}\omega \\
 & = & \langle\text{Const. factor}\rangle & \frac{1}{2\cdot\pi}\cdot\int_{-\infty}^{+\infty}\mathcal{L}\,f\,(\sigma+j\cdot\omega)\cdot e^{(\sigma+j\cdot\omega)\cdot t}\cdot\mathrm{d}\omega \\
 & = & \langle s := \sigma + j\cdot\omega\rangle & \frac{1}{2\cdot\pi\cdot j}\cdot\int_{\sigma-j\cdot\infty}^{\sigma+j\cdot\infty}\mathcal{L}\,f\,s\cdot e^{s\cdot t}\cdot\mathrm{d}s
\end{array}
$$

4.3 Characterization of properties of systems

- Definitions and conventions

  Define $\boxed{\mathcal{S}_A = \mathbb{T} \to A}$ for value space A and time domain $\mathbb{T}$. Then

  - A *signal* is a function of type $\mathcal{S}_A$
  - A *system* is a function of type $\mathcal{S}_A \to \mathcal{S}_B$.

  Note: the response of $s : \mathcal{S}_A \to \mathcal{S}_B$ to input signal $x : \mathcal{S}_A$ at time $t : \mathbb{T}$ is $s\,x\,t$.
  Recall: $s\,x\,t$ is read $(s\,x)\,t$, not to be confused with $s\,(x\,t)$.

- Characteristics Let $s : \mathcal{S}_A \to \mathcal{S}_B$. Then:
  - System s is

$$\boxed{\textit{memoryless} \text{ iff } \quad \exists\, f_{\_} : \mathbb{T} \to A \to B \,.\, \forall\, x : \mathcal{S}_A \,.\, \forall\, t : \mathbb{T} \,.\, s\,x\,t = f_t\,(x\,t)}$$

  - Let $\mathbb{T}$ be additive, and the *shift* function $\sigma_{\_}$ be defined by $\sigma_\tau\,x\,t = x\,(t+\tau)$ for any t and τ in $\mathbb{T}$ and any signal x. Then s is

$$\boxed{\textit{time-invariant} \text{ iff } \quad \forall\, \tau : \mathbb{T} \,.\, s \circ \sigma_\tau = \sigma_\tau \circ s}$$

  - Let now $s : \mathcal{S}_\mathbb{R} \to \mathcal{S}_\mathbb{R}$. Then system s is *linear* iff $\forall\,(x,y) : \mathcal{S}_\mathbb{R}^2 \,.\, \forall\,(a,b) : \mathbb{R}^2 \,.\, s\,(a \overset{\rightharpoonup}{\cdot} x \,\widehat{+}\, b \overset{\rightharpoonup}{\cdot} y) = a \overset{\rightharpoonup}{\cdot} s\,x \,\widehat{+}\, b \overset{\rightharpoonup}{\cdot} s\,y$. Equivalently, extending s to $\mathcal{S}_\mathbb{C} \to \mathcal{S}_\mathbb{C}$ in the evident way, system s is

$$\boxed{\textit{linear} \text{ iff } \quad \forall\, z : \mathcal{S}_\mathbb{C} \,.\, \forall\, c : \mathbb{C} \,.\, s\,(c \overset{\rightharpoonup}{\cdot} z) = c \overset{\rightharpoonup}{\cdot} s\,z}$$

  - A system is LTI iff it is both linear and time-invariant.

- Response of LTI systems

Define the parametrized exponential $\boxed{\mathsf{E}\_ : \mathbb{C} \to \mathbb{T} \to \mathbb{C} \text{ with } \mathsf{E}_c\, t = e^{c \cdot t}}$

Then we have:

$\boxed{\text{THEOREM: if } s \text{ is LTI then } s\, \mathsf{E}_c = s\, \mathsf{E}_c\, 0 \overset{\rightharpoonup}{\cdot} \mathsf{E}_c}$

Proof: we calculate $s\, \mathsf{E}_c\, (t + \tau)$ to exploit all properties.

$$\boxed{\begin{array}{rcll} s\, \mathsf{E}_c\, (t + \tau) & = & \langle \text{Definition } \sigma \rangle & \sigma_\tau\, (s\, \mathsf{E}_c)\, t \\ & = & \langle \text{Time inv. } s \rangle & s\, (\sigma_\tau\, \mathsf{E}_c)\, t \\ & = & \langle \text{Property } \mathsf{E}_c \rangle & s\, (\mathsf{E}_c\, \tau \overset{\rightharpoonup}{\cdot} \mathsf{E}_c)\, t \\ & = & \langle \text{Linearity } s \rangle & (\mathsf{E}_c\, \tau \overset{\rightharpoonup}{\cdot} s\, \mathsf{E}_c)\, t \\ & = & \langle \text{Defintion } \rightharpoonup \rangle & \mathsf{E}_c\, \tau \cdot s\, \mathsf{E}_c\, t \end{array}}$$

Substituting $t := 0$ yields $s\, \mathsf{E}_c\, \tau = s\, \mathsf{E}_c\, 0 \cdot \mathsf{E}_c\, \tau$ or, using $\rightharpoonup$,
$s\, \mathsf{E}_c\, \tau = (s\, \mathsf{E}_c\, 0 \overset{\rightharpoonup}{\cdot} \mathsf{E}_c)\, \tau$, so $s\, \mathsf{E}_c = s\, E_c\, 0 \overset{\rightharpoonup}{\cdot} \mathsf{E}_c$ by function equality.

The $\langle \text{Property } \mathsf{E}_c \rangle$ is $\sigma_\tau\, \mathsf{E}_c = \mathsf{E}_c\, \tau \overset{\rightharpoonup}{\cdot} \mathsf{E}_c$ (easy to prove).

Note that this proof uses only the essential hypotheses.

5 Examples II: Computing Science

5.1 From data structures to query languages

a. Aggregate data types (all aggregates are functions!) Some typical cases:

- List types: $\boxed{A^n = \times(\square\, n^{\bullet} A)}$ and $\boxed{A^* = \bigcup n : \mathbb{N} \,.\, A^n}$ and so on
- Record types: defining, for any $F : \mathsf{Fam}\,(\mathsf{Fam}\,\mathcal{T})$,

$$\boxed{\mathsf{Record}\, F = \times(\dot{\bigcup}\, F)}$$

  Example:

  Let $\boxed{Person := \mathsf{Record}\,(name \mapsto \mathbb{A}^*, age \mapsto \mathbb{N})}$

  Then $person : Person$ satisfies $person\; name \in \mathbb{A}^*$ and $person\; age \in \mathbb{N}$.

b. Overloading and polymorphism

- Aspects to be covered: disambiguation and refined typing
- Two main operators: (for family T of function types to be combined)
  - Parametrized (Church style): simply $\times T$
  - Unparametrized (Curry style): function type merge

$$\boxed{\mathbf{def}\ \otimes : \mathsf{Fam}\,(\mathcal{P}\,\mathcal{F}) \to \mathcal{P}\,\mathcal{F}\ \mathbf{with}\ \otimes T = \{\textstyle\bigcup\!\!\!\cdot\ F \mid F : \times T \wedge © F\}}$$

Note: for families F and G of functions: $F \otimes G = \otimes\,(F, G)$ or $F \otimes G = \{f \cup\!\!\!\cdot\, g \mid f, g : F \times G \wedge f © g\}$

c. Relational databases

- Formal description: by declarations (here explained by example)

$\boxed{\mathbf{def}\ CID := \mathsf{Record}\,(\mathsf{code} \mapsto Code, \mathsf{name} \mapsto \mathbb{A}^*, \mathsf{inst} \mapsto Staff\,, \mathsf{prrq} \mapsto Code^*)}$

Code	Name	Instructor	Prerequisites
CS100	Basic Mathematics for CS	R. Barns	none
MA115	Introduction to Probability	K. Jason	MA100
CS300	Formal Methods in Engineering	R. Barns	CS100, EE150
...	...	...	

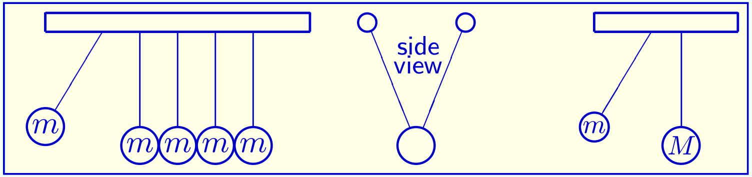
- Query operators: all the usual ones are subsumed by generic functionals
  - The usual *selection*-operator (σ) by $\boxed{\sigma\,(S, P) = S \downarrow P}$.
  - The usual *projection*-operator (π) by $\boxed{\pi\,(S, F) = \{r \rceil F \mid r : S\}}$.
  - The usual *join*-operator ($\bowtie$) by $\boxed{S \bowtie T = S \otimes T}$.

Observation: $S \bowtie T = \{s \mathbin{\dot\cup} t \mid (s, t) : S \times T \wedge s \circledcirc t\}$

Moreover, $\bowtie$ is assiciative, although $\dot\cup$ is not.

5.2 Deriving theories of programming

a. An analogy: colliding balls (”Newton’s Cradle”)



State $s := v, V$ (velocities); $`s$ before and s' after collision. Lossless collision:

$$\boxed{\begin{array}{lll} R\,(`s, s') & \equiv & m \cdot `v + M \cdot `V \ = m \cdot v' + M \cdot V' \\ & \wedge & m \cdot `v^2 + M \cdot `V^2 = m \cdot v'^2 + M \cdot V'^2 \end{array}}$$

Letting $a := M/m$, assuming $v' \neq `v$ and $V' \neq `V$ (discarding trivial case):

$$\boxed{R\,(`s, s') \ \equiv \ v' = -\tfrac{a-1}{a+1} \cdot `v + \tfrac{2 \cdot a}{a+1} \cdot `V \ \wedge \ V' = \tfrac{2}{a+1} \cdot `v + \tfrac{a-1}{a+1} \cdot `V}$$

Crucial point: mathematics is not used as just a “compact language”; rather: the calculations yield insights that are hard to obtain by intuition.

b. Program equations for a simple language (Dijkstra's guarded commands)

State change expressed by $\mathsf{R} : C \to \mathbb{S}^2 \to \mathbb{B}$, termination by $\mathsf{T} : C \to \mathbb{S} \to \mathbb{B}$.

Syntax	Behavior (program equations or equivalent program)	
Command c	State change $\mathsf{R}\,c\,(s,s')$	Termination $\mathsf{T}\,c\,s$
$v := e$	$s' = s[^v_e$	1
`skip`	$s' = s$	1
`abort`	0	0
$c' \,;\, c''$	$\exists\, t \bullet \mathsf{R}\,c'\,(s,t) \wedge \mathsf{R}\,c''\,(t,s')$	$\mathsf{T}\,c'\,s \wedge \forall\, t \bullet \mathsf{R}\,c'\,(s,t) \Rightarrow \mathsf{T}\,c''\,t$
`if` $[]\, i : I \,.\, b_i$ `->` c'_i `fi`	$\exists\, i : I \,.\, b_i \wedge \mathsf{R}\,c'_i\,(s,s')$	$\exists\, b \wedge \forall\, i : I \,.\, b_i \Rightarrow \mathsf{T}\,c'_i\,s$
`do` b `->` c' `od`	`if` $\neg b$ `-> skip` $[]$ b `->` $(c'\,;\,c)$ `fi`	

Abbreviation: $(s \bullet e) = s : \mathbb{S} \,.\, e$

c. Program theories expressed via the equations (no "special logics")

Example: ante/post semantics (Hoare style)

$\{a\}\; c\; \{p\}$	$\equiv$	$\forall\, `s \bullet \forall\, s' \bullet a[^s_{`s} \wedge \mathsf{R}\,c\,(`s, s') \Rightarrow p[^s_{s'}$	"partial correctness"
$\mathit{Term}\; c\; a$	$\equiv$	$\forall\, s \bullet a \Rightarrow \mathsf{T}\,c\,s$	"termination"
$[a]\; c\; [p]$	$\equiv$	$\{a\}\; c\; \{p\} \wedge \mathit{Term}\; c\; a$	"total correctness"

d. Calculate all properties of interest *Just predicate calculus,* ***no*** *special logics!*

Example: weakest antecondition (Dijkstra style)

$$
\begin{array}{lll}
& [a]\ c\ [p] & \\
\equiv & \langle \text{Def. } [a]\ c\ [p] \rangle & \{a\}\ c\ \{p\} \wedge \mathit{Term}\, c\, a \\
\equiv & \langle \text{Def. } \{a\}\ c\ \{p\} \rangle & \forall\, (\grave{}s \bullet \forall\, s' \bullet a[^s_{\grave{}s} \wedge \mathsf{R}\, c\, (\grave{}s, s') \Rightarrow p[^s_{s'}) \wedge \mathit{Term}\, c\, a \\
\equiv & \langle \text{Def. } \mathit{Term}\, c\, a \rangle & \forall\, (\grave{}s \bullet \forall\, s' \bullet a[^s_{\grave{}s} \wedge \mathsf{R}\, c\, (\grave{}s, s') \Rightarrow p[^s_{s'}) \wedge \forall\, (s \bullet a \Rightarrow \mathsf{T}\, c\, s) \\
\equiv & \langle \text{Repl. } \grave{}s \text{ by } s \rangle & \forall\, (s \bullet \forall\, s' \bullet a \wedge \mathsf{R}\, c\, (s, s') \Rightarrow p[^s_{s'}) \wedge \forall\, (s \bullet a \Rightarrow \mathsf{T}\, c\, s) \\
\equiv & \langle \text{Distr. } \forall/\wedge \rangle & \forall\, s \bullet \forall\, (s' \bullet a \wedge \mathsf{R}\, c\, (s, s') \Rightarrow p[^s_{s'}) \wedge (a \Rightarrow \mathsf{T}\, c\, s) \\
\equiv & \langle \text{Shunt } \wedge/\Rightarrow \rangle & \forall\, s \bullet \forall\, (s' \bullet a \Rightarrow \mathsf{R}\, c\, (s, s') \Rightarrow p[^s_{s'}) \wedge (a \Rightarrow \mathsf{T}\, c\, s) \\
\equiv & \langle \text{Ldist. } \Rightarrow/\forall \rangle & \forall\, s \bullet (a \Rightarrow \forall\, s' \bullet \mathsf{R}\, c\, (s, s') \Rightarrow p[^s_{s'}) \wedge (a \Rightarrow \mathsf{T}\, c\, s) \\
\equiv & \langle \text{Ldist. } \Rightarrow/\wedge \rangle & \forall\, s \bullet a \Rightarrow \forall\, (s' \bullet \mathsf{R}\, c\, (s, s') \Rightarrow p[^s_{s'}) \wedge \mathsf{T}\, c\, s
\end{array}
$$

So $[a]\ c\ [p] \ \equiv\ \forall\, s \bullet a \Rightarrow \forall\, (s' \bullet \mathsf{R}\, c\, (s, s') \Rightarrow p[^s_{s'}) \wedge \mathsf{T}\, c\, s$.

So $[a]\; c\; [p] \;\equiv\; \forall\, s \bullet a \Rightarrow \forall\,(s' \bullet \mathsf{R}\, c\,(s, s') \Rightarrow p[^s_{s'}) \wedge \mathsf{T}\, c\, s$.

Hence we define

$$
\boxed{
\begin{array}{rcll}
\mathsf{wla}\, c\, p & \equiv & \forall\, s' \bullet \mathsf{R}\, c\,(s, s') \Rightarrow p[^s_{s'} & \text{“weakest liberal antecondition”} \\
\mathsf{wa}\, c\, p & \equiv & \mathsf{wla}\, c\, p \wedge \mathsf{T}\, c\, s & \text{“weakest antecondition”}
\end{array}
}
$$

From this, we obtain by calculation (functional predicate calculus)

$$
\boxed{
\begin{array}{rcl}
\mathsf{wa}\, [\![v := e]\!]\, p & \equiv & p[^v_e \\
\mathsf{wa}\, [\![c' \,;\, c'']\!]\, p & \equiv & \mathsf{wa}\, c'\, (\mathsf{wa}\, c''\, p) \\
\mathsf{wa}\, [\![\texttt{if}\ \ [\!]\, i : I \,.\, b_i \;\texttt{->}\; c'_i\ \ \texttt{fi}]\!]\, p & \equiv & \exists\, b \wedge \forall\, i : I \,.\, b_i \Rightarrow \mathsf{wa}\, c'_i\, p \\
\mathsf{wa}\, [\![\texttt{do}\ b \;\texttt{->}\; c'\ \texttt{od}]\!]\, p & \equiv & \exists\, n : \mathbb{N} \,.\, w^n\,(\neg\, b \wedge p)\ \text{ defining } w \text{ by} \\
w\, q & \equiv & (\neg\, b \wedge p) \vee (b \wedge \mathsf{wa}\, c'\, q)
\end{array}
}
$$

The formula for loops is of theoretical interest only.

Development of invariants and bound functions is outlined in the notes.

6 Examples III: Common aspects

Example: Automata as systems

- Motivation and chosen topic

  Automata: classical common ground between computing and systems theory.

  Even here formalization yields unification and new insights.

  Topic: sequentiality and the derivation of properties by predicate calculus.

- Sequences Let $\boxed{\square\, n = \{m : \mathbb{N} \mid m < n\}}$ for $n : \mathbb{N}'$ where $\mathbb{N}' = \mathbb{N} \cup \iota\, \infty$.

  A *sequence* is any function whose domain is $\square\, n$ for some $n : \mathbb{N}'$

  - Operators Concatenation ($+\!\!+$), e.g., $(0, 7, e) +\!\!+ (3, d) = 0, 7, e, 3, d$.
    Append ($-\!\!<$): $x -\!\!< a = x +\!\!+ \tau\, a$. Length ($\#$): $\#\, x = n \equiv \mathcal{D}\, x = \square\, n$
  - List types For set A, define A^n by $A^n = \square\, n \to A$, e.g., $(0, 1, 1, 0) \in \mathbb{B}^4$.
    Also, $A^* = \bigcup n : \mathbb{N} \,.\, A^n$ (lists).

- Discrete systems: signals of type A^* (or B^*), and systems of type $A^* \to B^*$.

- Sequentiality Define $\leq$ on A^* (or B^* etc.) by $x \leq y \equiv \exists\, z : A^* \,.\, y = x +\!\!+\, z$.

  System s is *non-anticipatory* or $\boxed{\textit{sequential} \text{ iff } x \leq y \Rightarrow s\,x \leq s\,y}$

  Function $r : (A^*)^2 \to B^*$ is a $\boxed{\textit{residual behavior} \text{ of } s \text{ iff } s\,(x +\!\!+\, y) = s\,x +\!\!+\, r\,(x, y)}$

$\boxed{\text{THEOREM: } s \text{ is sequential iff it has a residual behavior function.}}$

Proof: we start from the sequentiality side.

$$
\boxed{
\begin{array}{ll}
\multicolumn{2}{l}{\forall\,(x,y) : (A^*)^2 \,.\, x \leq y \Rightarrow s\,x \leq s\,y} \\
\equiv \ \langle \text{Definit. } \leq \rangle & \forall\,(x,y) : (A^*)^2 \,.\, \exists\,(z : A^* \,.\, y = x +\!\!+\, z) \Rightarrow \exists\,(u : B^* \,.\, s\,y = s\,x +\!\!+\, u) \\
\equiv \ \langle \text{Rdst } \Rightarrow/\exists \rangle & \forall\,(x,y) : (A^*)^2 \,.\, \forall\,(z : A^* \,.\, y = x +\!\!+\, z \Rightarrow \exists\, u : B^* \,.\, s\,y = s\,x +\!\!+\, u) \\
\equiv \ \langle \text{Nest, swp} \rangle & \forall\, x : A^* \,.\, \forall\, z : A^* \,.\, \forall\,(y : A^* \,.\, y = x +\!\!+\, z \Rightarrow \exists\, u : B^* \,.\, s\,y = s\,x +\!\!+\, u) \\
\equiv \ \langle \text{1-pt, nest} \rangle & \forall\,(x,z) : (A^*)^2 \,.\, \exists\, u : B^* \,.\, s\,(x +\!\!+\, z) = s\,x +\!\!+\, u \\
\equiv \ \langle \text{Compreh.} \rangle & \exists\, r : (A^*)^2 \to B^* \,.\, \forall\,(x,z) : (A^*)^2 \,.\, s\,(x +\!\!+\, z) = s\,x +\!\!+\, r\,(x,z)
\end{array}
}
$$

We used the *function comprehension* axiom: for any relation $R : X \times Y \to \mathbb{B}$,

$$\boxed{\forall\,(x : X \,.\, \exists\, y : Y \,.\, R\,(x,y)) \equiv \exists\, f : X \to Y \,.\, \forall\, x : X \,.\, R\,(x, f\,x)}$$

- Derivatives and primitives The preceding framework leads to the following.
  - Observation: An rb function is unique (exercise).
  - We define the *derivation* operator D on sequential systems by

$$\boxed{D\, s\, \varepsilon = \varepsilon \quad \text{and} \quad \mathsf{D}\, s\, (x -\!\!< a) = s\, x +\!\!\!+ \mathsf{D}\, s\, (x -\!\!< a)}$$

    With the rb function r of s, $\mathsf{D}\, s\, (x -\!\!< a) = r\, (x, \tau\, a)$.
  - *Primitivation* I is defined for any $g : A^* \to B^*$ by

$$\boxed{\mathsf{I}\, g\, \varepsilon = \varepsilon \quad \text{and} \quad \mathsf{I}\, g\, (x -\!\!< a) = \mathsf{I}\, g\, x +\!\!\!+ g\, (x +\!\!\!+ a)}$$

  - Properties (note a striking analogy from analysis)

$$\boxed{\begin{array}{l|l} s\, (x -\!\!< a) = s\, x +\!\!\!+ \mathsf{D}\, s\, (x -\!\!< a) & s\, x = s\, \varepsilon +\!\!\!+ \mathsf{I}\, (\mathsf{D}\, s)\, x \\ f\, (x + h) = f\, x + \mathsf{D}\, f\, x \cdot h & f\, x = f\, 0 + \mathsf{I}\, (\mathsf{D}\, f)\, x \end{array}}$$

    In the second row, D is derivation as in analysis, and $\mathsf{I}\, g\, x = \int_0^x g\, y \cdot \mathsf{d}\, y$.
  - The $\boxed{\textit{state space} \text{ is } \{y : A^* \,.\, r\, (x, y) \mid x : A^*\}}$.

7 Final considerations

- What we have shown
  - A formalism with a very simple language and powerful formal rules
  - Notational and methodological unification of CS and other engineering theories
  - Unification also encompassing a large part of mathematics.
- Ramifications
  - Scientific: obvious
  - Educational: unified basis for ECE (Electrical and Computer Engineering)
- Possible curriculum structure
  - Formal calculation at early stage
  - Other engineering math courses rely on it and provide consolidation
- Possible impediments: student mathphobia and required effort of lecturers