Declarative specification and calculation in view of software evolution[0]

Raymond Boute INTEC — Ghent University

Overview

0. **Motivation**
1. **Functionals for transformation** (induction: collecting the tools)
2. **Making the functionals generic** (design: generalizing the tools)
3. **Functional predicate calculus** (deduction: applying the new tools)
4. **Conclusion**

[0]Prepared for the 2002/01/18 meeting of the FWO Research Network "Foundations of Software Evolution".

0 Motivation

- **Software evolution**

  a. Software systems

  b. Design media (languages, software engineering tools)

  c. Intellectual means (paradigms, models)

  d. ⋯

- **Evolution-insensitive methods and means**

  – Support for **(abstract) specification** and **symbolic reasoning** for a, b, c, d ⋯

  – Program-like formalisms too **detailed, implementation-oriented** and **unsuitable for human discourse and reasoning**

  – Declarative formalisms meet the needs, provided they meet their own ideals

    * Functional and logic programming formalisms are still algorithmic

  Only "genuine" declarativity can meet the objectives

    * Mathematics as the proven formalism in other branches of engineering

- **Observation** (Reynolds):

  > In designing a programming language, the central problem is to organize a variety of concepts in a way which exhibits uniformity and generality. Substantial leverage can be gained in attacking this problem if the concepts can be defined concisely within a framework which has already proven its ability to impose uniformity and generality upon a wide variety of mathematics

- **Problem with the existing formalisms of mathematics**
  - Intended for informal and semi-formal (human) discourse
  - Heterogeneous mixture of very well-designed parts and very ad hoc designs
    - Well-designed parts in algebra and analysis (due to Descartes and Leibniz)
    - Ad hoc designs in discrete mathematics, logic and computer science (with exceptions)
  - In software engineering, the highest standards of formality and precision are imposed by the discrete nature of the subject

1 Functionals for transformation

1.0 Functional Mathematics

- **Principle:** (re)defining mathematical objects, whenever feasible, as functions.
- **Advantages**
  - Conceptual: uniformity in treatment while respecting (only) essential differences
  - Practical: sharing general-purpose operators over functions
- **Example:** sequences (tuples, lists, ...)
  - Motivation for this choice: "interface" between discrete and continuous mathematics
  - Wide ramifications:
    * Removal of all conventions having poor calculational properties
      Worst kind of violation: against Leibniz's principle
      (equals replaceable by equals, no exceptions)
      Example: ellipsis $a_0 + a_1 + \cdots + a_7$
      Letting $a_i = i^2$ yields $0 + 1 + \cdots + 49$
    * Replacement by well-defined operators and algebraic calculation rules
  - Importance: mathematical software, OpenMath etc.

- **Sequences as functions**
  - Principle: $(a, b, c)\,0 = a$ and $(a, b, c)\,1 = b$ and $(a, b, c)\,2 = c$
  - Intuitively trivial, yet:
    - in the literature handled often as entirely or subtly distinct from functions,
    - in the few exceptions, functional properties left unexploited.
  - Examples of functional properties of sequences
    - Inverses: $(a, b, c, d)^{-}\,c = 2$
    - Composition: $(0, 3, 5, 7) \circ (2, 3, 1) = 5, 7, 3$ and $f \circ (x, y) = f\,x, f\,y$
    - Transposition: $(f, g)^T x = f\,x, g\,x$

    Seemingly secondary, but very useful once discovered
  - Not obtainable by the usual formal treatments of lists, e.g.,
    - recursive definition: $[]$ is a list and, if x is a list, so is $\mathit{cons}\,a\,x$
    - index function separate: $\mathit{ind}\,(\mathit{cons}\,a\,x)\,0 = a$ and $\mathit{ind}\,(\mathit{cons}\,a\,x)\,(n+1) = \mathit{ind}\,x\,n$
      e.g., in Haskell: `ind [a:x] 0 = a` and `ind [a:x] (n + 1) = x n`

- **Function(al)s for sequences**
  - Domain specification: "block" $\Box$

    $\Box\, n = \{k : \mathbb{N} \mid k < n\}$ for $n : \mathbb{N}$ or $n := \infty$
  - Length: $\#$

    $\#\, x = n \,\equiv\, \mathcal{D}\, x = \Box\, n$, equivalently: $\mathcal{D}\, x = \Box\, (\#\, x)$, even $\#\, x = \Box^{-}\, (\mathcal{D}\, x)$
  - Prefix: $\succ\!\!-$

    $\#\, (a \succ\!\!- x) = \#\, x + 1$ and
    $i \in \mathcal{D}\, (a \succ\!\!- x) \Rightarrow (i = 0)\, ?\, a \dagger x\, (i - 1)$

    Observe use of the conditional: $c\, ?\, b \dagger a = (a, b)c$.
  - Shift: σ (for nonempty x)

    $\#\, (\sigma\, x) = \#\, x - 1$ and
    $i \in \mathcal{D}\, (\sigma\, x) \Rightarrow \sigma\, x\, i = x\, (i + 1)$
  - The usual induction principle is a *theorem* (not an axiom)

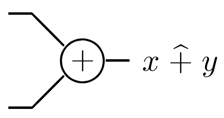
    $\forall\, (x : A^* \,.\, P\, x) \,\equiv\, P\, \varepsilon \wedge \forall\, (x : A^* \,.\, P\, x \Rightarrow \forall\, a : A \,.\, P\, (a \succ\!\!- x))$

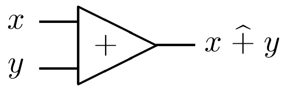
1.1 Towards point-free formulations

- **Signal flow systems** are assemblies of interconnected components whose dynamical behavior is modelled by functionals mapping input signals to output signals.

- **Basic building blocks**

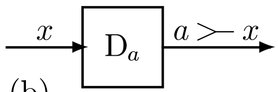
  - Memoryless devices realizing arithmetic operations
    * Sum (product, ...) of two signals x and y modelled as $(x \mathbin{\widehat{+}} y)\,t = x\,t + y\,t$
    * Explicit *direct extension* operator $\widehat{\ }$ (in engineering often left implicit)

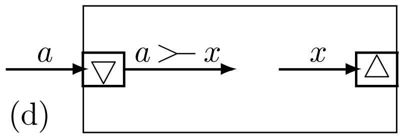
(a) 

(c) 

  - Memory devices: latches (discrete case), integrators (continuous case)

$$\mathrm{D}_a\, x\, n = (n = 0)\, ?\, a \nmid x\,(n-1) \text{ or, without the time variable, } \mathrm{D}_a\, x = a \mathrel{>\!\!-} x$$



(b)


(d)

- **Time is not structural,** hence transformational design = eliminating the time variable

1.2 A transformation example

- **From specification to realization**
  - Recursive specification, given: set A and $a:A$ and $g:A\to A$

$$\textbf{def}\ f:\mathbb{N}\to A\ \textbf{with}\ f\,n = (n=0)\,?\,a \nmid g\,(f\,(n-1)) \qquad (0)$$

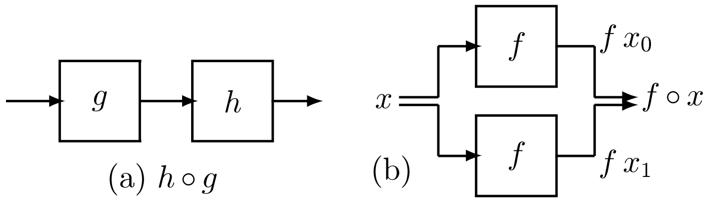
  - Calculational transformation

$$\begin{aligned} f\,n =&\ \langle\text{Def. } f\rangle && (n=0)\,?\,a \nmid g\,(f\,(n-1)) \\ =&\ \langle\text{Def. } \circ\rangle && (n=0)\,?\,a \nmid (g\circ f)\,(n-1) \\ =&\ \langle\text{Def. D}\rangle && \mathrm{D}_a\,(g\circ f)\,n \\ =&\ \langle\text{Def. } \overline{}\rangle && \mathrm{D}_a\,(\overline{g}\ f)\,n \\ =&\ \langle\text{Def. } \circ\rangle && (\mathrm{D}_a\circ\overline{g}\,)\,f\,n, \end{aligned} \qquad (1)$$

    hence $f = (\mathrm{D}_a\circ\overline{g}\,)\,f$ by function extensionality.

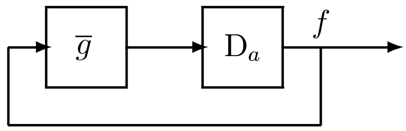
- **Functionals introduced** (ignoring types for the time being)
  - Function composition: $\circ$, defined by $(f\circ g)\,x = f\,(g\,x)$
  - Direct extension (1 argument): $\overline{}$, defined by $\overline{g}\ x = g\circ x$

- **Structural interpretations**
  - Note: the time variable is gone in $f = (\mathrm{D}_a \circ \overline{g}\,)\, f$
  - Structural interpretations of composition: (a) cascading; (b) replication

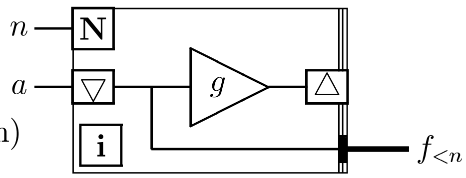


    Property: $\overline{h \circ g} = \overline{h} \circ \overline{g}$ (proof: exercise)
  - Immediate structural solution for the fixpoint equation $f = (\mathrm{D}_a \circ \overline{g}\,)\, f$

(Block diagram)



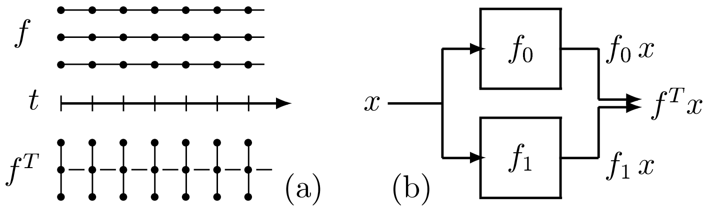
(LabVIEW diagram)



- **A third operator: transposition** (already seen: composition, direct extension)
  - Purpose: swapping the arguments of a higher-order function

$$f^T y\, x = f\, x\, y$$

  - Nomenclature borrowed from matrix theory
  - Structural interpretations:
    (a) from a family of signals to a tuple-valued signal,
    (b) signal fanout



  - Subsumes the `zip` operator from functional programming

```
zip[[a,b,c],[a',b',c']] = [[a,a'],[b,b'],[c,c']]
```

    assuming lists taken as functions.

- **Calculating with transposition, composition and direct extension**
  - Duality between composition and transposition: provided x is not free in M,

$$M \circ (\lambda x.N) = \lambda x.MN \quad \text{and} \quad (\lambda x.N)^T M = \lambda x.NM.$$

  - Generalizing direct extension to an arbitrary number of arguments:

$$\begin{aligned}(f \mathbin{\widehat{\star}} f')\, x &= f\, x \star f'\, x \\ &= (\star)\, (f\, x, f'\, x) \\ &= (\star)\, ((f, f')^T x) \\ &= ((\star) \circ (f, f'))^T x\end{aligned}$$

  (hints added orally) hence $f \mathbin{\widehat{\star}} f' = (\star) \circ (f, f')^T$.
  We define the generalized direct extension operator $\stackrel{<}{—}$ by

$$\stackrel{<}{g}\, h = g \circ h^T \tag{2}$$

  for any function g whose argument is a function and any family h of functions.

2 Making the functionals generic

2.0 Conventions for functions

- **Function** = *domain* ($\mathcal{D}\, f$) and *mapping* (unique $f\, x$ for every x in $\mathcal{D}\, f$).
- **Function equality** = equality of the domains and the mappings
  - Leibniz's principle:
$$f = g \Rightarrow \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, f \cap \mathcal{D}\, g \Rightarrow f\, x = g\, x) \tag{3}$$
  - function extensionality: using a fresh dummy x,
$$\frac{q \Rightarrow \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, f \cap \mathcal{D}\, g \Rightarrow f\, x = g\, x)}{q \Rightarrow f = g}. \tag{4}$$
- **Style of definition** (awaiting quantifiers)
  - a *domain axiom* of the form $x \in \mathcal{D}\, f \;\equiv\; x \in X \wedge p_x$
  - a *mapping axiom* of the form $x \in \mathcal{D}\, f \Rightarrow q_{f,x}$

(x a variable, X a set, p_x and $q_{f,x}$ propositions, subscripts specify free occurrences).
Example: the *constant function specifier* $^\bullet$: for any set X and any e,
$$\mathcal{D}\, (X^\bullet\, e) = X \quad \text{and} \quad x \in X \Rightarrow (X^\bullet\, e)\, x = e. \tag{5}$$

- **Denoting functions by abstractions**
  - Principle: recall the style of definition
    - a *domain axiom* of the form $x \in \mathcal{D}\, f \ \equiv\ x \in X \wedge p_x$
    - a *mapping axiom* of the form $x \in \mathcal{D}\, f \Rightarrow q_{f,x}$

    If $q_{f,x}$ has the explicit form $f\, x = e_x$,
    then we denote the function by $x : X \wedge p\, .\, e$ ($\wedge p$ optional)
  - Axioms (a typed lambda calculus)

$$\begin{aligned} d \in \mathcal{D}\,(x : X \wedge p\, .\, e) &\ \equiv\ d \in X \wedge p^x_d \\ d \in \mathcal{D}\,(x : X \wedge p\, .\, e) &\ \Rightarrow\ (x : X \wedge p\, .\, e)\, d = e^x_d \end{aligned} \qquad (6)$$

  - Examples
    - $X^{\bullet}\, e = x : X\, .\, e$ (choosing x not free in e)
    - $n : \mathbb{Z}\, .\, 2 \cdot n$ doubles every natural number

2.1 Design criteria and method for generic functionals

- **Reason for making fuctionals generic:**
  in functional mathematics, they become shared by many more kinds of objects than usual.
- **Shortcomings of traditional operators:** the restrictions on the arguments, e.g.,
  - $f \circ g$ requires $\mathcal{R}\, g \subseteq \mathcal{D}\, f$, in which case $\mathcal{D}\,(f \circ g) = \mathcal{D}\, g$
  - f^- requires f injective, in which case $\mathcal{D}\, f^- = \mathcal{R}\, f$
- **Approach used here;**
  - No restrictions on the argument function(s)
  - Refine domain of the result function
  - Conservative, i.e., if the traditional restriction is satisfied, the generalization yields the "old" case

2.2 Some important generic functionals

- **Filtering** ($\downarrow$) generalizes $f = x : \mathcal{D} f \,.\, f\,x$ as follows: for any function f and predicate P,

$$f \downarrow P \quad = \quad x : \mathcal{D} f \cap \mathcal{D} P \wedge P\,x \,.\, f\,x \tag{7}$$

  Shorthand: f_P for $f \downarrow P$. Example: $f_{<n}$.

  Also defined for sets: $x \in S_P \equiv x \in S \wedge P\,x$, yielding convenient abbreviations like $\mathbb{R}_{\geq 0}$.

- **Composition** ($\circ$) generalizes traditional composition: for *any* functions f and g,

$$\begin{aligned} x \in \mathcal{D}(f \circ g) \quad &\equiv \quad x \in \mathcal{D}\,g \wedge g\,x \in \mathcal{D}\,f \\ x \in \mathcal{D}(f \circ g) \quad &\Rightarrow \quad (f \circ g)\,x = f\,(g\,x). \end{aligned} \tag{8}$$

  Conservational: if the traditional requirement $\mathcal{R}\,g \subseteq \mathcal{D}\,f$ is satisfied, then $\mathcal{D}(f \circ g) = \mathcal{D}\,g$.

  Illustrations

  - Since sequences are functions,
    $(0,3,5,7) \circ (2,3,1) = 5,7,3$ and $(0,3,5,7) \circ (2,3,5) = 5,7$,
    but also $(0,3,5,7) \circ (5,3,1) = (7,3) \circ (-1)$ (not a sequence).
  - Similarly, since $f \circ (x,y) = f\,x, f\,y$ (x and y in $\mathcal{D}\,f$), $\circ$ subsumes the *map* operator from functional programming, viz., `f @ [x, y] = [f x, f y]`.

- **Direct extension** ($\widehat{\ }$)
  - Principle: for any (infix) operator $\star$ and any functions f and g, we let the domain of $f \mathbin{\widehat{\star}} g$ contain exactly those values x for which the expression $f\,x \star g\,x$ does not contain any out-of-domain applications
  - Resulting definition:

$$\begin{array}{rcl} x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) & \equiv & x \in \mathcal{D}\,f \cap \mathcal{D}\,g \wedge (f\,x, g\,x) \in \mathcal{D}\,(\star) \\ x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) & \Rightarrow & (f \mathbin{\widehat{\star}} g)\,x = f\,x \star g\,x. \end{array} \tag{9}$$

- **Transposition** ($—^T$) Recall the definition ignoring types: $f^T y\,x = f\,x\,y$
  - Simplest argument type: $A \to (B \to C)$ (given sets A, B, C).
    The image f^T of $f : A \to (B \to C)$ has type $B \to (A \to C)$ and property $(f^T)^T = f$.
    Note: one usually writes $A \to B \to C$ for $A \to (B \to C)$.
  - We want the argument of ^{T} to be *any* function family.
    * Liberal design: $\mathcal{D}\,f^T = \bigcup\,x : \mathcal{D}\,f\,.\,\mathcal{D}\,(f\,x)$ or, in point-free style, $\mathcal{D}\,f^T = \bigcup\,(\mathcal{D} \circ f)$ (not elaborated here)
    * Preferred design is with *intersection* in view of $\overset{<}{g}\,h = g \circ h^T$ to generalize (9)

$$\begin{array}{rcl} \mathcal{D}\,f^T & = & \bigcap\,(x : \mathcal{D}\,f\,.\,\mathcal{D}\,(f\,x)) \\ y \in \mathcal{D}\,f^T & \Rightarrow & x \in \mathcal{D}\,f \Rightarrow f^T y\,x = f\,x\,y \end{array} \tag{10}$$

      or, in compact form, $f^T = y : \bigcap\,(\mathcal{D} \circ f)\,.\,x : \mathcal{D}\,f\,.\,f\,x\,y$.

3 Functional predicate calculus

3.0 Axioms

- **Predicates** are a boolean-valued functions.

  Choice ***false***/ ***true*** versus 0 / 1 secondary here but, in a wider context, 0 / 1 is advantageous.

- **Quantifiers** $\forall$ and $\exists$ are predicates over predicates.
  - Informally:
    * $\forall P$ means that P is the constant 1-valued predicate
    * $\exists P$ means that P is *not* the constant 0-valued predicate.
  - Formal axioms:

  $$\forall P \equiv (P = \mathcal{D}\, P^{\bullet} 1) \text{ and } \exists P \equiv (P \neq \mathcal{D}\, P^{\bullet} 0). \tag{11}$$

    The axioms are conceptually indeed as simple as they seem,
    but they create a rich algebraic structure (dozens of useful calculation rules)
  - Observation: $\forall$ and $\exists$ are typical *elastic operators*.

3.1 Intermezzo: elastic operators and ramifications

- **Principle:** functionals replacing the various kinds of common ad hoc abstractors, e.g.,

$$\forall\, x : X \qquad \textstyle\sum_{i=m}^{n} \qquad \lim_{x \to a} .$$

  Together with function abstraction (6) they yield readily recognizable expressions, e.g.,

$$\forall\, x : X \,.\, P\, x \quad \textstyle\sum\, i : m \,..\, n \,.\, x_i \quad \mathsf{lim}\ (x : \mathbb{R} \,.\, f\, x)\, a$$

  or, for less casual readers, point-free forms such as

$$\forall\, P \quad \textstyle\sum\, x \quad \mathsf{lim}\ f\, a$$

  Example: $\forall\, x : \mathbb{R} \,.\, x^2 \geq 0$ obtains familiar form and meaning,
  but also a novel decomposition: $\forall$ and $x : \mathbb{R} \,.\, x^2 \geq 0$, are both *functions*.

- **General importance:** Same functionals for point-free and point-wise expressions.

- **For predicate calculus:**
  - A calculus of *functions* (familiar to working mathematicians and engineers)
  - Algebraic flavor, laws more calculation-friendly

3.2 Derived calculation rules

- **First batch**: simple rules derived directly from axioms (11) and function equality (3,4).
  - $\forall (X^{\bullet} 1) \equiv 1$ and $\exists (X^{\bullet} 0) \equiv 0$
  - $\forall \varepsilon \equiv 1$ and $\exists \varepsilon \equiv 0$ (ε is the *empty* function or predicate with $\mathcal{D}\,\varepsilon = \emptyset$)
  - For any non-constant P: $\forall P \equiv 0$ and $\exists P \equiv 1$

  Theorems illustrative of the algebraic equational style:

  - *Duality*: $\forall (\overline{\neg}\, P) \equiv (\overline{\neg}\, \exists)\, P$
  - *Meeting*: $\forall P \wedge \forall Q \Rightarrow \forall (P \mathbin{\widehat{\wedge}} Q)$.
    Conditional converse: $\mathcal{D}\,P = \mathcal{D}\,Q \Rightarrow \forall (P \mathbin{\widehat{\wedge}} Q) \Rightarrow \forall P \wedge \forall Q$.

  Typical calculational proof for duality

$$
\begin{array}{rcll}
\forall (\overline{\neg}\, P) & \equiv & \langle \text{Def. } \forall \text{ (11)}, \mathcal{D}(\overline{\neg}\, P) = \mathcal{D}\,P \rangle & \overline{\neg}\, P = \mathcal{D}\,P^{\bullet} 1 \\
 & \equiv & \langle \overline{\neg}\, P = Q \equiv P = \overline{\neg}\, Q \rangle & P = \overline{\neg}\, (\mathcal{D}\,P^{\bullet} 1) \\
 & \equiv & \langle e \in \mathcal{D}\,g \Rightarrow \overline{g}\,(X^{\bullet} e) = X^{\bullet} (g\,e) \rangle & P = \mathcal{D}\,P^{\bullet} (\neg 1) \\
 & \equiv & \langle \neg 1 = 0, \text{ def. } \exists \text{ (11)} \rangle & \neg\, (\exists P) \\
 & \equiv & \langle x \in \mathcal{D}\,(\overline{g}\, f) \Rightarrow \overline{g}\, f\, x = g\,(f\,x) \rangle & \overline{\neg}\, \exists\, P
\end{array}
$$

  Justifications are given between $\langle\rangle$ (Feyen's convention).
  All are properties of generic functionals (exercises).

- **First batch (continued)**
  - Properties of constant predicates revealing the role of types (uncommon in logic textbooks)

    $$\forall\,(X^{\bullet}\,0) \;\equiv\; X = \emptyset \quad \text{and} \quad \exists\,(X^{\bullet}\,1) \;\equiv\; X \neq \emptyset$$

    Combined with the earlier properties,

    $$\forall\,(X^{\bullet}\,x) \;\equiv\; x \vee X = \emptyset \quad \text{and} \quad \exists\,(X^{\bullet}\,x) \;\equiv\; x \wedge X \neq \emptyset$$

  - Fast technique for laws of this kind: *case analysis* (a) and *Shannon expansion* (b, c)
    a. $\forall\,P[^v_0] \wedge \forall\,P[^v_1] \Rightarrow \forall\,P$
    b. $\forall\,P \;\equiv\; (v \wedge \forall\,P[^v_1]) \vee (\neg\,v \wedge \forall\,P[^v_0])$
    c. $\forall\,P \;\equiv\; (v \Rightarrow \forall\,P[^v_1]) \wedge (\neg\,v \Rightarrow \forall\,P[^v_0])$

    assuming v is a boolean variable in P. Similarly for $\exists$.
  - Important consequenses are *semidistributivity rules*:
    * $\forall\,(x \overrightharpoon{\wedge} P) \;\equiv\; (x \wedge \forall\,P) \vee \mathcal{D}\,P = \emptyset$
    * $\forall\,(x \overrightharpoon{\Rightarrow} P) \;\equiv\; x \Rightarrow \forall\,P$
    * $\forall\,(P \overleftharpoon{\Rightarrow} x) \;\equiv\; \exists\,P \Rightarrow x$

    where $\overrightharpoon{\;\;}$ is the (right) half direct extension operator

    $$x \overrightharpoon{\star} f \;\;=\;\; (\mathcal{D}\,f^{\bullet}\,x)\,\widehat{\star}\,f \tag{12}$$

- **Second batch:** metatheorems whose counterparts are axioms in logical textbooks. Here they are again *consequences* of the axioms (11) and function equality (3,4).
  - Instantiation: $\forall P \Rightarrow x \in \mathcal{D}\,P \Rightarrow P\,x$
  - Generalization: $q \Rightarrow x \in \mathcal{D}\,P \;\Rightarrow P\,x \;\vdash\; q \Rightarrow \forall P$

  Importance:

  - Basis for proving all properties usually appearing in logic textbooks
  - Additional important rules for practical applications,e.g.,, *trading*

$$\forall P_R \equiv \forall (R \mathrel{\widehat{\Rightarrow}} P) \quad \text{and} \quad \exists P_R \equiv \exists (R \mathrel{\widehat{\wedge}} P) \tag{13}$$

- **Third batch:** shows correspondence between point-free and conventional formulas. Convention: P, Q be *predicates*, $R : X \to Y \to \mathbb{B}$ for some X and Y.

Empty rule	$\forall\, \varepsilon = 1$
1-point rule	$\forall\,(x \mapsto y) = y$
Merge rule	$P \copyright Q \Rightarrow \forall\,(P \cup Q) = \forall\, P \wedge \forall\, Q$
Distribution	$\mathcal{D}\, P = \mathcal{D}\, Q \Rightarrow \forall\,(P \mathbin{\widehat{\wedge}} Q) = \forall\, P \wedge \forall\, Q$
Transposition	$\forall\,(\forall \circ R) = \forall\,(\forall \circ R^T)$
Composition	$\forall\, P \equiv \forall\,(P \circ f)$ provided $\mathcal{D}\, P \subseteq \mathcal{R}\, f$
Trading	$\forall\,(P \downarrow Q) \equiv \forall\,(Q \mathbin{\widehat{\Rightarrow}} P)$

Replacing predicates by abstractions with boolean *expressions*, under proper conditions:

Empty rule	$\forall\,(x : \emptyset \,.\, p) = 1$
1-point rule	$\forall\,(x : X \,.\, x = y \Rightarrow p) \equiv y \in X \wedge p^x_y$
Domain split	$\forall\,(x : X \cup Y \,.\, p)$
(if compat.)	$\equiv \forall\,(x : X \,.\, p) \wedge \forall\,(x : Y \,.\, p)$
Distribution	$\forall\,(x : X \,.\, p \wedge q)$
	$\equiv \forall\,(x : X \,.\, p) \wedge \forall\,(x : X \,.\, q)$
Dummy swap	$\forall\,(x : X \,.\, \forall\, y : Y \,.\, p)$
	$\equiv \forall\,(y : Y \,.\, \forall\, x : X \,.\, p)$
Dummy chng	$\forall\,(x : X \,.\, p) \equiv \forall\,(y : Y \,.\, p^x_{f\,y})$
Trading	$\forall\,(x : X \wedge p \,.\, q) \equiv \forall\,(x : X \,.\, p \Rightarrow q)$

3.3 Example: refined function typing

Predicate calculus applicable in pure and applied mathematics, esp. software engineering.
Here: only one example, wrapping up a few issues about the function range.

- **Function range** ($\mathcal{R}$): for any function F and any y,

$$y \in \mathcal{R}\, f \quad \equiv \quad \exists\, x : \mathcal{D}\, f \,.\, y = f\, x \tag{14}$$

- **Alternative symbol (same axiom):** { }
  - Motivation: expressions like $\{a, b, c\}$ and $\{n : \mathbb{Z} \,.\, 2 \cdot n\}$ have their usual meaning.
  - Abstraction vatiant: $x : X \mid p$ stands for $x : X \wedge p \,.\, x$, as in $\Box\, n = \{k : \mathbb{N} \mid k < n\}$.
  - Useful derived rule: $y \in \{x : X \mid p\} \equiv y \in X \wedge p^x_y$ (most often used rule in practice)
  - We do not use { } as a singleton set operator (ι instead)

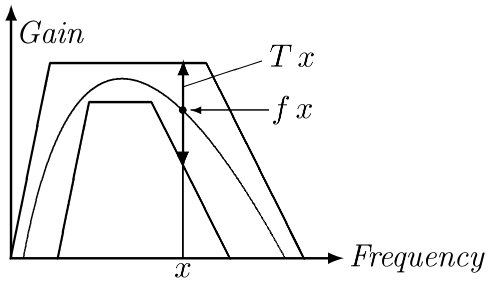
- **Illustration: the function inverse** We define f^- for *any* f (not only injective f)
  - Principle: let $\mathcal{D}\,f^-$ contain just the points corresponding to unique elements in $\mathcal{D}\,f$
  - Formalization: *bijectivity domain* and the *bijectivity range*:

$$\begin{aligned} \mathsf{Bdom}\,f &= \{x : \mathcal{D}\,f \mid \forall\, x' : \mathcal{D}\,f\,.\,f\,x = f\,x' \Rightarrow x = x'\} \\ \mathsf{Bran}\,f &= \{x : \mathsf{Bdom}\,f\,.\,f\,x\}. \end{aligned} \tag{15}$$

  - Generic function inverse functional —$^-$, defined for any function f by

$$\mathcal{D}\,f^- = \mathsf{Bran}\,f \;\wedge\; \forall\, x : \mathsf{Bdom}\,f\,.\,f^-\,(f\,x) = x. \tag{16}$$

- **The function approximation paradigm for range refinement**
  - Purpose: formalizing *tolerances* for *functions*
  - Principle: a *tolerance function* T specifiying, for every domain value x the set $T\,x$ of allowable values. Important: the domain of T serves as the domain specification
    Formalized: a function f *meets* tolerance T iff
    $$\mathcal{D}\,f = \mathcal{D}\,T \quad \wedge \quad x \in \mathcal{D}\,f \cap \mathcal{D}\,T \Rightarrow f\,x \in T\,x.$$
    Pictorial representation (example: radio frequency filter characteristic).



  - *Generalized Functional Cartesian Product* $\times$: for *any* family T of sets,
    $$f \in \times T \quad \equiv \quad \mathcal{D}\,f = \mathcal{D}\,T \wedge \forall\, x : \mathcal{D}\,f \cap \mathcal{D}\,T\,.\,f\,x \in T\,x. \tag{17}$$
    Properties: (a) If $\times T \neq \emptyset$, then $\times^{-}(\times T) = T$
    (b) with function *equality* ($f = g \equiv \mathcal{D}\,f = \mathcal{D}\,g \wedge \forall\, x : \mathcal{D}\,f \cap \mathcal{D}\,g\,.\,f\,x = g\,x$), we obtain $f = g \equiv f \in \times(\iota \circ g)$ (exact approximation).

- **Applications in the discrete mathematics**
  * Expressing the common Cartesian product: with $T := A, B$ (a pair of sets),

$$\bigtimes (A, B) = A \times B$$

    assuming the common Cartesian product is defined (for pairs as fuctions) by

$$(a, b) \in A \times B \equiv a \in A \wedge b \in B$$

    If $A \neq \emptyset$ and $B \neq \emptyset$, then $\bigtimes^{-}(A \times B)\, 0 = A$ and $\bigtimes^{-}(A \times B)\, 1 = B$.
  * Expressing dependent types: letting $T := a : A \,.\, B$ with a free in B,

$$\bigtimes (a : A \,.\, B) = \{f : A \to \bigcup\, a : A \,.\, B \mid \forall\, a : A \,.\, f\, a \in B\}$$

    Convenient shorthand: $A \ni a \to B_a$ for $\bigtimes\, a : A \,.\, B_a$
    Example: $A^{+} \ni x \to A^{\#\, x - 1}$ for the type of the σ-operator.
    Other use: clearer in chained dependencies, e.g., $A \ni a \to B_a \ni b \to C_{a,b}$.

4 Conclusion

- Mathematical concepts and operators arising from a seemingly specialized area of engineering (signal flow realizations) can be made generic and thereby extend their applicability to a much wider area of engineering and mathematics.
- This was illustrated by an algebraic and functional formulation of predicate calculus, providing a convenient formalism for specification and reasoning about software systems of an evolutionary nature.