Formal calculation with Functions, Predicates and Quantifiers in Software, Computer and Electronics Engineering

Raymond Boute

INTEC — Ghent University

Time	Section number and subject	
08:30–09:25	0	Introduction: motivation and approach
	1, 2	Simple expressions and equational calculation, examples
09:35–10:30	3, 4	Point-wise and point-free expressions, application examples
11:00–11:40	5	Binary algebra & proposition calculus: calculation rules
1:50–12:30	6	Binary algebra & proposition calculus: application examples
	7	Sets and functions: equality and calculation
13:30–14:25	8	Functional Predicate Calculus: calculating with quantifiers
14:35–15:30	9	Functional Predicate Calculus: application examples
	10	Well-foundedness, induction and application examples
16:00–16:40	11, 12	Funmath and Generic Functionals
16:50–17:30	13	Various examples in continuous and discrete mathematics

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

0 Introduction: Motivation and Approach

0.0 Motivation

a. Mathematics in Engineering

- Rationale: 2 variants (by C. Michael Holloway)
- Simple parallels between software and classical engineering

b. Formal Methods

- Definition
- Advantages
- Rough classification

0.1 Choice of approach

a. Why not “off-the-shelf” or “cut-and dried” mathematics?

b. Chosen approach: Functional Mathematics

0.0 Motivation

a. Mathematics in Engineering

- Rationale: 2 variants
  C. Michael Holloway, "Why engineers should consider formal methods"
  `http://techreports.larc.nasa.gov/ltrs/PDF/1997/mtg/NASA-97-16dasc-cmh.pdf`

Traditional rationale: reasoning leading to the conclusion that
"Reducing software errors and design effort requires formal methods"
Critique: complex argumentation, many unnecessary assumptions.

Revised rationale:

Software engineers strive to be true engineers (Q1);
true engineers use appropriate mathematics (Q2);
therefore, software engineers should use appropriate mathematics (Q3).
Thus, given that formal methods is the mathematics of software (Q4),
Software engineers should use appropriate formal methods (Q5).

Clearly (Q3) is the issue for software engineers aspiring to be true engineers!

a. **Mathematics in Engineering** (continued)

- A parallel between electronics and software at the engineering level

Difference between professional engineers and other designers

> Professional engineers can often be distinguished from other designers by the engineers' ability to use mathematical models to describe and analyze their products
> David L. Parnas, "Predicate Logic for Software Engineering",
> *IEEE Trans. Software Engineering 19*, 9, pp. 856–862 (Sept. 1993)

Technicians level (`http://www.passpathways.on.ca/resources/PASSWEB/page93.html`): "Technicians generally do not do design work ... [thus require less] math than in a 3-year technology program. For example, calculus [is not always included]".

Yet, even so, many electronics technicians courses include calculus, e.g., in a typical program preparing entry-level electronic engineering technicians (reference: `http://selland.boisestate.edu/academic_programs/programs/ETAAS.htm`)

Suggestive question: In practice, how does the professional level of software engineers compare to that of electronics technicians? (not further elaborated ...)

a. **Mathematics in Engineering** (continued): giving software engineers a mathematical background at a level comparable to that of classical engineers

Classification	Classical engineering	Software/computer engineering
Basic	Analysis, Linear algebra Probability and Statistics Discrete mathematics (combinatorics, graphs)	Formal proposition & predicate logic Relations, (h.-o.) functions, orderings Lambda calculus (elements) Lattice theory, Induction principles, . . .
Targeted	Physics Control theory Stochastic processes Information Theory, . . .	Formal languages and automata Formal language semantics Concurrency (parallel, mobile etc.) Type theory, . . .
"Advanced"	Functional analysis Distribution Theory Hilbert and Banach spaces Measure theory, . . .	Category theory Unified algebra Modal logic Co-algebras and co-induction, . . .

Remark: the "basic" mathematics for software engineering is equally relevant to classical engineering mathematics — hence useful in *any* engineering curriculum!

b. **Formal Methods**

- Definition/essence

  $\boxed{\text{UT FACIANT OPUS SIGNA}}$ “Let the symbols do the work”

  (Maxim of the congress series “Mathematics of Program Construction”)

  - “Formal methods” means more than just “mathematical methods”
  - Traditional math: manipulating expressions by interpretation (symbols no more than a notation, often even ambiguous/incondsistent)
  - Formal math: manipulating expressions om the basis of their form (precise calculation rules, reasoning guided by structure)

- Advantages of the formal approach
  - Added dimension: reasoning guided by the shape of the expressions
    Note: standard practice in use of calculus, pitiful in traditional proofs
  - Multi-pronged attack to problems (developing parallel intuition)
  - Greater diversity of subjects can be captured by uniform methodology
  - Especially useful for exploring unfamiliar terrain
    (where intuition based on interpretationis still lacking)
  - Avoiding hidden assumptions, making all assumptions explicit
  - Importance for supporting true declarativity: verifying that specifications reflect intent by proving multiple formalizations equivalent
    (in this way, the formal approach captures interpretation issues)

- Rough classification of formal methods
  - Hardcore formal methods: using formal mathematics

    Advantages: deeper understanding,
    better handling of conceptual complexity and diversity
    generality, adaptability, more support for innovation

    Disadvantage: learning curve (steeper, more time and effort)
  - Lightweight formal methods: using some tool

    Advantage: lower threshold, faster near-term user motivation

    Disadvantages: limited scope (specialized),
    use is ineffective and unsafe without hardcore background

Related issue: use of automated tools

- Some tools support (some) hardcore methods
- Tools can never supplant hardcore methods

0.1 Choice of approach

Goal: $\boxed{\text{UT FACIANT OPUS SIGNA}}$ (no second-rate formalisms)

a. **Why not "off-the-shelf" or "cut-and -dried" mathematics?**
Criterion: support for formal calculation in everyday practice
This support is inversely proportional to the needs in software engineering

- Well-developed in classical mathematics
  Examples: algebra (due: Viète, Descartes), calculus (due: Leibniz)
  Calculation in everyday practice is *de facto* formal (or nearly so)
- Rather poor for discrete mathematics, e.g.,
  Formal rules for $\sum$ rarely stated, often error-prone (Mathematica)
  Ellipsis (as in $a_0 + a_1 + \cdots + a_{n-1}$), other ad hoc conventions
- Woefully inadequate in logic and set theory, e.g.,
  Quantifiers $\exists$ and $\forall$ in everyday practice only used as abbreviations for words in natural language ("there exists" and "for all")
  Set comprehension/selection: disparate, ambiguous conventions like $\{x \in X \mid x > y\}$ and $\{x^2 \mid x \in X\}$, no formal calculation rules

Example A: symbolic manipulation in engineering calculus is essentially formal

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

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

The only shortcoming w.r.t. the “formal norm” is the absence of justifications.

Major defect: only for expressions in arithmetic, differential, integral calculus, not in logical reasoning. This constitutes a serious $\boxed{\text{style breach}}$.

> "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
>
> $$\text{R. c. L. 2 p. di m. 11 L} \qquad \text{for our} \qquad 3\sqrt{2+11i}.$$
>
> Many professional mathematicians to this day use the quantifiers (∀,∃) in a similar fashion,
>
> $$\exists\delta > 0 \text{ s.t. } |f(x) - f(x_0)| < \epsilon \text{ if } |x - x_0| < \delta, \text{ for all } \epsilon > 0,$$
>
> 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")

Key issue (quote): "understanding the logical structure of the arguments"

Example B: ambiguities in conventions for sets

- Patterns typical in mathematical writing:

Patterns	with logical expression p $\{x \in X \mid p\}$	with any expression e $\{e \mid x \in X\}$
Examples	$\{m \in \mathbb{Z} \mid m < n\}$	$\{n \cdot m \mid m \in \mathbb{Z}\}$

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 $\mathit{Even} := \{2 \cdot m \mid m \in \mathbb{Z}\}$ in

Patterns	$\{x \in X \mid p\}$ (logical p)	$\{e \mid x \in X\}$ (any e)
Examples	Both patterns are matched by both examples $\{n \in \mathbb{Z} \mid n \in \mathit{Even}\}$ and $\{n \in \mathit{Even} \mid n \in \mathbb{Z}\}$	

So $\{n \in \mathbb{Z} \mid n \in \mathit{Even}\}$ and $\{n \in \mathit{Even} \mid n \in \mathbb{Z}\}$ are ambiguous.

- Worse: such conventions *prohibit even the formulation of calculation rules*!
Formal calculation with such 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 $\forall\, x \in \mathbb{R} \,.\, x^2 \geq 0$.

b. **Choice: the Functional Mathematics approach**

- Formal calculation throughout continuous and discrete mathematics

  Realizing the principle UT FACIANT OPUS SIGNA

  Precise rules, calculational chaining of steps: eliminating the style breach.

  $$\boxed{\begin{array}{llll} a & R & \langle\text{Justification for } a\,R\,b\rangle & b \\ & S & \langle\text{Justification for } b\,S\,c\rangle & c \end{array}}$$

  This allows chaining steps in a clear synoptic way with minimal repetition.
  For arithmetic reasoning, R and S are $=$ or $\leq$ or $<$ (as illustrated).
  For logical reasoning, R and S are $\equiv$ or $\Rightarrow$ etc.

- Supported by adequate conventions
  - Entirely free of ambiguities and inconsistencies
  - Extremely simple syntax: 4 syntactic constructs only.
- Concrete embodiment: Funmath
  - Bonus: captures existing conventions in easily recognizable form
  - Provides new, useful forms of expression by orthogonal combination

An example (just to illustrate the style; calculation rules introduced later)

Proposition 2.1. for any function $f:\mathbb{R}\nrightarrow\mathbb{R}$, any subset S of $\mathcal{D}\,f$ and any a adherent to S,

(i) $\exists\,(L:\mathbb{R}\,.\,L\,\mathsf{islim}_f\,a) \Rightarrow \exists\,(L:\mathbb{R}\,.\,L\,\mathsf{islim}_{f\rceil S}\,a)$,

(ii) $\forall\,L:\mathbb{R}\,.\,\forall\,M:\mathbb{R}\,.\,L\,\mathsf{islim}_f\,a \wedge M\,\mathsf{islim}_{f\rceil S}\,a \Rightarrow L = M$.

Proof for (ii): Letting $b\,R\,\delta$ abbreviate $\forall\,x:S\,.\,|x-a|<\delta \Rightarrow |f\,x-b|<\epsilon$,

$$
\begin{array}{lll}
\multicolumn{3}{l}{L\,\mathsf{islim}_f\,a \wedge M\,\mathsf{islim}_{f\rceil S}\,a} \\
\Rightarrow & \langle\text{Hint in proof for (i)}\rangle & L\,\mathsf{islim}_{f\rceil S}\,a \wedge M\,\mathsf{islim}_{f\rceil S}\,a \\
\equiv & \langle\text{Def. } \mathsf{islim}\text{, hypoth.}\rangle & \forall\,(\epsilon:\mathbb{R}_{>0}\,.\,\exists\,\delta:\mathbb{R}_{>0}\,.\,L\,R\,\delta) \wedge \forall\,(\epsilon:\mathbb{R}_{>0}\,.\,\exists\,\delta:\mathbb{R}_{>0}\,.\,M\,R\,\delta) \\
\equiv & \langle\text{Distributivity } \forall/\wedge\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,(\delta:\mathbb{R}_{>0}\,.\,L\,R\,\delta) \wedge \exists\,(\delta:\mathbb{R}_{>0}\,.\,M\,R\,\delta) \\
\equiv & \langle\text{Rename, dstr. } \wedge/\exists\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,\delta:\mathbb{R}_{>0}\,.\,\exists\,\delta':\mathbb{R}_{>0}\,.\,L\,R\,\delta \wedge M\,R\,\delta' \\
\Rightarrow & \langle\text{Closeness lemma}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,\delta:\mathbb{R}_{>0}\,.\,\exists\,\delta':\mathbb{R}_{>0}\,.\,a\in\mathsf{Ad}\,S \Rightarrow |L-M|<2\cdot\epsilon \\
\equiv & \langle\text{Hypoth. } a\in\mathsf{Ad}\,S\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,\delta:\mathbb{R}_{>0}\,.\,\exists\,\delta':\mathbb{R}_{>0}\,.\,|L-M|<2\cdot\epsilon \\
\equiv & \langle\text{Const. pred. sub } \exists\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,|L-M|<2\cdot\epsilon \\
\equiv & \langle\text{Vanishing lemma}\rangle & L-M=0 \\
\equiv & \langle\text{Leibniz, group } +\rangle & L=M
\end{array}
$$

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

1 Simple expressions and equational calculation

1.0 Purpose: formalizing equational calculation

1.1 Syntax of simple expressions and equational formulas

a. Syntax conventions

b. Syntax of simple expressions

c. Syntax of equational formulas [and a sneak preview of semantics]

1.2 Formalizing substitution

1.3 Formal calculation with equality

a. Formal deduction: general

b. Deduction with equality

c. Equational calculation

1.0 Purpose: formalizing equational calculation

Typical calculational example from high school algebra, with explicit justifications

$$
\begin{array}{lcl}
\multicolumn{3}{l}{(a+b)\cdot(a-b)} \\
\quad = & \langle \text{Definition of } - \rangle & (a+b)\cdot(a+{}^{-}b) \\
\quad = & \langle \text{Distributivity of } \cdot \text{ over } + \rangle & (a+b)\cdot a+(a+b)\cdot{}^{-}b \\
\quad = & \langle \text{Commutativity of } \cdot \rangle & a\cdot(a+b)+{}^{-}b\cdot(a+b) \\
\quad = & \langle \text{Distributivity of } \cdot \text{ over } + \rangle & (a\cdot a+a\cdot b)+({}^{-}b\cdot a+{}^{-}b\cdot b) \\
\quad = & \langle \text{Associativity of } + \rangle & a\cdot a+((a\cdot b+{}^{-}b\cdot a)+{}^{-}b\cdot b) \\
\quad = & \langle \text{Commutativity of } \cdot \rangle & a\cdot a+((b\cdot a+{}^{-}b\cdot a)+{}^{-}b\cdot b) \\
\quad = & \langle \text{Semidistributivity of } {}^{-} \text{ over } \cdot \rangle & a\cdot a+((b\cdot a+{}^{-}(b\cdot a))+{}^{-}(b\cdot b) \\
\quad = & \langle \text{Additive inverse} \rangle & a\cdot a+(0+{}^{-}(b\cdot b)) \\
\quad = & \langle \text{Additive unit element} \rangle & a\cdot a+{}^{-}(b\cdot b) \\
\quad = & \langle \text{Definition of } - \rangle & a\cdot a-b\cdot b \\
\quad = & \langle \text{Definition power} \rangle & a^2-b^2
\end{array}
$$

Example (continued): formulas corresponding to the textual justifications

Textual name	Symbolic formula
Definition of $-$	$x - y = x + {}^{-}y$
Distributivity of $\cdot$ over $+$	$x \cdot (y + z) = x \cdot y + x \cdot z$
Commutativity of $\cdot$	$x \cdot y = y \cdot x$
Associativity of $+$	$(x + y) + z = x + (y + z)$
Semidistributivity of ${}^{-}$ over $\cdot$	${}^{-}x \cdot y = {}^{-}(x \cdot y)$
Additive inverse	$x + {}^{-}x = 0$
Additive unit element	$0 + x = x$
Notation for squares	$x^2 = x \cdot x$

The equality $(a+b) \cdot {}^{-}b = {}^{-}b \cdot (a+b)$ is an *instantiation* of the rule $x \cdot y = y \cdot x$.

1.1 Syntax of simple expressions and equational formulas

a. Syntax conventions

Wirth's self-defining context-free grammar (CFG) — ASCII-oriented

```
syntax      ::=  { production }.
production  ::=  nonterminal "::=" regular "." .
regular     ::=  sequence { "|" sequence }.
sequence    ::=  item { item }.
item        ::=  nonterminal | terminal | "(" regular ")"
              |  "{" regular "}" | "[" regular "]".
terminal    ::=  """"character { character }"""".
```

Plus: our metalanguage conventions (not ASCII-oriented)

- Terminal symbols are indicated by an underscore (instead of Wirth's quotes).
- Single quote marks denote strings written in compact form, omitting commas and underscores. For instance, 'cab' stands for $\underline{c}$, $\underline{a}$, $\underline{b}$.
- Special quote marks ⟦ ⟧ are similar, but allow for metavariables.

b. Syntax of simple expressions

$$\begin{array}{lcl} \mathit{expression} & ::= & \mathit{variable} \mid \mathit{constant}_0 \mid \mathit{application} \\ \mathit{application} & ::= & \underline{(}\mathit{cop}_1\ \mathit{expression}\underline{)} \mid \underline{(}\mathit{expression}\ \mathit{cop}_2\ \mathit{expression}\underline{)} \end{array}$$

The variables, constants (including 1- and 2-argument operators) are domain-dependent. A simple example (arithmetic for natural numbers or integers:

$$\begin{array}{lcl@{\qquad}lcl} \mathit{variable} & ::= & \underline{x} \mid \underline{y} \mid \underline{z} & \mathit{cop}_1 & ::= & \underline{\mathit{succ}} \mid \underline{\mathit{pred}} \\ \mathit{constant}_0 & ::= & \underline{a} \mid \underline{b} \mid \underline{c} & \mathit{cop}_2 & ::= & \underline{+} \mid \underline{\cdot} \end{array}$$

Alternative style (names **E0** etc. for patterns are for easy reference)

Ev:	$[\![v]\!]$	where v is a variable from V
E0:	$[\![c]\!]$	where c is a constant from C_0
E1:	$[\![(\phi\, e)]\!]$	where ϕ is an operator from C_1 and e an expression
E2:	$[\![(d \star e)]\!]$	where $\star$ is an operator from C_2 and d and e expressions

Observe the conventions: $\mathit{variable}$ is a nonterminal, V the corresponding syntactic category, v a typical element of V (always the same initial letter).

c. Syntax of equational formulas [and a sneak preview of semantics]

$$\boxed{\mathit{formula} \ ::= \ \mathit{expression} \ \underline{=} \ \mathit{expression}}$$

A sneak preview of semantics

- Informal

  What does $x + y$ mean? This clearly depends on x and y.

  What does $x + y = y + x$ mean? Usually the commutativity of $+$.

- Formal: denotation space D, interpretation: family k of three functions: $k_0 \in C_0 \rightarrow D$ for the constants, $k_1 \in C_1 \rightarrow D \rightarrow D$ for the one-place operators, $k_2 \in C_2 \rightarrow D^2 \rightarrow D$ for the two-place operators.

  Finally: state space $S := V \rightarrow D$, meaning function $\mathcal{E} : E \rightarrow S \rightarrow D$ with

ref.	image definition for $\mathcal{E}$	for any state s in S and any
Mv:	$\mathcal{E}\,[\![v]\!]\,s \quad = s\,v$	variable v in V
M0:	$\mathcal{E}\,[\![c]\!]\,s \quad = k_0\,c$	constant c in C_0
M1:	$\mathcal{E}\,[\![(\phi\,e)]\!]\,s \quad = k_1\,\phi\,(\mathcal{E}\,e\,s)$	ϕ in C_1 and e in E
M2:	$\mathcal{E}\,[\![(d \star e)]\!]\,s = k_2\,\star\,(\mathcal{E}\,d\,s, \mathcal{E}\,e\,s)$	$\star$ in C_2, d in E and e in E

However:

ref.	image definition for $\mathcal{E}$		for any state s in S and any
Mv:	$\mathcal{E}\,[\![v]\!]\,s$	$= s\,v$	variable v in V
M0:	$\mathcal{E}\,[\![c]\!]\,s$	$= k_0\,c$	constant c in C_0
M1:	$\mathcal{E}\,[\![(\phi\,e)]\!]\,s$	$= k_1\,\phi\,(\mathcal{E}\,e\,s)$	ϕ in C_1 and e in E
M2:	$\mathcal{E}\,[\![(d \star e)]\!]\,s$	$= k_2 \star (\mathcal{E}\,d\,s, \mathcal{E}\,e\,s)$	$\star$ in C_2, d in E and e in E

FORGET FORGET FORGET FORGET FORGET FORGET . . .

We want to calculate FORMALLY, that is: without thinking about meaning!

1.2 Formalizing substitution

a. Definition

We define a "postfix" operator $[v := g]$ (written after its argument), parametrized by variable w and expression g (both arbitrary), with the intention that $e[w := g]$ is the result of substituting g for w in e.

We do this by recursion on the structure of the argument expression.

ref.	image definition for $[w := g]$		for arbitrary
Sv:	$v[w := g]$	$= (v = w)\,?\,g \dagger [\![v]\!]$	variable v in V
S0:	$c[w := g]$	$= [\![c]\!]$	constant c in C_0
S1:	$(\phi\, e)[w := g]$	$= [\![(\phi\, e[w := g])]\!]$	ϕ in C_1 and e in E
S2:	$(d \star e)[w := g]$	$= [\![(d[w := g] \star e[w := g])]\!]$	$\star : C_2$, $d : E$ and $e : E$

Legend for conditionals $c\,?\,b \dagger a$ ("if c then b else a") is $c\,?\,e_1 \dagger e_0 = e_c$.

Remarks

- Straightforward extension to simultaneous substitution: $[w', w'' := g', g'']$
- Convention: often we write $[^w_g$ for $[w := g]$ (saves horizontal space).

b. Example (detailed calculation)

$$
\begin{aligned}
&(a \cdot succ\, x + y)[x := z \cdot b] \\
&\quad = \quad \langle \text{Normalize} \rangle \quad \text{‘}((a \cdot (succ\, x)) + y)\text{’}[x := (z \cdot b)] \\
&\quad = \quad \langle \text{Rule } \textbf{S2} \rangle \quad [\![((a \cdot (succ\, x))[x := (z \cdot b)] + y[x := (z \cdot b)])]\!] \\
&\quad = \quad \langle \text{Rule } \textbf{S2} \rangle \quad [\![((a[x := (z \cdot b)] \cdot (succ\, x)[x := (z \cdot b)]) + y[x := (z \cdot b)])]\!] \\
&\quad = \quad \langle \text{Rule } \textbf{S1} \rangle \quad [\![((a[x := (z \cdot b)] \cdot (succ\, x[x := (z \cdot b)])) + y[x := (z \cdot b)])]\!] \\
&\quad = \quad \langle \text{Rule } \textbf{S0} \rangle \quad [\![((a \cdot (succ\, x[x := (z \cdot b)])) + y[x := (z \cdot b)])]\!] \\
&\quad = \quad \langle \text{Rule } \textbf{SV} \rangle \quad \text{‘}((a \cdot (succ\ (z \cdot b))) + y)\text{’} \\
&\quad = \quad \langle \text{Opt. par.} \rangle \quad \text{‘}a \cdot succ\,(z \cdot b) + y\text{’}
\end{aligned}
$$

Observe how the rules (repeated below) distribute s over the variables.

ref.	image definition for $[w := g]$	for arbitrary
Sv:	$v[w := g] \;=\; (v = w)\,?\,g \nmid [\![v]\!]$	variable v in V
S0:	$c[w := g] \;=\; [\![c]\!]$	constant c in C_0
S1:	$(\phi\, e)[w := g] \;=\; [\![(\phi\, e[w := g])]\!]$	ϕ in C_1 and e in E
S2:	$(d \star e)[w := g] \;=\; [\![(d[w := g] \star e[w := g])]\!]$	$\star : C_2$, $d : E$ and $e : E$

1.3 Formal calculation with equality

a. Formal deduction: general

An *inference rule* is a little table of the form

$$\boxed{\frac{Prems}{q}}$$

where $Prems$ is a collection of formulas (the *premisses*) and q is a formula (the *conclusion* or *direct consequence*).

It is used as follows. Given a collection $Hpths$ (the *hypotheses*). Then a formula q is a *consequence* of $Hpths$, written $Hpths \vdash q$, in case

- either q is a formula in the collection $Hpths$
- or q is the conclusion of an inference rule where the premisses are consequences of $Hpths$

An *axiom* is a hypothesis expressly designated as such (i.e., as an axiom). A *theorem* is a consequence of hypotheses that are axioms exclusively. (Note: axioms are chosen s. t. they are valid in some useful intrerpretation.)

b. Deduction with equality

The inference rules for equality are:

0. *Instantiation* (strict): $$\frac{p}{p[v := e]} \qquad (\alpha)$$

1. *Leibniz's principle* (non-strict): $$\frac{d' = d''}{e[v := d'] = e[v := d'']} \qquad (\beta)$$

2. *Symmetry of equality* (non-strict): $$\frac{e = e'}{e' = e} \qquad (\gamma)$$

3. *Transitivity* of equality (non-strict): $$\frac{e = e', e' = e''}{e = e''} \qquad (\delta)$$

Remarks

- An inference rule is *strict* if all of its premises must be theorems. Example: instantiating the axiom $x \cdot y = y \cdot x$ with $[x, y := (a + b), {}^{-}b]$
- Reflexivity of $=$ is captured by Leibniz (if v does not occur in e).

c. Equational calculation: embedding the inference rules into the format

$$\boxed{\begin{array}{llll} e_0 & = & \langle\text{justification}_0\rangle & e_1 \\ & = & \langle\text{justification}_1\rangle & e_1 \qquad \text{and so on.} \end{array}}$$

Using an inference rule with single premiss p and conclusion $e' = e''$ is written $e' \;\; = \langle p \rangle \;\; e''$, capturing each of the inference rules as follows.

(α) *Instantiation* Premiss p is a theorem of the form $d' = d''$, and hence the conclusion $p[v := e]$ is $d'[v := e] = d''[v := e]$ which has the form $e' = e''$. Example: $(a + b) \cdot {}^{-}b \;=\langle x \cdot y = y \cdot x \rangle \;\; {}^{-}b \cdot (a + b)$.

(β) *Leibniz* Premiss p, not necessarily a theorem, is of the form $d' = d''$ and the conclusion $e[v := d'] = e[v := d'']$ is of the form $e' = e''$. Example: if $y = a \cdot x$, then we may write $x + y \;\; =\langle y = a \cdot x \rangle \;\; x + a \cdot x$.

(γ) *Symmetry* Premiss p, not necessarily a theorem, is of the form $e'' = e'$. However, this simple step is usually taken tacitly.

(δ) *Transitivity* has two equalities for premisses. It is used implicitly to justify chaining $e_0 = e_1$ and $e_1 = e_2$ in the format shown to conclude $e_0 = e2$.

Additional conventions for making calculations even more succinct

- Tacit instantiation when using Leibniz's principle: if the premiss $d' = d''$ in (β) is the *instantiation* of a theorem p, then we *still* write the deduction step as $e' \quad = \langle p \rangle \; e''$, leaving the instantiation from p to $d' = d''$ implicit. Example: $x + z \cdot (a + y) \;=\!\langle x \cdot y = y \cdot x \rangle \; x + (a + y) \cdot z$.
- Tacit use of symmetry when using Leibniz's principle: if the sides of the equality in the premiss of (β) are not in the desired order, they are swapped tacitly, as expressed by metatheorem $d'' = d' \;\vdash\; e[v := d'] = d[v := d'']$. Example: using $y = a \cdot x$ as $a \cdot x = y$ in $x + a \cdot x \;=\!\langle y = a \cdot x \rangle \; x + y$.
- Tacit use of symmetry in the entire calculation, viz., deducing $e_k = e_0$ to deduce $e_0 = e_k$. This is helpful if the chain from e_k to e_0 is easier, which is often the case when e_k is "more structured" in some sense than e_0. For instance, proving $x^2 - y^2 = (x + y) \cdot (x - y)$ is easier starting from the right (why?)

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

2 Equational calculation: application examples

2.0 Axiomatic semantics of assignment in simple imperative languages

2.0.0 Syntax of a simple imperative language

2.0.1 Axiomatic semantics: Hoare style, Dijkstra style

2.0.2 Axiomatic semantics of assignment

2.1 Simple equational algebras

2.1.0 Some terminology about algebras

2.1.1 Proving properties for simple equational algebras

2.1.2 Applications in functional programming

2.0 Axiomatic semantics of assignment in simple imperative languages

2.0.0 Syntax of a simple imperative language

$$
\begin{array}{l}
\mathit{program} \;\; ::= \;\; \mathit{declaration}\ \mathit{command}. \\
\mathit{command} \;\; ::= \;\; \mathit{elementary} \mid \mathit{structured}. \\
\quad \mathit{elementary} \;\; ::= \;\; \mathit{assignment} \mid \underline{\textbf{skip}}. \\
\qquad \mathit{assignment} \;\; ::= \;\; \mathit{variable}\ \underline{:=}\ \mathit{expression}. \\
\quad \mathit{structured} \;\; ::= \;\; \mathit{composition} \mid \mathit{selection} \mid \mathit{repetition}. \\
\qquad \mathit{composition} \;\; ::= \;\; \mathit{command}\ \underline{;}\ \mathit{command}. \\
\qquad \mathit{selection} \;\; ::= \;\; \underline{\textbf{if}}\ \mathit{expression}\ \underline{\textbf{then}}\ \mathit{command}\ \underline{\textbf{else}}\ \mathit{command}\ \underline{\textbf{fi}}. \\
\qquad \mathit{repetition} \;\; ::= \;\; \underline{\textbf{while}}\ \mathit{expression}\ \underline{\textbf{do}}\ \mathit{command}\ \underline{\textbf{od}}. \\
\mathit{expression} \;\; ::= \;\; \mathit{variable} \mid \mathit{constant} \mid \mathit{application}.
\end{array}
$$

2.0.1 Axiomatic semantics: Hoare style, Dijkstra style

Let c be a command, a and p formulas (*assertions*).
A *Hoare-triple* or *pre-post formula* ("ppf") is a formula of the form

$$\boxed{\{a\}\,c\,\{p\}}$$

Two formulations: *partial* or *total* correctness, with intuitive explanation:

- *Partial correctness*: if the state before execution of c satisfies a and execution of c terminates, then the state after execution of c satisfies p.
- *Total correctness*: if the state before execution of c satisfies a, then execution of c terminates and the state after execution of c satisfies p.

Reducing inference rules (in favour of axioms) makes the style more elegant. Dijkstra's *weakest precondition* for a given command c and postcondition p is written $\mathsf{wp}\,c\,p$, and is defined by two properties:

- It is a precondition, i.e., $\{\mathsf{wp}\,c\,p\}\,c\,\{p\}$
- Any condition a satisfying $\{a\}\,c\,\{p\}$ is as least as strong ($\mathsf{wp}\,c\,p \preceq a$).

2.0.2 Axiomatic semantics of assignment

- Hoare and Dijkstra formulations

$$\boxed{\{p[v := e]\}\ \ \llbracket v := e \rrbracket\ \ \{p\} \qquad \text{and} \qquad \mathsf{wp}\,\llbracket v := e \rrbracket\, p = p[v := e].}$$

Less confusing: $\{p[^{v}_{e}\}\ \ \llbracket v := e \rrbracket\ \ \{p\}$ and $\mathsf{wp}\,\llbracket v := e \rrbracket\, p = p[^{v}_{e}$

- Intuitive engineering mechanics (rarely emphasized in the literature):
  - Explicitly distinguish before and after, writing v and v' respectively.
  - Hence v' satisfies the assertion $p[^{v}_{v'}$.
  - Clearly, v' depends on v as expressed by the equation $v' = e$.
  - An assertion for v is now derived via Leibniz: $p[^{v}_{v'} \ =\!\langle v' = e\rangle\ p[^{v}_{e}$.

Illustration: assignment $\mathsf{x} := \mathsf{x} + 3$, postcondition $\mathsf{x} = 7$

- With explicit markings: $\mathsf{x}' := \mathsf{x} + 3$ and $\mathsf{x}' = 7$
- This yields the equation $7 = \mathsf{x} + 3$, which is '$\mathsf{x} = 7$'$[^{\mathsf{x}}_{\mathsf{x}+3}$.
- Solving for x yields $\mathsf{x} = 4$.

- A simple example: swapping values
  - In languages with multiple assignment: $\mathsf{x}, \mathsf{y} := \mathsf{y}, \mathsf{x}$
    Proof: calculate the wp to establish the postcondition $\mathsf{y}, \mathsf{x} = u, v$.

$$\boxed{\begin{array}{lcll} \text{wp ‘x, y} := \text{y, x’ ‘y, x} = u, v\text{’} & = & \langle\text{Def. wp}\rangle & \text{‘y, x} = u, v\text{’}[\text{x, y} := \text{y, x}] \\ & = & \langle\text{Rule Sv}\rangle & \text{‘x, y} = u, v\text{’} \end{array}}$$

  - Without multiple assignment: $\mathsf{t} := \mathsf{x}\ ;\ \mathsf{x} := \mathsf{y}\ ;\ \mathsf{y} := \mathsf{t}$.
    So now we must calculate wp ‘$\mathsf{t} := \mathsf{x}\ ;\ \mathsf{x} := \mathsf{y}\ ;\ \mathsf{y} := \mathsf{t}$’ ‘$\mathsf{y}, \mathsf{x} = u, v$’.
    The composition rule in the calculation is wp $[\![c\,;\,c']\!]\ p =$ wp c (wp $c'\ p$).

$$\boxed{\begin{array}{lll} \multicolumn{3}{l}{\text{wp ‘t} := \text{x ; x} := \text{y ; y} := \text{t’ ‘y, x} = u, v\text{’}} \\ \quad = & \langle\text{Composition}\rangle & \text{wp ‘t} := \text{x ; x} := \text{y’ (wp ‘y} := \text{t’ ‘y, x} = u, v\text{’)} \\ \quad = & \langle\text{Assignment}\rangle & \text{wp ‘t} := \text{x ; x} := \text{y’ ‘t, x} = u, v\text{’} \\ \quad = & \langle\text{Composition}\rangle & \text{wp ‘t} := \text{x’ (wp ‘x} := \text{y’ ‘t, x} = u, v\text{’)} \\ \quad = & \langle\text{Assignment}\rangle & \text{wp ‘t} := \text{x’ ‘t, y} = u, v\text{’} \\ \quad = & \langle\text{Assignment}\rangle & \text{‘x, y} = u, v\text{’} \end{array}}$$

2.1 Simple equational algebras

2.1.0 Some terminology about algebras

Simple:	only one kind (type) of expressions
Equational:	axioms and theorems expressed as equalities
Concrete:	pertaining to a specific class of objects (e.g. numbers, lists)
Abstract:	(“default”) capturing common properties of concrete algebras

2.1.1 Proving properties for simple equational algebras

a. Identity elements

Definitions

left identity constant l such that $l \cdot x = x$ is a theorem

right identity constant r such that $x \cdot r = x$ is a theorem

2-sided ident. constant e such that $e \cdot x = x$ and $x \cdot e = x$ are theorems

Note: an *identity* is often called a *unit element* (not be confused with a *unit* in a ring, which is an element whose inverse is also in the ring).

A simple theorem If l is a left identity and r is a right identity, then $l = r$.

Proof:

$$\boxed{\begin{aligned} l &= \langle x \cdot r = x \text{ with } x := l \rangle \quad l \cdot r \\ &= \langle l \cdot x = x \text{ with } x := r \rangle \quad r \end{aligned}}$$

Note: in the proof, we made the substitution explicit (just for once).

b. Extensionality: if $\delta\, x = \sigma\, x$ is a theorem for operators δ and σ, then $\delta = \sigma$. A metatheorem justifying this is that $\delta = \sigma$ does not yield more than $\delta\, x = \sigma\, x$.

c. Associativity and inverses

Definitions If $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ is a theorem, then $\cdot$ is *associative*. Let e be a 2-sided identity, then we define:

left inverse operator	l such that $l\, x \cdot x = e$ is a theorem
right inverse operator	r such that $x \cdot r\, x = e$ is a theorem
2-sided inverse oper.	i such that $i\, x \cdot x = e$ and $x \cdot i\, x = e$ are theorems

A simple theorem If the operator $\cdot$ is associative and l is a left inverse operator and r is a right inverse operator, then $l = r$. Proof:

$$\boxed{\begin{array}{lcll} l\, x & = & \langle x = x \cdot e \rangle & l\, x \cdot e \\ & = & \langle e = x \cdot r\, x \rangle & l\, x \cdot (x \cdot r\, x) \\ & = & \langle \text{Associat. } \cdot \rangle & (l\, x \cdot x) \cdot r\, x \\ & = & \langle l\, x \cdot x = e \rangle & e \cdot r\, x \\ & = & \langle e \cdot x = x \rangle & r\, x \end{array}}$$

Hence $l = r$ by function extensionality.

2.1.2 Applications in functional programming

a. General: most proofs about functional programs are based on equational axioms characterizing the functions of interest.
The simple proofs involve just equational calculation as shown for algebra.
More complicated/interesting cases involve induction (discussed later).
See e.g. Richard Bird, *Introduction to Functional Programming using Haskell*. Prentice Hall, London (1998)

b. Specific: most functions and data structures in functional programming have many useful equational algebraic properties. Some examples:

- The *concatenation* operator $+\!\!+$ for lists is associative and the empty list ε is a two-sided identity element, since one can prove

$$x +\!\!+ (y +\!\!+ z) = (x +\!\!+ y) +\!\!+ z$$
$$x +\!\!+ \varepsilon = x = \varepsilon +\!\!+ x$$

- Bijective functions from a set onto itself form a group under composition. More generally, our generic composition operator $(\circ)$ defined later satisfies associativity $f \circ (g \circ h) = (f \circ g) \circ h$ for arbitrary functions f, g, h.

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

3 Point-wise and point-free styles of expression and calculation

3.0 Motivation

3.1 Lambda calculus as a model for handling dummies

3.1.0 “Inventing” lambda abstraction as a function interface

3.1.1 Syntax of “pure” lambda terms; bound and free variables

3.1.2 Designing axioms and substitution rules

3.1.3 Calculation rules and axiom variants

3.1.4 Practical calculation methods with shortcuts

3.1.5 Redexes and the Church-Rosser property

3.2 Combinator calculus as a point-free formalism

3.2.0 Syntax and calculation rules

3.2.1 Converting lambda terms into combinator terms

3.0 Motivation

a. Common motivation

- Handling functions as “first-class” objects, in particular: allowing them as function arguments, and allowing function applications as functions
- Contrasting point-wise and point-free calculation

b. Specific motivation for lambda calculus

- An interface for packaging expressions as functions
- A conceptual frame of reference for handling free and bound variables
- A basis for later extension with domain types, synthesizing common mathematical notation
- An archetype for point-wise calculation

c. Specific motivation for combinator calculus

- An archetype for point-free calculation

3.1 Lambda calculus as a model for handling dummies

3.1.0 "Inventing" lambda abstraction as a function interface

a. Goals

- Denoting functions by anonymous expressions
  For instance, $succ\,3 = 4$ uses a *function name* (operator) $succ$.
- Encapsulating expressions as functions
  For instance, $x + y$ is not a function, e.g., $(x + y)\,6$ is nonsense

b. Design considerations

- A typical *function definition* is an axiom of the form $\boxed{f\,v = e}$.
  Example: $foo\,x = 3 \cdot x$. This is standard in functional programming.
- Application to an expression is *instantiation* of this axiom:
  $(f\,v = e)[v := d]$ or, after elaboration, $\boxed{f\,d = e[v := d]}$
- Conclusion: any expression capturing f must mention both e and v.

c. Typical solution: $\boxed{f = \lambda v.e}$. By Leibniz's principle: $\boxed{(\lambda v.e)\,d = e[v := d]}$.
(assuming f not in e). This becomes the basic axiom of lambda calculus.

3.1.1 Syntax of “pure” lambda terms; bound and free variables

a. Syntax: the expressions are the *(lambda) terms* defined by the following CFG.

$$\boxed{\textit{term} \;\; ::= \;\; \textit{variable} \mid \underline{(}\textit{term term}\underline{)} \mid \underline{(}\underline{\lambda}\textit{variable}\underline{.}\textit{term}\underline{)}}$$

Examples: x $\quad (xy)$ $\quad (\lambda x.x)$ $\quad (\lambda x.(\lambda y.(x(yz))))$

We write Λ for the syntactic category, $L \mathinner{..} R$ as metavariables for terms, u, v, w as metavariables for variables, **C**, **D**, **I** etc, as shorthands for certain terms.

b. Conventions for making certain parentheses and dots optional

- Optional: outer parentheses in (MN), and in $(\lambda v.M)$ if standing by itself or as an abstrahend, e.g., $\lambda v.MN$ stands for $\lambda v.(MN)$, not $(\lambda v.M)N$.
- Application “associates to the left”, e.g., (LMN) stands for $((LM)N)$.
- Nested abstractions may be merged by writing $\lambda u.\lambda v.M$ as $\lambda uv.M$.

So $\lambda x.y(\lambda xy.xz(\lambda z.xyz))yz$ is $(\lambda x.(((y(\lambda x.(\lambda y.((xz)(\lambda z.((xy)z))))))y)z))$.

c. Terminology: a form (MN) is an *application* and $(\lambda v.M)$ an *abstraction*. In $(\lambda v.M)$, the $\lambda v.$ is the *abstractor* and M the *abstrahend* or the *scope of λv*.

d. Bound and free variables

- Definitions
  - Every occurrence of v in $\lambda v.M$ is called *bound*.
  - Occurrences that are not bound are called *free*.
  - A term without free variables is a *closed term* or a (lambda-)*combinator*.
  - Bound variables are also called *dummies*.
- Examples
  - In $\lambda x.y(\lambda xy.xz(\lambda z.xyz))yz$ number all occurrences from 0 to 11. The only free occurrences are those of y and z in positions 1, 5, 10, 11.
  - An operator φ giving the set of variables that occur free in a term:

    $$\boxed{\varphi\,[\![v]\!] = \iota\,v \quad \varphi\,[\![(M\,N)]\!] = \varphi\,M \cup \varphi N \quad \varphi\,[\![(\lambda v.M)]\!] = (\varphi\,M) \setminus (\iota\,v).}$$

    Legend: ι for singleton sets, $\cup$ for set union, $\setminus$ for *set difference*.
  - Typical (and important) combinators are $\lambda xyz.x(yz)$ and $\lambda xyz.xzy$ and $\lambda x.(\lambda y.x(yy))(\lambda y.x(yy))$, abbreviated **C**, **T**, **Y** respectively.

3.1.2 Designing axioms and substitution rules

a. Basic goal: ensure correct generalization of $(\lambda v.e)\, d = e[v := d]$, i.e.,

$$\boxed{\textsc{Axiom, } \beta\textsc{-conversion:}\quad (\lambda v.M)\, N = M[v := N]}$$

This requires defining $M[v := N]$ for lambda terms.

b. Pitfall to be avoided: "naive substitution"

$$\begin{array}{lcll}
(\lambda xy.xy)yx & = & \langle\text{Normalize}\rangle & ((\lambda x.(\lambda y.xy))y)x \\
 & = & \langle\beta\text{-convers.}\rangle & ((\lambda y.xy)[x := y])x \\
 & = & \langle\text{Naive subst.}\rangle & (\lambda y.yy)x \qquad \text{(wrong!)} \\
 & = & \langle\beta\text{-convers.}\rangle & (yy)[y := x] \\
 & = & \langle\text{Var. subst.}\rangle & xx \qquad \text{(expected: } yx\text{)}
\end{array}$$

What happened? In $(\lambda x.(\lambda y.xy))y$, the last occurrence of y is free.
By the naive substitution in $(\lambda y.xy)[x := y]$ it becomes bound by λy.
This causes a "name clash" with other variables already bound by λy.
Such a clash would not happen in $(\lambda uv.uv)yx$, which yields yx.

c. Avoidance principle: choice of names for dummies is incidental.
Justification: function extensionality (if $\delta\, x = \sigma\, x$ is a theorem, then $\delta = \sigma$).
Application to lambda terms: for "new" variable u (i.e., not already used)

$$\begin{array}{rcll}
(\lambda v.M)N & = & \langle \beta\text{-conversion} \rangle & M[v := N] \\
 & = & \langle \text{Remark below} \rangle & M[v := u][u := N] \\
 & = & \langle \beta\text{-conversion} \rangle & (\lambda u.M[v := u])N
\end{array}$$

Remark: this step assumes that substitution causes no other unwanted effects.
However, for the time being we do *not* assume extensionality, but define:

d. Definitive substitution rules

$$\boxed{\begin{array}{lrcll}
\textbf{Svar:} & v[^{w}_{L} & = & (v = w)\,?\,L \vdash [\![v]\!] & \\
\textbf{Sapp:} & (M\,N)[^{w}_{L} & = & (M[^{w}_{L}\,N[^{w}_{L}) & \\
\textbf{Sabs:} & (\lambda v.M)[^{w}_{L} & = & (\lambda u.M[^{v}_{u}[^{w}_{L}) & \quad (\text{new } u)
\end{array}}$$

Variant: $(\lambda v.M)[^{w}_{L} = (v = w)\,?\,(\lambda v.M) \vdash (v \not\in \varphi\, L)\,?\,(\lambda v.M[^{w}_{L}) \vdash (\lambda u.M[^{v}_{u}[^{w}_{L})$,
checks for name clashes; if there are none, introducing new u is unnecessary.

3.1.3 Calculation rules and axiom variants

a. The rules of equality: symmetry, transitivity and Leibniz's principle:

$$\boxed{\frac{M = N}{N = M} \qquad \frac{L = M \quad M = N}{L = N} \qquad \frac{M = N}{L[v := M] = L[v := N]}}$$

b. The proper axioms common to most variants of the lambda calculus

AXIOM, β-CONVERSION:	$(\lambda v.M)N = M[^v_N$
AXIOM, α-CONVERSION:	$(\lambda v.M) = (\lambda w.M[^v_w)$ provided $w \not\in \varphi\, M$

Certain authors consider α-conversion subsumed by syntactic equality.

c. Specific additional axioms characterizing variants of the lambda calculus.

(i) *Rule ξ:* $\dfrac{M = N}{\lambda v.M = \lambda v.N}$ (note: extends Leibniz's principle)

(ii) *Rule η* (or η-**conversion**): $(\lambda v.Mv) = M$ provided $v \not\in \varphi\, M$

(iii) *Rule ζ* (or **extensionality**): $\dfrac{Mv = Nv}{M = N}$ provided $v \not\in \varphi\,(M, N)$

Note: given the rules in (a) and (b), rule ζ is equivalent to ξ and η combined. Henceforth we assume all these rules.

3.1.4 Practical calculation methods with shortcuts

a. Convention: we write $M =_\beta N$ for $M \ = \langle \beta$-conversion & substitution$\rangle\ N$ etc. Examples: $(\lambda xy.xy)z =_\beta \lambda y.zy$ and $(\lambda xy.xy)y =_\beta \lambda z.yz$; furthermore, $(\lambda xy.xy) =_\alpha \lambda xz.xz$ and $(\lambda x.yzx) =_\eta yz$ and $\lambda xy.xy =_\eta \lambda x.x$.

b. Calculating with shortcuts (but always know what you are doing)

- Using naive substitution $(\lambda v.M)[{}^{w}_{L} = \lambda v.M[{}^{w}_{L}$ provided choice of dummies avoids clashes. Example: $(\lambda xz.xz)yx =_\beta (\lambda z.yz)x =_\beta yx$.
- When introducing metavariables, assume the denoted terms contain none of the dummies present in the calculation. Example: $(\lambda xyz.x(yuz))LMN =_\beta (\lambda yz.L(yuz))MN =_\beta (\lambda z.L(Muz))N =_\beta L(MuN)$
- In calculation with (lambda) combinators, use encapsulated properties. Example: $\lambda xyz.x(yz)$, abbreviated **C**, has property **C**$LMN = L(MN)$; similarly, $\lambda xyz.xzy$, abbreviated **T**, has property **T**$LMN = LNM$. Note: in calculations, add metavariables as needed to enable properties. Example: given **CTT**, enable **C**$LMN = L(MN)$ by evaluating **CTT**L.

c. Some often-used lambda combinators

Name	*Symbol*	*Term*	*Property*
Composition	**C**	$\lambda xyz.x(yz)$	$\mathbf{C}LMN = L(MN)$
Duplication	**D**	$\lambda x.xx$	$\mathbf{D}M = MM$
Identity	**I**	$\lambda x.x$	$\mathbf{I}N = N$
Constant	**K**	$\lambda xy.x$	$\mathbf{K}MN = M$
Shuffle	**S**	$\lambda xyz.xz(yz)$	$\mathbf{S}LMN = LN(MN)$
Transposition	**T**	$\lambda xyz.xzy$	$\mathbf{T}LMN = LNM$

d. A calculation example: deriving a property of **CTT**

$$\begin{aligned} \mathbf{CTT}LMN &= \langle \text{Prop. } \mathbf{C} \rangle \quad \mathbf{T}(\mathbf{T}L)MN \\ &= \langle \text{Prop. } \mathbf{T} \rangle \quad \mathbf{T}LNM \\ &= \langle \text{Prop. } \mathbf{T} \rangle \quad \mathbf{L}MN \\ &= \langle \text{Prop. } \mathbf{I} \rangle \quad \mathbf{I}LMN \end{aligned}$$

Hence, by extensionality (thrice), $\mathbf{CTT} = \mathbf{I}$.

Note: during the calculation, L, M, N were added one by one, as need arises.

3.1.5 Redexes and the Church-Rosser property

a. Redexes

- A β-*redex* is a term of the form $(\lambda v.M)N$. Example: $(\lambda xy.yx)(\lambda x.y)$
- A η-*redex* is a term of the form $\lambda v.Mv$ (met $v \notin \varphi M$).
- Warning example: $\lambda x.x(\lambda y.y)x$ contains no redex.

b. Normal forms

- A $\beta\eta$-*normal form* (or simply *normal form*) is a term containing no redex.
- A term *has a normal form* if it can be reduced to a normal form. Examples:
  - $(\lambda xyz.x(yz))(\lambda x.y)$ has normal form $\lambda uz.y$.
  - $\lambda xyz.yxz$ has normal form $\lambda xy.yx$.
  - $(\lambda x.xx)(\lambda x.xx)$ has no normal form.

c. Church-Rosser property: a term has at most one normal form.

3.2 Combinator calculus as a point-free formalism

3.2.0 Syntax and calculation rules

a. Syntax (CFG): $\boxed{\mathit{term} ::= \underline{\boldsymbol{K}} \mid \underline{\boldsymbol{S}} \mid \underline{(}\mathit{term}\ \mathit{term}\underline{)}}$

Conventions: outer parentheses optional, application associates to the left.

b. The calculation rules are

- The rules for equality: symmetry, transitivity, “Leibniz”.
  Since there are no variables, “Leibniz” is written $\frac{M = N}{LM = LN}$ and $\frac{M = N}{ML = NL}$.
- The axioms: $\boxed{\boldsymbol{K} L M = L \qquad \text{and} \qquad \boldsymbol{S} P Q R = P R (Q R)}$
- Extensionality: if M and N satisfy $ML = NL$ for any L, then $M = N$.

Calculation example: let M and N be arbitrary combinator terms, then

$$\boxed{\boldsymbol{S}\boldsymbol{K} M N \quad = \langle \text{by } \boldsymbol{S}\text{-axiom} \rangle \quad \boldsymbol{K} N (M N) \quad = \langle \text{by } \boldsymbol{K}\text{-axiom} \rangle \quad N}$$

By extensionality, $\boldsymbol{S}\boldsymbol{K} M$ is an identity operator. Abbreviation $\boldsymbol{I} := \boldsymbol{S}\boldsymbol{K}\boldsymbol{K}$.

3.2.1 Converting lambda terms into combinator terms

a. Method (Note: here combinators may mix with lambda terms: “Cλ-terms”).

- De-abstractor: for every v, we define a syntactic operator $\widehat{v}$ on Cλ-terms:

Argument term	Definition	Reference
Variable v itself:	$\widehat{v}v = \boldsymbol{I}$	(Rule I)
Variable w ($\neq v$):	$\widehat{v}w = \boldsymbol{K}v$	(Rule K’)
Constant c:	$\widehat{v}c = \boldsymbol{K}c$	(Rule K”)
Application:	$\widehat{v}(MN) = \boldsymbol{S}(\widehat{v}M)(\widehat{v}N)$	(Rule S)
Abstraction:	$\widehat{v}(\lambda w.M) = \widehat{v}(\widehat{w}M)$	(Rule D)

Property (metatheorem): For any Cλ-term M, $\boxed{\lambda v.M = \widehat{v}M}$.

- Shortcuts: for any Cλ-term M with $v \notin \varphi M$,

$$\boxed{\begin{array}{rcll} \widehat{v}M & = & \boldsymbol{K}M & \text{(Rule K)}, \\ \widehat{v}(Mv) & = & M & \text{(Rule } \eta\text{)}. \end{array}}$$

Rule K subsumes rules K’ and K”.

b. Example: converting **T** (namely, $\lambda xyz.xzy$) into a combinator term $\boldsymbol{T}$.

$\boldsymbol{T} = \widehat{x}\widehat{y}\widehat{z}xzy$ by rule D. We start with $\widehat{z}xzy$ separately, to avoid rewriting $\widehat{x}\widehat{y}$.

$$\begin{aligned}
\widehat{x}zy &= \langle\text{Rule S}\rangle & &\boldsymbol{S}(\widehat{z}xz)(\widehat{z}y)\\
&= \langle\text{Rule }\eta\rangle & &\boldsymbol{S}x(\widehat{z}y)\\
&= \langle\text{Rule K}\rangle & &\boldsymbol{S}x(\boldsymbol{K}y)\\[1ex]
\widehat{y}\boldsymbol{S}x(\boldsymbol{K}y) &= \langle\text{Rule S}\rangle & &\boldsymbol{S}(\widehat{y}\boldsymbol{S}x)(\widehat{y}\boldsymbol{K}y)\\
&= \langle\text{Rule }\eta\rangle & &\boldsymbol{S}(\widehat{y}\boldsymbol{S}x)\boldsymbol{K}\\
&= \langle\text{Rule K}\rangle & &\boldsymbol{S}(\boldsymbol{K}(\boldsymbol{S}x))\boldsymbol{K}\\[1ex]
\widehat{x}\boldsymbol{S}(\boldsymbol{K}(\boldsymbol{S}x))\boldsymbol{K} &= \langle\text{Rule S}\rangle & &\boldsymbol{S}(\widehat{x}\boldsymbol{S}(\boldsymbol{K}(\boldsymbol{S}x)))(\widehat{x}\boldsymbol{K})\\
&= \langle\text{Rule S}\rangle & &\boldsymbol{S}(\boldsymbol{S}(\widehat{x}\boldsymbol{S})(\widehat{x}\boldsymbol{K}(\boldsymbol{S}x)))(\widehat{x}\boldsymbol{K})\\
&= \langle\text{Rule K}\rangle & &\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\widehat{x}\boldsymbol{K}(\boldsymbol{S}x)))(\boldsymbol{K}\boldsymbol{K})\\
&= \langle\text{Rule S}\rangle & &\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\boldsymbol{S}(\widehat{x}\boldsymbol{K})(\widehat{x}\boldsymbol{S}x)))(\boldsymbol{K}\boldsymbol{K})\\
&= \langle\text{Rule }\eta\rangle & &\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\boldsymbol{S}(\widehat{x}\boldsymbol{K})\boldsymbol{S}))(\boldsymbol{K}\boldsymbol{K})\\
&= \langle\text{Rule K}\rangle & &\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{K})\boldsymbol{S}))(\boldsymbol{K}\boldsymbol{K})
\end{aligned}$$

For uniformity, we use for $\widehat{v}$ and $\lambda v.$ the same conventions about parentheses.

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

4 Point-wise and point-free styles: application examples

4.0 The pure lambda calculus as a functional programming language

4.0.0 Principle and choice of representation

a. Principle (Gödel): for classical computation, it suffices to consider some initial functions (zero, successor, selection) on natural numbers and ways of combining functions (composition, primitive recursion amd bounded minimization).

b. Convention for representation of natural numbers:
representation operator ρ mapping numbers to lambda terms.

- Church numerals (elaborated here):

$$\boxed{\rho\, n = \lambda xy.x^n y}$$

  Notation: $M^n N$ is an abbreviation for a term defined recursively on the syntax by $M^0 N = N$ and $M^{n+1} N = M(M^n N)$ for all n in $\mathbb{N}$.
  Examples: $\rho\, 0 = \lambda xy.y \qquad \rho\, 1 = \lambda xy.xy \qquad \rho\, 4 = \lambda xy.x(x(x(xy)))$.

- Other representations (left as exercises)

4.0.1 Example: deriving a lambda term for arithmetic from a specification

a. Successor function for natural numbers: $succ\, n = n + 1$.

b. Specification: we want a lambda combinator $\mathbf{S}^+$ that *realizes* $succ$ in the sense that it maps Church numerals to Church numerals and satisfies

$$\boxed{\mathbf{S}^+\,(\rho\, n) = \rho\,(succ\, n)}$$

For instance, $\mathbf{S}^+\,(\lambda xy.x(x(xy))) = \lambda xy.x(x(x(xy)))$

c. Calculational derivation (starting from the r.h.s. that contains known items)

$$\boxed{\begin{array}{llcl}
\rho\,(succ\, n) & = & \langle succ\, n = n+1 \rangle & \rho\,(n+1) \\
 & = & \langle \rho\, n = \lambda xy.x^n y \rangle & \lambda xy.x^{n+1} y \\
 & = & \langle M^{n+1} N = M(M^n N) \rangle & \lambda xy.x(x^n y) \\
 & = & \langle x^n y[^{x,y}_{M,N} = M^n N \rangle & \lambda xy.x((\lambda xy.x^n y)xy) \\
 & = & \langle \beta\text{-conversion} \rangle & (\lambda zxy.x(zxy))(\lambda xy.x^n y) \\
 & = & \langle \rho\, n = \lambda xy.x^n y \rangle & (\lambda zxy.x(zxy))(\rho\, n)
\end{array}}$$

Conclusion: letting $\mathbf{S}^+ := \lambda zxy.x(zxy)$ meets the specification.

4.0.2 Lambda terms for arithmetic, logic and conditional expressions

a. A complete collection of arithmetic operators:

- Predecessor: $\mathbf{P}^{-} := \lambda xyz.x(\mathbf{Q}y)(\mathbf{K}z)\mathbf{I}$ where $\mathbf{Q} := \lambda xyz.z(yx)$.
  Difficult to design (and not unique), but easy to prove by induction.
- Addition $\mathbf{A} := \lambda xy.x\mathbf{S}^{+}y$ and multiplication **M** (similar) are easy to design.

b. Logical constants and conditional expressions (easy to design)

- Traditional representation: $\mathbf{V} := \mathbf{K}$ (“verum”) and $\mathbf{F} := \mathbf{TK}$ (“falsum”)
  Negation: realized by **T** (obvious)
  Conditional: realized by **I** since $\mathbf{IV}MN = M$ and $\mathbf{IF}MN = N$
  Zero test: $\mathbf{Z} := \lambda x.x(\lambda xyz.z)(\lambda xy.x)$
- Number representation: $\mathbf{V} := \rho\,1$ and $\mathbf{F} := \rho\,0$
  Negation: $\mathbf{Not} := \lambda x.x(\lambda xyz.z)(\lambda xy.xy)$
  Conditional: $\mathbf{Cnd} := \lambda x.x(\lambda xyz.y)(\lambda xy.y)$
  Zero test $\mathbf{Z} := \lambda x.x(\lambda xyz.z)(\lambda xy.xy)$

4.0.3 Making things compete: a mechanism for recursion

a. Recursion is still missing. Consider the example

$$fac\,n = (n = 0)\,?\,1 \nmid fac\,(n-1)\cdot n$$

Transliterated into the functional programming language we have thus far:

$$\mathbf{Fac}\,N = \mathbf{Cnd}(\mathbf{Z}N)(\rho\,1)(\mathbf{M}(\mathbf{Fac}(\mathbf{P}^{-}N))N) \tag{1}$$

Problem: this yields no lambda term for **Fac**.

b. Solution (easily generalized): rewrite (1) as $\mathbf{Fac}\,N = \mathbf{H}\,\mathbf{Fac}\,N$ with

$$\mathbf{H}FN = \mathbf{Cnd}(\mathbf{Zro}\,N)(\rho\,1)(\mathbf{M}(F(\mathbf{P}^{-}N))N)$$

or, as a lambda term, $\mathbf{H} = \lambda fn.\mathbf{Cnd}(\mathbf{Zro}\,n)(\rho\,1)(\mathbf{M}(f(\mathbf{P}^{-}n))n)$.

A sufficient condition for $\mathbf{Fac}\,N = \mathbf{H}\,\mathbf{Fac}\,N$ is that **Fac** satisfies $\mathbf{Fac} = \mathbf{H}\,\mathbf{Fac}$.

Observe that $\mathbf{Y}H = H(\mathbf{Y}H)$ for any H (exercise).

Conclusion: $\mathbf{Fac} := \mathbf{YH}$ is a lambda term satisfying (1).

Note: if you simulate all this by a program, it actually works!

4.1 Applications in various domains

4.1.0 Other applications in programming

a. In programming languages

Lambda abstraction is available in the functional programming language Haskell.

A kind of lambda notation (less powerful, no functionals) is available in LISP.

Backus's FP is a point-free functional programming language (no variables)!

Lambda calculus is used for modelling language implementations

. . .

b. In the theory of programming

Variable binding, local variables, block structures etc. are best understood by comparison (similarities, differences) with lambda calculus.

Lambda calculus plays an important role in formal semantics, type theory, theory of computation etc.

The relational theory of data types of Backhouse et al. is point-free.

4.1.1 Applications in classical mathematics

a. There is a point-free variant of axiomatic set theory (Tarski, Givant)

b. Lambda calculus formalizes bindings and handling clashes in integrals, e.g.,

$$
\begin{aligned}
\left(\int_{-\infty}^{+\infty} e^{-x^2/2}\cdot \mathrm{d}x\right)^2
&= \langle\textsf{Def. square}\rangle && \left(\int_{-\infty}^{+\infty} e^{-x^2/2}\cdot \mathrm{d}x\right)\cdot\left(\int_{-\infty}^{+\infty} e^{-x^2/2}\cdot \mathrm{d}x\right)\\
&= \langle\textsf{Mul. const.}\rangle && \int_{-\infty}^{+\infty}\left(\int_{-\infty}^{+\infty} e^{-x^2/2}\cdot \mathrm{d}x\right)\cdot e^{-x^2/2}\cdot \mathrm{d}x\\
&= \langle\alpha\textsf{-conversion}\rangle && \int_{-\infty}^{+\infty}\left(\int_{-\infty}^{+\infty} e^{-y^2/2}\cdot \mathrm{d}y\right)\cdot e^{-x^2/2}\cdot \mathrm{d}x\\
&= \langle\textsf{Mul. const.}\rangle && \int_{-\infty}^{+\infty}\left(\int_{-\infty}^{+\infty} e^{-x^2/2}\cdot e^{-y^2/2}\cdot \mathrm{d}y\right)\cdot \mathrm{d}x\\
&= \langle\textsf{Merging}\rangle && \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-(x^2+y^2)/2}\cdot \mathrm{d}y\cdot \mathrm{d}x\\
&= \langle\textsf{Polar coord.}\rangle && \int_{0}^{2\cdot\pi}\int_{0}^{+\infty} e^{-r^2/2}\cdot r\cdot \mathrm{d}r\cdot \mathrm{d}\theta\\
&= \langle\textsf{Mul. const.}\rangle && \left(\int_{0}^{2\cdot\pi} \mathrm{d}\theta\right)\cdot\left(\int_{-\infty}^{+\infty} e^{-r^2/2}\cdot r\cdot \mathrm{d}r\right)\\
&= \langle\textsf{Integration}\rangle && 2\cdot\pi
\end{aligned}
$$

4.1.2 Applications in formal circuit description

a. To be completed

b. References given in situ

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

5 Binary algebra and proposition calculus

5.0 Motivation

5.1 Binary Algebra

5.1.0 Binary Algebra as a restriction of minimax algebra

5.1.1 Relation to various other algebras and calculi

5.2 Axiomatic proposition calculus

5.2.0 Syntax and conventions

5.2.1 Implication: axioms, formal deduction: from classical to calculational

5.2.2 Negation: making proposition algebra complete

5.2.3 Quick calculation rules and proof techniques

5.2.4 Brief summary of derived operators and calculation rules

5.2.5 Some “wrap-up” remarks on calculational proposition logic

5.3 Motivation

a. Proposition logic together with its extension, predicate logic, constitutes the most general and versatile basis for natural and mathematical reasoning.

b. The chosen approach includes three additional “features”

- Link to real number arithmetic as *binary algebra*
- Logic formulated *calculationally* as in engineering mathematics
- Formal calculation rules for *conditional expressions*

5.4 Binary Algebra

5.4.0 Binary Algebra as a restriction of minimax algebra

a. Minimax algebra: algebra of the *least upper bound* ($\curlyvee$) and *greatest lower bound* ($\curlywedge$) operators over $\mathbb{R}' := \mathbb{R} \cup \{-\infty, +\infty\}$. One definition is

$$\boxed{a \curlyvee b \leq c \;\equiv\; a \leq c \wedge b \leq c \quad \text{and} \quad c \leq a \curlywedge b \;\equiv\; c \leq a \wedge c \leq b}$$

Here $\leq$ is a total ordering, yielding an explicit form $\boxed{a \curlywedge b = (b \leq a)\,?\,b \nmid a}$ and laws that can be taken as alternative definitions

$$\boxed{c \leq a \curlyvee b \;\equiv\; c \leq a \vee c \leq b \quad \text{and} \quad a \curlyvee b \leq c \;\equiv\; a \leq c \wedge b \leq c}$$

b. Typical properties/laws: (derivable by high school algebra, duality saving work)

- Laws among the $\curlyvee$ and $\curlywedge$ operators: commutativity $a \curlyvee b = b \curlyvee a$, associativity $a \curlyvee (b \curlyvee c) = (a \curlyvee b) \curlyvee c$, distributivity $a \curlyvee (b \curlywedge c) = (a \curlyvee b) \curlywedge (a \curlyvee c)$, monotonicity $a \leq b \Rightarrow a \curlyvee c \leq b \curlyvee c$ and so on, plus their duals.
- Combined with other arithmetic operators: rich algebra of laws, e.g., distributivity: $a + (b \curlyvee c) = (a+b) \curlyvee (a+c)$ and $a - (b \curlyvee c) = (a-b) \curlywedge (a-c)$.

c. Binary algebra: algebra of $\vee$ and $\wedge$ as restrictions of $⩛$ and $⩚$ to $\mathbb{B} := \{0, 1\}$.

- Illustration for the 16 functions from $\mathbb{B}^2$ to $\mathbb{B}$:

x, y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0,0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0,1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1,0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1,1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$\mathbb{B}$		$\barvee$	$<$		$>$		$\not\equiv$	$\barwedge$	$\wedge$	$\equiv$	$\rangle\!\rangle$	$\Rightarrow$	$\langle\!\langle$	$\Leftarrow$	$\vee$	
$\mathbb{R}'$			$<$		$>$		$\neq$		$⩚$	$=$	$\rangle\!\rangle$	$\leq$	$\langle\!\langle$	$\geq$	$⩛$	

- Remark: $\equiv$ is just the restriction of $=$ to booleans, BUT:
  Many advantages in keeping $\equiv$ as a separate operator: fewer parentheses ($\equiv$ lowest precedence), highlighting associativity of $\equiv$ (not shared by $=$).

- All laws of minimax algebra particularize to laws over $\mathbb{B}$, for instance, $a ⩛ b \leq c \;\equiv\; a \leq c \wedge b \leq c$ and $c \leq a ⩚ b \;\equiv\; c \leq a \wedge c \leq b$ yield

$$\boxed{a \vee b \Rightarrow c \;\equiv\; (a \Rightarrow c) \wedge (b \Rightarrow c) \quad \text{and} \quad c \Rightarrow a \wedge b \;\equiv\; (c \Rightarrow a) \wedge (c \Rightarrow b)}$$

5.4.1 Relation to various other algebras and calculi

Here are some examples, that also justify explain our preference for $\mathbb{B} := \{0, 1\}$.

- Fuzzy logic, usually defined on the interval $[0, 1]$. Basic operators are restrictions of $⩛$ and $⩚$ to $[0, 1]$. We define fuzzy predicates on a set X as functions from X to $[0, 1]$, with ordinary predicates (to $\{0, 1\}$) a simple limiting case.

- Ordinary arithmetic, with $\{0, 1\}$ embedded in numbers without separate mapping. This proves especially useful in software specification and in word problems (examples later).

- Combinatorics, where a *characteristic function* "$C_P = 1$ if $P\,x$, 0 otherwise" is often introduced (admitting a regrettable design decision?) when calculations threaten to become unwieldy. The choice $\{0, 1\}$ obviates this.

- Modulo 2 arithmetic (as a Boolean ring or Galois field), as in coding theory. The associativity of $\equiv$ is counterintuitive for "logical equivalence" with truth values, but is directly clear from the equality $(a \equiv b) = (a \oplus b) \oplus 1$, which links it to modulo-2 arithmetic, where associativity of $\oplus$ is quite intuitive.

5.5 Axiomatic proposition calculus

5.5.0 Syntax and conventions

a. The language follows the syntax of simple expressions

$$\boxed{\begin{array}{lcl} \mathit{proposition} & ::= & \mathit{variable} \mid \mathit{constant} \mid \mathit{application} \\ \mathit{application} & ::= & \underline{(}\,\mathit{cop}_1\ \mathit{proposition}\underline{)} \mid \underline{(}\mathit{proposition}\ \mathit{cop}_2\ \mathit{proposition}\underline{)} \end{array}}$$

b. Variables are chosen near the end of the alphabet, e.g., x, y, z.
Constants and the operators in applications will be introduced one at a time.
Lowercase letters around p, q, r are metavariables standing for propositions.

c. We start with implication ($\underline{\Rightarrow}$) as the *only* propositional operator.

In $p \Rightarrow q$, we call p the *antecedent* and q the *consequent*. The rules for reducing parentheses in expressions with $\Rightarrow$ are the following.

- Outer parentheses may always be omitted.
- $x \Rightarrow y \Rightarrow z$ stands for $x \Rightarrow (y \Rightarrow z)$ (*right associativity* convention).
  Warning: do not read $x \Rightarrow y \Rightarrow z$ as $(x \Rightarrow y) \Rightarrow z$.

5.5.1 Implication: axioms, formal deduction: from classical to calculational

a. Inference rule, axioms and deduction

- The rules for implication (recall: instantiation is *strict*)

Inference rule, Instantiation of theorems: $\dfrac{p}{p[v := q]}$ (INS)

Inference rule, Modus Ponens: $\dfrac{p \Rightarrow q \, , \, p}{q}$ (MP)

Axioms, Weakening: $x \Rightarrow y \Rightarrow x$ (W⇒)

(left) Distributivity: $(x \Rightarrow y \Rightarrow z) \Rightarrow (x \Rightarrow y) \Rightarrow (x \Rightarrow z)$ (D⇒)

- Their use in deduction: if $\mathcal{H}$ is a collection of propositions (*hypotheses*), we say that q is a *consequence* of $\mathcal{H}$, written $\mathcal{H} \vdash q$, if q is either

  (i) an axiom, or (ii) a proposition in $\mathcal{H}$, or

  (iii) the conclusion of MP where the premisses are consequences of $\mathcal{H}$.

  If $\mathcal{H}$ is empty, we write $\vdash q$ for $\mathcal{H} \vdash q$, and q is a *theorem*. Being a theorem or a consequence of the hypotheses is called the *status* of a proposition. A (formal) *proof* or *deduction* is a record of how $\mathcal{H} \vdash q$ is established.

b. Two classical formats for deduction

Chosen example: the theorem of *reflexivity* (R⇒), namely $x \Rightarrow x$

Proof in *statement list* style (the numbers are for reference)

0.	INS D⇒	$(x \Rightarrow (x \Rightarrow x) \Rightarrow x) \Rightarrow (x \Rightarrow x \Rightarrow x) \Rightarrow (x \Rightarrow x)$
1.	INS W⇒	$x \Rightarrow (x \Rightarrow x) \Rightarrow x$
2.	MP 0, 1	$(x \Rightarrow x \Rightarrow x) \Rightarrow (x \Rightarrow x)$
3.	INS W⇒	$x \Rightarrow x \Rightarrow x$
4.	MP 2, 3	$x \Rightarrow x$

Proof in *sequent* style

$$\dfrac{\dfrac{\dfrac{(x \Rightarrow y \Rightarrow z) \Rightarrow (x \Rightarrow y) \Rightarrow x \Rightarrow z}{(x \Rightarrow (x \Rightarrow x) \Rightarrow x) \Rightarrow (x \Rightarrow x \Rightarrow x) \Rightarrow x \Rightarrow x}\,\mathsf{I} \quad \dfrac{x \Rightarrow y \Rightarrow x}{x \Rightarrow (x \Rightarrow x) \Rightarrow x}\,\mathsf{I}}{(x \Rightarrow x \Rightarrow x) \Rightarrow x \Rightarrow x}\,\mathsf{M} \quad \dfrac{x \Rightarrow y \Rightarrow x}{x \Rightarrow x \Rightarrow x}\,\mathsf{I}}{x \Rightarrow x}\,\mathsf{M}$$

c. Casting classical deduction in calculational style

- Remark: only as a temporary stepping stone
- Convention for chaining steps with modus ponens (which has 2 premisses): make explicit which of $p \Rightarrow q$ or p lies along the main line of reasoning

$$\boxed{\begin{array}{lll@{\qquad}lll} & \langle\text{Deduction steps for } p\rangle & p & & \langle\text{Steps for } p \Rightarrow q\rangle & p \Rightarrow q \\ \Downarrow & \langle\text{Justification for } p \Rightarrow q\rangle & q & \times & \langle\text{Justification for } p\rangle & q \end{array}}$$

The justification in $\langle\ \rangle$ is either a hypothesis or a theorem instance.

- Example: proving reflexivity $x \Rightarrow x$

$$\boxed{\begin{array}{lll} & \langle\text{W}{\Rightarrow}\rangle & x \Rightarrow (x \Rightarrow x) \Rightarrow x \\ \Downarrow & \langle\text{D}{\Rightarrow}\rangle & (x \Rightarrow x \Rightarrow x) \Rightarrow x \Rightarrow x \\ \times & \langle\text{W}{\Rightarrow}\rangle & x \Rightarrow x \end{array}}$$

- Next step: replace pseudo-calculational "labels" ⇓ and × by operators *in* the language (here ⇒), as justified next.

d. Making deduction with implication fully calculational

- Metatheorem, *Transitivity* (T⇒): $\boxed{p \Rightarrow q \ ,\ q \Rightarrow r \ \vdash\ p \Rightarrow r}$. Proof:

$$\begin{array}{lll}
 & \langle \mathsf{W}{\Rightarrow} \rangle & (q \Rightarrow r) \Rightarrow p \Rightarrow q \Rightarrow r \\
\times & \langle q \Rightarrow r \rangle & p \Rightarrow q \Rightarrow r \\
\Downarrow & \langle \mathsf{D}{\Rightarrow} \rangle & (p \Rightarrow q) \Rightarrow (p \Rightarrow r) \\
\times & \langle p \Rightarrow q \rangle & p \Rightarrow r
\end{array}$$

This subsumes ⇓-steps since it justifies chaining of the form

$$\boxed{\begin{array}{lll}
p & \Rightarrow \langle \text{Justification for } p \Rightarrow q \rangle & q \\
 & \Rightarrow \langle \text{Justification for } q \Rightarrow r \rangle & r,
\end{array}}$$

Example: $x \Rightarrow y \ \Rightarrow\!\langle \mathsf{W}{\Rightarrow} \rangle\ z \Rightarrow x \Rightarrow y \ \Rightarrow\!\langle \mathsf{D}{\Rightarrow} \rangle\ (z \Rightarrow x) \Rightarrow (z \Rightarrow y)$,
proving *(right) Monotonicity* (M⇒): $(x \Rightarrow y) \Rightarrow (z \Rightarrow x) \Rightarrow (z \Rightarrow y)$.

- Theorem, *modus ponens as a formula* (P⇒): $\boxed{x \Rightarrow (x \Rightarrow y) \Rightarrow y}$
This subsumes ×-steps since it justifies writing

$$\boxed{p \Rightarrow q \quad \Rightarrow \langle \text{Justification for } p \rangle \quad q}$$

e. Some representative calculation rules

Rules that prove often useful in practice are given a suggestive name.
This is a valuable mnemonic aid for becoming familiar with them.
The terminology is due to the "calculational school" (Dijkstra, Gries e.a.).

Name (rules for ⇒)	Formula	Ref.
Weakening	$x \Rightarrow y \Rightarrow x$	W⇒
Distributivity (left)	$(x \Rightarrow y \Rightarrow z) \Rightarrow (x \Rightarrow y) \Rightarrow (x \Rightarrow z)$	D⇒
Reflexivity	$x \Rightarrow x$	R⇒
Right monotonicity	$(x \Rightarrow y) \Rightarrow (z \Rightarrow x) \Rightarrow (z \Rightarrow y)$	RM⇒
MP as a formula	$x \Rightarrow (x \Rightarrow y) \Rightarrow y$	MP⇒
Shunting	$(x \Rightarrow y \Rightarrow z) \Rightarrow x \Rightarrow y \Rightarrow z$	SH⇒
Left Antimonotonicity	$(x \Rightarrow y) \Rightarrow (y \Rightarrow z) \Rightarrow (x \Rightarrow z)$	LA⇒

Metatheorems (following from RM and SA respectively)

Name of metatheorem	Formulation	Ref.
Weakening the Consequent	$p \Rightarrow q \vdash (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$	(WC)
Strengthening the Antecedent	$p \Rightarrow q \vdash (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$	(SA)

f. A first proof shortcut: the deduction (meta)theorem

- Motivation: common practice in informal reasoning:
  if asked to prove $p \Rightarrow q$, one assumes p (hypothesis) and deduces q.
  The proof so given is a demonstration for $p \vdash q$, *not* one for $\vdash p \Rightarrow q$.
  A proof for $p \Rightarrow q$ is different and usually much longer.
- Significance of the deduction theorem: justification of this shortcut
  - A demonstration for $p \vdash q$ implies existence of one for $\vdash p \Rightarrow q$.
  - More: the proof of the deduction theorem is *constructive*: an *algorithm* for transforming a demonstration for $p \vdash q$ into a one for $p \Rightarrow q$.

  Proving $p \Rightarrow q$ by deriving q from p is called *assuming the antecedent*.
- Formal statement

$$\boxed{\textsc{Deduction theorem:}\quad \text{If } \mathcal{H} \,\&\, p \vdash q \text{ then } \mathcal{H} \vdash p \Rightarrow q}$$

  - Convention: $\mathcal{H} \,\&\, p$ is the collection $\mathcal{H}$ of hypotheses augmented by p.
  - Converse of the deduction theorem: If $\mathcal{H} \vdash p \Rightarrow q$ then $\mathcal{H} \,\&\, p \vdash q$.
    Proof: a deduction for $\mathcal{H} \vdash p \Rightarrow q$ is a deduction for $\mathcal{H} \,\&\, p \vdash p \Rightarrow q$.
    Adding the single step $\times \langle p \rangle\, q$ yields a deduction for $\mathcal{H} \,\&\, p \vdash q$.

g. Introducing the truth constant

- Strictly speaking, no truth constant is needed: any theorem can serve.

  Lemma: theorems as right zero and left identity for "$\Rightarrow$".

  Any theorem t has the following properties (exercise).

  - Right zero: $x \Rightarrow t$. Also: $t \Rightarrow (x \Rightarrow t)$ and $(x \Rightarrow t) \Rightarrow t$.
  - Left identity: $x \Rightarrow (t \Rightarrow x)$ and $(t \Rightarrow x) \Rightarrow x$.

- In view of the algebraic use of $\Rightarrow$, we introduce the constant $\underline{1}$ by

  $$\boxed{\textsc{Axiom, the truth constant:}\quad 1}$$

  Theorem: 1 as right zero and left identity for "$\Rightarrow$"

  - Right zero: $x \Rightarrow 1$. Also: $1 \Rightarrow (x \Rightarrow 1)$ and $(x \Rightarrow 1) \Rightarrow 1$.
  - Left identity: $x \Rightarrow (1 \Rightarrow x)$ and $(1 \Rightarrow x) \Rightarrow x$.

  Proof: direct consequences of the preceding lemma.

Thus far, we addressed many technicalities to establish the formal proof style. For the remainder, we leave the technicalities as exercises.

5.5.2 Negation: making proposition algebra complete

a. Syntax and axiom: the negation operator $\underline{\neg}$ is a 1-place function symbol, with

$$\boxed{\textsc{Axiom, Contrapositive:}\quad (\neg\, x \Rightarrow \neg\, y) \Rightarrow y \Rightarrow x \qquad \text{(CP}\Rightarrow\text{)}}$$

b. Some initial theorems and their proofs

- Theorem, Contradictory antecedents: $\boxed{\neg\, x \Rightarrow x \Rightarrow y \qquad \text{(CA}\Rightarrow\text{)}}$
  Proof: on sight. Corollary: (meta), contradictory hypotheses: $p\,, \neg\, p \vdash q$
- Theorem, Skew idempotency of "⇒": $\boxed{(\neg\, x \Rightarrow x) \Rightarrow x \qquad \text{(SI}\Rightarrow\text{)}}$
  Proof: subtle; start by instantiating (CA⇒) as $\neg\, x \Rightarrow x \Rightarrow \neg\,(\neg\, x \Rightarrow x)$.

$$\begin{array}{lcl}
\multicolumn{3}{l}{\neg\, x \Rightarrow x \Rightarrow \neg\,(\neg\, x \Rightarrow x)} \\
\Rightarrow & \langle \text{LD}\Rightarrow \rangle & (\neg\, x \Rightarrow x) \Rightarrow \neg\, x \Rightarrow \neg\,(\neg\, x \Rightarrow x) \\
\Rightarrow & \langle \text{WC by CP}\Rightarrow \rangle & (\neg\, x \Rightarrow x) \Rightarrow (\neg\, x \Rightarrow x) \Rightarrow x \\
\Rightarrow & \langle \text{AB}\Rightarrow \rangle & (\neg\, x \Rightarrow x) \Rightarrow x
\end{array}$$

After this, all further theorems are relatively straightforward.

c. Some important calculation rules for negation

- Convention: $(\neg^0 p)$ stands for p and $(\neg^{n+1} p)$ for $\neg(\neg^n p)$.
  This avoids accumulation of parentheses as in $\neg(\neg(\neg p))$.
- Most frequently useful rules

Name (rules for $\neg/\Rightarrow$)	Formula	Ref.
Contrapositive	$(\neg x \Rightarrow \neg y) \Rightarrow y \Rightarrow x$	CP$\Rightarrow$
Contradictory antecedents	$\neg x \Rightarrow x \Rightarrow y$	CA$\Rightarrow$
Skew idempotency of "$\Rightarrow$"	$(\neg x \Rightarrow x) \Rightarrow x$	SI$\Rightarrow$
Double negation	$\neg^2 x \Rightarrow x$ and $x \Rightarrow \neg^2 x$	DN
Contrapositive Reversed	$(x \Rightarrow y) \Rightarrow (\neg y \Rightarrow \neg x)$	CPR
Contrapositive Strengthened	$(\neg x \Rightarrow \neg y) \Rightarrow (\neg x \Rightarrow y) \Rightarrow x$	CPS
Dilemma	$(\neg x \Rightarrow y) \Rightarrow (x \Rightarrow y) \Rightarrow y$	DIL

d. Introducing the falsehood constant

$\boxed{\text{Axiom, the falsehood constant:} \quad \neg\, 0}$

Some simple properties:

- $0 \Rightarrow x$
- $(x \Rightarrow 0) \Rightarrow \neg\, x$ and $\neg\, x \Rightarrow (x \Rightarrow 0)$
- $1 \Rightarrow \neg\, 0$ and $\neg\, 1 \Rightarrow 0$

5.5.3 Quick calculation rules and proof techniques

a. Case analysis

- Lemma, Binary cases: for any proposition p and variable v,
  - (0-case) $\vdash \neg\, v \Rightarrow p[^v_0 \Rightarrow p$ and $\vdash \neg\, v \Rightarrow p \Rightarrow p[^v_0$
  - (1-case) $\vdash v \Rightarrow p[^v_1 \Rightarrow p$ and $\vdash v \Rightarrow p \Rightarrow p[^v_1$

  Proof: structural induction (discussed later)

- Lemma, Case analysis: for any proposition p and variable v,

  $\vdash p[^v_0 \Rightarrow p[^v_1 \Rightarrow p$ and, equivalently, $p[^v_0 \,\&\, p[^v_1 \;\vdash\; p$

  Significance: to prove or verify p, it suffices proving $p[^v_0$ and $p[^v_1$.
  After a little practice, this can be done by inspectionor head calculation.

b. An implicative variant of a theorem (presented later) attributed to Shannon:
Shannon expansion with implication: for any proposition p and variable v,

Accumulation: $\vdash (\neg\, v \Rightarrow p[^v_0) \Rightarrow (v \Rightarrow p[^v_1) \Rightarrow p$

Weakening: $\vdash p \Rightarrow \neg\, v \Rightarrow p[^v_0$ and $\vdash p \Rightarrow v \Rightarrow p[^v_1$

c. Summary of quick proof techniques

- *Assuming the antecedent*: to prove $p \Rightarrow q$, assume p and deduce q. Warning: p is normally not a theorem: hence may *not* be instantiated.
- *Case analysis*: to prove p, prove $p[^v_1$ and $p[^v_0$. Note: choosing v well can significantly reduce the effort. The method can be used recursively.
- *Proof by negation*: to prove p, prove $\neg p \Rightarrow 0$.
  Classical variant: *'reductio ad absurdum'*: to prove $\neg p$, prove $p \Rightarrow 0$.
- *Proof by contradiction*: to prove p, assume $\neg p$ and deduce p.
  Justification: the deduction theorem, $(\neg p \Rightarrow p) \Rightarrow p$, and (MP).
  Warning A degraded *'junkyard'* version to be avoided: to prove p, some people assume $\neg p$ and prove p without even using the assumption. Many uniqueness proofs fall into this category.
  Introducing assumptions that are not used makes an intellectual junkyard.
- *Proof by contrapositive*: to prove $p \Rightarrow q$, prove $\neg q \Rightarrow \neg p$.

The justification of most of these proof techniques helps finding 'direct' proof. It may also indicate the way to shorter proofs.

5.5.4 Brief summary of derived operators and calculation rules

a. Logical equivalence, with symbol $\equiv$ (lowest precedence) and Axioms:

Antisymmetry of $\Rightarrow$:	$(x \Rightarrow y) \Rightarrow (y \Rightarrow x) \Rightarrow (x \equiv y)$	(AS$\Rightarrow$)
Weakening of $\equiv$:	$(x \equiv y) \Rightarrow x \Rightarrow y$ and $(x \equiv y) \Rightarrow y \Rightarrow x$	(W$\equiv$)

Theorem Given p, q, let s be specified by $p \Rightarrow q \Rightarrow s$ and $s \Rightarrow p$ and $s \Rightarrow q$. Then $s := \neg\,(p \Rightarrow \neg\, q)$ satisfies the spec, and any solution s' satisfies $s \equiv s'$.

Main property of $\equiv$: logical equivalence is *propositional equality*

- It is an equivalence relation

Reflexivity:	$x \equiv x$	(R$\equiv$)
Symmetry:	$(x \equiv y) \Rightarrow (y \equiv x)$	(S$\equiv$)
Transitivity:	$(x \equiv y) \Rightarrow (y \equiv z) \Rightarrow (x \equiv z)$	(T$\equiv$)

- It obeys Leibniz's principle:

Leibniz: $(x \equiv y) \Rightarrow (p[^{v}_{x} \equiv p[^{v}_{y})$ (L$\equiv$)

Some earlier theorems in equational form

Name	Formula	Ref.
Shunting with "$\Rightarrow$"	$x \Rightarrow y \Rightarrow z \equiv y \Rightarrow x \Rightarrow z$	ESH$\Rightarrow$
Contrapositive	$(x \Rightarrow y) \equiv (\neg y \Rightarrow \neg x)$	ECP$\Rightarrow$
Left identity for "$\Rightarrow$"	$1 \Rightarrow x \equiv x$	LE$\Rightarrow$
Right negator for "$\Rightarrow$"	$x \Rightarrow 0 \equiv \neg x$	RN$\Rightarrow$
Identity for "$\equiv$"	$(1 \equiv x) \equiv x$	E$\equiv$
Negator for "$\equiv$"	$(0 \equiv x) \equiv \neg x$	N$\equiv$
Double negation (equationally)	$\neg^2 x \equiv x$	EDN
Negation of the constants	$\neg 0 \equiv 1$ and $\neg 1 \equiv 0$	

Some new theorems about $\equiv$

Semidistributivity $\neg/\equiv$	$\neg (x \equiv y) \equiv (\neg x \equiv y)$	SD$\neg/\Rightarrow$
Associativity of $\equiv$	$((x \equiv y) \equiv z) \equiv (x \equiv (y \equiv z))$	A$\equiv$
Shannon by equivalence	$p \equiv \neg x \Rightarrow p[^{x}_{0} \equiv x \Rightarrow p[^{x}_{1}$	
Left distributivity $\Rightarrow/\equiv$	$z \Rightarrow (x \equiv y) \equiv z \Rightarrow x \equiv z \Rightarrow y$	LD$\Rightarrow/\equiv$
Right skew distrib. $\Rightarrow/\equiv$	$(x \equiv y) \Rightarrow z \equiv x \Rightarrow z \equiv \neg y \Rightarrow z$	SD$\Rightarrow/\equiv$

b. Propositional inequality with symbol $\not\equiv$ and axiom

AXIOM, PROPOSITIONAL INEQUALITY: $(x \not\equiv y) \equiv \neg (x \equiv y)$

Via the properties of $\equiv$, one quickly deduces the following algebraic laws

Name	Formula	Ref.
Irreflexivity	$\neg (x \not\equiv x)$	IR$\not\equiv$
Symmetry	$(x \not\equiv y) \equiv (y \not\equiv x)$	S$\not\equiv$
Associativity:	$((x \not\equiv y) \not\equiv z) \equiv (x \not\equiv (y \not\equiv z))$	A$\not\equiv$
Mutual associativity	$((x \not\equiv y) \equiv z) \equiv (x \not\equiv (y \equiv z))$	MA$\not\equiv/\equiv$
Mutual interchangeability:	$x \not\equiv y \equiv z \equiv x \equiv y \not\equiv z$	MI$\not\equiv/\equiv$

A formula with (only) $\equiv$ and $\not\equiv$ depends only on even/odd number of occurrences.

c. Disjunction ($\vee$) and conjunction ($\wedge$)

- These operators have highest precedence. An equational axiomatization is:

$$\boxed{\begin{array}{ll} \textsc{Axiom, Disjunction:} & x \vee y \equiv \neg\, x \Rightarrow y \\ \textsc{Axiom, Conjunction:} & x \wedge y \equiv \neg\,(x \Rightarrow \neg\, y) \end{array}}$$

- An immediate consequence (using EDN) is the following theorem (De Morgan)

$$\boxed{\neg\,(x \vee y) \equiv \neg\, x \wedge \neg\, y \qquad \neg\,(x \wedge y) \equiv \neg\, x \wedge \neg\, y \qquad \text{(DM)}}$$

- There are dozens of other useful theorems about conjunction and disjunction. Most are well-known, being formally identical to those from switching algebra. Example: proposition calculus constitutes a Boolean algebra w.r.t. $\vee$ and $\wedge$.

- Others are unknown in switching algebra but very useful in calculation, e.g.,

$$\boxed{\textsc{Theorem, Shunting } \wedge \textsc{ and } \Rightarrow\text{:} \quad x \wedge y \Rightarrow z \equiv x \Rightarrow y \Rightarrow z \quad (\text{SH}\wedge)}$$

Caution: with emphasizing parentheses, $((x \wedge y) \Rightarrow z) \equiv (x \Rightarrow (y \Rightarrow z))$.

5.5.5 Some “wrap-up” remarks on calculational proposition logic

a. Chaining calculation steps In chained calculations, we can mix $\equiv$ with $\Rightarrow$:

$$\boxed{\begin{array}{llll} r & \Rightarrow & \langle r \Rightarrow p\rangle & p \\ & \equiv & \langle p \equiv q\rangle & q \end{array} \qquad\qquad \begin{array}{llll} p & \equiv & \langle p \equiv q\rangle & q \\ & \Rightarrow & \langle q \Rightarrow s\rangle & s \end{array}}$$

Convention, an operator used as a calculation link has lowest precedence.
In principle, no ambiguity since recognizable by the presence of $\langle\ \rangle$.
In practice: a source of errors for the unwary. Example:

$$\boxed{p \Rightarrow q \quad \Rightarrow \quad \langle p \Rightarrow q \Rightarrow r\rangle \quad r \qquad \text{Wrong!}}$$

or, more insidiously,

$$\boxed{s \Rightarrow p \Rightarrow q \quad \Rightarrow \quad \langle \text{WC by } \ p \Rightarrow q \Rightarrow r\rangle \quad s \Rightarrow r. \qquad \text{Wrong!}}$$

b. From propositional equality to equality in general

- The laws for general equality (re)formulated as axioms using implication

Reflexivity:	$x = x$	(R=)
Symmetry:	$x = y \Rightarrow y = x$	(S=)
Transitivity:	$x = y \Rightarrow y = z \Rightarrow x = z$	(T=)
"Leibniz":	$x = y \Rightarrow e[{}^z_x = e[{}^z_y$	(L=)

- Propositional equality as equality

$$\boxed{x = y \;\equiv\; x \;\equiv\; y}$$

In general context: assumes x and y boolean.

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

6 Binary algebra and proposition calculus: applications

6.0 Calculating with conditional expressions, formally

6.0.0 Outline of the enabling context

6.0.1 Conditionals as binary indexing: definition and calculation rules

6.1 Formal description of combinational circuits

6.1.0 A minimal formal syntax for circuit description

6.1.1 System semantics

6.1.2 Specifications and realizations

6.2 Formalizing and solving unparametrized word problems

6.2.0 Formalizing simple word problems

6.2.1 Solving simple word problems

6.2.2 Additional examples

6.0 Calculating with conditional expressions, formally

6.0.0 Outline of the enabling context

Design decisions making the approach described here possible.

a. Defining tuples as functions that take natural numbers as arguments in the sense that, for instance

$$\boxed{(a,b,c)\,0 = a \quad \text{and} \quad (a,b,c)\,1 = b \quad \text{and} \quad (a,b,c)\,2 = c}$$

b. Embedding propositional calculus in arithmetic, treating the propositional constants 0 and 1 as numbers.

c. Using the generic functionals *function composition* ($\circ$) and *transposition* ($—^{\mathsf{U}}$):

$$\boxed{(f \circ g)\,x = f\,(g\,x) \quad \text{and} \quad f^{\mathsf{U}} y\,x = f\,x\,y}$$

Remark: we ignore types for the time being (types give rise to later variants). Observe the analogy with the lambda combinators:

$$\mathbf{C} := \lambda fxy.f(xy) \quad \text{and} \quad \mathbf{T} := \lambda fxy.fyx$$

6.0.1 Conditionals as binary indexing: definition and calculation rules

a. Syntax and axiom: the general form is $c\,?\,b \dagger a$; furthermore,

$$\boxed{\textsc{Axiom for conditionals:}\quad c\,?\,b \dagger a \;\;=\;\; (a, b)\,c}$$

b. Deriving calculation rules using the distributivity laws for —^{U} and $\circ$:

$$(f, g, h)^{\mathsf{U}} x = f\,x, g\,x, h\,x \quad\text{and}\quad f \circ (x, y, z) = f\,x, f\,y, f\,z.$$

Theorem, distributivity laws for conditionals:

$$\boxed{(c\,?\,f \dagger g)\,x = c\,?\,f\,x \dagger g\,x \quad\text{and}\quad f\,(c\,?\,x \dagger y) = c\,?\,f\,x \dagger f\,y.}$$

Proof: given for one variant only, the other being very similar.

$$\begin{aligned}
(c\,?\,f \dagger g)\,x &= \langle\text{Def. conditional}\rangle && (g, f)\,c\,x \\
&= \langle\text{Def. transposition}\rangle && (g, f)^{\mathsf{U}} x\,c \\
&= \langle\text{Distributivity } \text{—}^{\mathsf{U}}\rangle && (g\,x, f\,x)\,c \\
&= \langle\text{Def. conditional}\rangle && c\,?\,f\,x \dagger g\,x.
\end{aligned}$$

c. Particular case where a and b (and, of course, c) are all binary:

$$\boxed{c\,?\,b \nmid a \quad\equiv\quad (c \Rightarrow b) \wedge (\neg\, c \Rightarrow a)}$$

Proof:

$$\begin{array}{rcll}
c\,?\,b \nmid a & \equiv & \langle\text{Def. cond.}\rangle & (a,b)\,c \\
& \equiv & \langle\text{Shannon}\rangle & (c \wedge (a,b)\,1) \vee (\neg\, c \wedge (a,b)\,0) \\
& \equiv & \langle\text{Def. tuples}\rangle & (c \wedge b) \vee (\neg\, c \wedge a) \\
& \equiv & \langle\text{Binary alg.}\rangle & (\neg\, c \vee b) \wedge (c \vee a) \\
& \equiv & \langle\text{Defin. } \Rightarrow\rangle & (c \Rightarrow b) \wedge (\neg\, c \Rightarrow a)
\end{array}$$

d. Finally, since predicates are functions and $(z =)$ is a predicate,

$$\boxed{z = (c\,?\,x \nmid y) \quad\equiv\quad (c \Rightarrow z = x) \wedge (\neg\, c \Rightarrow z = y)}$$

Proof:

$$\begin{array}{rcll}
z = (c\,?\,x \nmid y) & \equiv & \langle\text{Distributivity}\rangle & c\,?\,(z = x) \nmid (z = y) \\
& \equiv & \langle\text{Preceding law}\rangle & (c \Rightarrow z = x) \wedge (\neg\, c \Rightarrow z = y)
\end{array}$$

These laws are all one ever needs for working with conditionals!

6.1 Formal description of combinational circuits

6.1.0 A minimal formal syntax for circuit description

a. *Language of circuit descriptions* **CD** The sentences are nonempty lists of *statements* of the form $v = e$, separated by semicolons (;). To such a *primary list* can be added an optional *secondary* list (same syntax) preceded by **where**.

Here v is an *identifier* and e an *expression* with *prefix operators* (AND, OR, etc.) with one or more operands, and single-operand prefix operators (NOT).

Abbreviation conventions (variadic notation)

$$\begin{array}{rll} a \wedge b \wedge c & \text{abbreviates} & AND\,(a, b, c), \\ a \wedge (b \wedge c) & \text{abbreviates} & AND\,(a, AND\,(b, c)), \\ \neg\,(a \wedge \neg\, b) & \text{abbreviates} & NOT\,(AND\,(a, NOT\, b)). \end{array}$$

b. Examples

$$z = (x \wedge \neg\, s) \vee (y \wedge s)\ ;\ s = u \wedge v$$

$$c = a \wedge b\ ;\ s = \theta \vee \pi\ \ \mathbf{where}\ \ \theta = a \wedge \phi\ ;\ \pi = b \wedge \omega\ ;\ \phi = \neg\, b\ ;\ \omega = \neg\, a$$

6.1.1 System semantics

a. Principle: multiple interpretations for the same formal description

b. *Structural semantics* expresses connectivity

- Operators “are” devices
- Identifiers “are” nodes; those at the l.h.s of $=$ are connected to the output of the device at the r.h.s., and are circuit outputs if in the priomary list
- Outputs of applications in argument positions are anonymous nodes.

Example

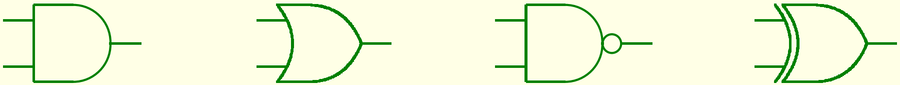
$$\boxed{c = a \wedge b\ ;\ s = \theta \vee \pi\ \ \mathbf{where}\ \ \theta = a \wedge \phi\ ;\ \pi = b \wedge \neg\, a\ ;\ \phi = \neg\, b}$$

Circuit outputs: c and s. Internal nodes: θ, π, ϕ. Anonymous: output of $\neg\, a$.

c. *Behavioural semantics* expresses behaviour. Several kinds, e.g.,

- Static behaviour: simply interpret the circuit as propositional equations
- Dynamic behaviour: again various kinds (beyond proposition calculus).

Example of various interpretations: for the operators AND, OR, $NAND$, XOR, we show the structural and the the static behavioural interpretations



AND			OR			$NAND$			XOR		
x	y	$x \wedge y$	x	y	$x \vee y$	x	y	$x \barwedge y$	x	y	$x \oplus y$
0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	1	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0	1	1	0

More on system semantics (dynamic behaviour, unidirectional analog circuits)

- R. T. Boute, "System semantics and formal circuit description", *IEEE Transactions on Circuits and Systems, CAS–33*, No. 12, pp. 1219–1231, (Dec 1986)
- R. T. Boute, "System Semantics: Principles, Applications and Implementation", *ACM Trans. Prog. Lang. and Syst., 10*, No. 1, pp. 118–155, (1988)

6.1.2 Specifications and realizations

a. Convention: $\widehat{C}$ is the (static) behavioural interpretation of circuit C.

b. Specification: informally, a specification is a boolean expression S characterizing all acceptable states in the (static) behaviour.
Formally: circuit C satisfies specification S iff $\widehat{C} \Rightarrow S$.

c. Examples: consider the following four specifications.

$$\begin{array}{ll} \text{(a)} & (z \equiv x \wedge y) \vee (z \oplus x) \\ \text{(b)} & z \Rightarrow x \\ \text{(c)} & (z \equiv x) \vee (z \equiv \neg x) \\ \text{(d)} & (z \equiv x) \wedge (z \equiv \neg x) \end{array}$$

Then
(a) can be realized by several circuits, including $z = x \wedge y$ and $z = \neg x$.
(b) can be realized by several circuits, including $z = 0$ and $z = x$.
(c) can be realized by any circuit.
(d) can be realized by no circuit.

6.2 Calculating with conditional expressions, formally

6.2.0 Formalizing simple word problems

a. Terminology Word problem: a problem stated in words to be solved by logic. Unparametrized: pertaining to a finite, fixed number of objects.

b. Examples

- Pure logical examples:

  This statement is false

  Here is another one:

  Exactly one of these two statements is false.
  Both of these two statements are false.

- A variant of a riddle from Shakespeare's "The Merchant of Venice" [Gries]

  The key to Portia's heart is hidden in one of two caskets: a golden one bearing the inscription "The key is not in here", and a silver one with the inscription "Exactly one of the inscriptions on these caskets is true". Question: where is the key?

c. Formalization

- Principle (outline): replace atomic substatements by variables and translate the connectives into logical symbols according to the following table

if a then b	$a \Rightarrow b$
a only if b	$a \Rightarrow b$
a if b	$a \Leftarrow b$
a and b	$a \wedge b$
a or b	$a \vee b$
etc.	

  In statements referencing themselves or each other regarding truthfulness, referenced statements are substatements and hence designated by a variable, resulting in $x \equiv$ contents of x.

- Example A: "This statement is false" becomes $x \equiv \neg x$.
- Example B: the pair of statements "Exactly one of these two statements is true." and "Both of these two statements are true."
  becomes $(x \equiv x \oplus y) \wedge (y \equiv x \wedge y)$.

6.2.1 Solving simple word problems

a. Principle: consider the formalized information as equations, with unknowns as specified in the question of the problem statement.

Observation: solutions may be nonexistent or not unique, as in basic algebra:

0 solutions	1 solution	many solutions
$3 \cdot x + 2 \cdot y = 7$ $6 \cdot x + 4 \cdot y = 5$	$3 \cdot x + 2 \cdot y = 7$ $6 \cdot x + 2 \cdot y = 10$	$3 \cdot x + 2 \cdot y = 7$ $6 \cdot x + 4 \cdot y = 14$

b. Examples (for A and B the questions are implicitly: "What gives?")

- Example A: $x \equiv \neg x$ clearly has no solution since $\vdash x \equiv \neg x \equiv 0$.
- Example B: $(x \equiv x \oplus y) \wedge (y \equiv x \wedge y)$ is solved for x and y as follows.

$$\boxed{\begin{array}{lll} x \equiv x \oplus y & \equiv & \langle x \oplus y \equiv x \not\equiv y \rangle \quad x \equiv (x \not\equiv y) \\ & \equiv & \langle \text{Assoc.}, x \equiv x \rangle \quad 1 \not\equiv y \end{array}}$$

So $y \equiv 0$ regardless of x. Clearly, $y \equiv 0$ also satisfies $y \equiv x \wedge y$, again regardless of x. Hence there are 2 solutions: $x, y := 0, 0$ and $x, y := 1, 0$.

6.2.2 Additional Examples

- Example C: variant of a riddle from Shakespeare's "The Merchant of Venice"

> The key to Portia's heart is in one of two caskets: a golden one with the inscription "The key is not in here", and a silver one saying "Exactly one of the inscriptions is true". Question: where is the key?

Formalizing the information: assuming gc and sc represent the fact that the key is in the golden/silver casket, whereas g and s represent the inscriptions,

$$\boxed{(gc \oplus sc) \wedge (g \equiv \neg\, gc) \wedge (s \equiv g \oplus s).}$$

Solution: answering "Question: where is the key?" means finding gc and sc satisfying the equation. With the insight that $s \equiv g \oplus s$ carries the essence,

$$\boxed{\begin{array}{lll} s \equiv g \oplus s & \equiv & \langle \text{As xmpl C} \rangle \quad g \equiv 0 \\ & \equiv & \langle g \equiv \neg\, gc \rangle \quad gc \end{array}}$$

Checking whether this satisfies $gc \oplus sc$ is evident and also yields $\neg\, sc$. Hence the key should be in the golden casket.

- Example D (first of two problems due to Johnnson-Laird)

> One of the following assertions is true about a particular hand of cards, and one of them is false about the same hand of cards.
>
> If there is a king in the hand, then there is an ace in the hand.
>
> If there isn't a king in the hand, then there is an ace in the hand.
>
> What follows?

Formalizing the information Two equivalent forms, with evident conventions:

$$\boxed{(k \Rightarrow a) \oplus (\neg\, k \Rightarrow a) \quad \text{or, equivalently,} \quad (k \Rightarrow a) + (\neg\, k \Rightarrow a) = 1}$$

Solution: answering "What follows?" means finding all k and a satisfying the equation. Let us use the second form, $p := (k \Rightarrow a) + (\neg\, k \Rightarrow a) = 1$

$$\begin{array}{lclc}
p & \equiv & \langle\text{Shannon theorem}\rangle & (\neg\, a \wedge p[^a_0]) \vee (a \wedge p[^a_1]) \\
 & \equiv & \langle\text{Subst., rules} \Rightarrow\rangle & (\neg\, a \wedge \neg\, k + k = 1) \vee (a \wedge 1 + 1 = 1) \\
 & \equiv & \langle 1 + 1 \neq 1 \rangle & \neg\, a \wedge \neg\, k + k = 1 \\
 & \equiv & \langle \neg\, k + k = 1 \rangle & \neg\, a
\end{array}$$

Hence there are two solutions: $k, a := 0, 0$ and $k, a := 1, 0$

- Example E:

  Only one of the following is true about a particular hand of cards:
  There is a king in the hand, or an ace, or both.
  There is a queen in the hand, or an ace, or both.
  There is a jack in the hand, or a ten, or both.
  Is is possible that there is an ace in the hand?

  Formalizing the information: with self-explanatory conventions,

$$\boxed{(k \vee a) + (q \vee a) + (j \vee t) = 1}$$

  Solution: answering the question "Is is possible that there is an ace in the hand?" amounts to determining whether there are solutions with $a \equiv 1$.
  Applying Shannon's theorem to the equation yields $\neg\, a \wedge k + q + (j \vee t) = 1$.
  Even simpler: assuming $a = 1$ yields $1 + 1 + (j \vee t) = 1$, which is 0.

Problems D and E are due to Johnson-Laird and used in his research on the psychology of logic: `http://www.princeton.edu/~psych/PsychSite/fac_phil.html`

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

7 Sets and functions, equality

7.0 Motivation

7.1 Calculating with sets, formally

7.1.0 Rationale of the formulation

7.1.1 Equality for sets

7.1.2 Set expressions, operators and calculation rules

7.2 Calculating with functions, formally

7.2.0 Rationale of the formulation

7.2.1 Equality for functions

7.2.2 Function expressions, operators and calculation rules

7.3 Generic functionals, a sneak preview

7.3.0 Principle

7.3.1 Some generic functionals (only as needed for point-free predicate calculus)

7.0 Motivation

a. General (in the context of mathematics and computing)

- Thus far in this tutorial, formulations were either untyped (lambda calculus) or implicitly singly typed (simple algebras, proposition algebra)
- In the practice of mathematics, sets are ubiquitous
- In declarative formalisms, sets provide flxible typing
- Functions are perhaps the most powerful single concept in mathematics. Arguably also in computing: power/elegance of functional programming

b. Specific (in the context of a functional formalism, as considered here)

- Functions are a fundamental concept (not identified with sets of pairs)
- Sets are extremely useful for defining function domains
- The functional predicate calculus is based on predicates as functions
- The "reach" of quantification is captured by function domains.

7.1 Calculating with sets, formally

7.1.0 Rationale of the formalization

a. Relation to axiomatizations

- Intuitive notion of sets assumed known
- The approach is aimed at *formal calculation* with sets
- Largely independent of particular axiomatizations (portability)

b. Set membership as the basic set operator

- Syntax: $\boxed{e \in X}$ with (normally)
  e any expression, X a set expression (introduced soon)
- Examples: $(p \wedge q) \in \mathbb{B}$ $\qquad \pi/2 \in \mathbb{R}$ $\qquad f \mathbin{+\!\!+} g \in \times (F \mathbin{+\!\!+} G)$
  Note: $(e \in X) \in \mathbb{B}$ for any e and X
- Warning: *never* overload the relational operator $\in$ with binding. So,
  $\forall\, x \in X \,.\, p$ and $\{x \in X \mid p\}$ and $\sum n \in \mathbb{N} \,.\, 1/n^2$ are bad syntax.
  Correct will be: $\forall\, x : X \,.\, p$ and $\{x : X \mid p\}$ and $\sum n : \mathbb{N} \,.\, 1/n^2$, shown later.

7.1.1 Equality for sets

Henceforth, X, Y, etc. are metasymbols for set expressions, unless stated otherwise.

a. *Leibniz's principle* $\boxed{e = e' \Rightarrow d[^v_e = d[^v_{e'}}$ as the universal guideline

- Particularization to sets $d = e \Rightarrow (d \in X \;\equiv\; e \in X)$ but, more relevant,

$$\boxed{X = Y \Rightarrow (x \in X \;\equiv\; x \in Y)}$$

- By (WC$\Rightarrow$): $(p \Rightarrow X = Y) \Rightarrow p \Rightarrow (x \in X \;\equiv\; x \in Y)$

b. *Set extensionality* as the converse of Leibniz's principle for sets

$$\boxed{\textsc{Inference rule (strict)}: \quad \frac{p \Rightarrow (x \in X \;\equiv\; y \in Y)}{p \Rightarrow X = Y} \qquad (x \text{ a new variable})}$$

Role of p is proof-technical: the deduction theorem and chaining calculations

$$\boxed{\begin{array}{lll} p \;\Rightarrow & \langle\text{Calculations}\rangle & x \in X \;\equiv\; x \in Y \\ \;\;\;\Rightarrow & \langle\text{Extensionality}\rangle & X = Y \end{array}}$$

Warning: such a proof is for $p \Rightarrow X = Y$, *not* $(x \in X \;\equiv\; x \in Y) \Rightarrow X = Y$.

7.1.2 Set expressions, operators and their calculation rules

a. Set symbols (constants): $\mathbb{B}$ (binary), $\mathbb{N}$ (natural), $\mathbb{Z}$ (integer), etc.

Empty set $\emptyset$, with axiom $x \notin \emptyset$, abbreviating $\boxed{\neg\,(x \in \emptyset)}$

b. Operators and axioms (defined by reduction to proposition calculus)

- Singleton set injector ι with axiom $\boxed{x \in \iota\, y \ \equiv\ x = y}$
  We do *not* use $\{\ \}$ for singletons, but for a more useful purpose.
- Function range $\mathcal{R}$, or synonym $\{\ \}$ (axioms later). Property (proof later)

$$\boxed{e \in \{v \colon X \mid p\} \ \equiv\ e \in X \wedge p[^v_e \text{ provided } x \notin \varphi\, X}$$

- Combining operators: $\cup$ (union), $\cap$ (intersection), $\setminus$ (difference). Axioms

$$\boxed{\begin{array}{rcl} x \in X \cup Y & \equiv & x \in X \vee x \in Y \\ x \in X \cap Y & \equiv & x \in X \wedge x \in Y \\ x \in X \setminus Y & \equiv & x \in X \wedge x \notin Y \end{array}}$$

- Relational operator: *subset* ($\subseteq$). Axiom: $\boxed{X \subseteq Y \ \equiv\ Y = X \cup Y}$

7.2 Calculating with functions, formally

7.2.0 Rationale of the formulation

a. Relation to axiomatizations: a function is *not* a set of pairs, which no more than a set-theoretical *representation* called the *graph* of the function.

b. A *function* is defined by its *domain* (argument type) and its *mapping*, usually

A *domain axiom* of the form	$\mathcal{D}\,f = X$ or $x \in \mathcal{D}\,f \equiv p \quad (f \not\in \varphi\,p)$
A *mapping axiom* of the form	$x \in \mathcal{D}\,f \Rightarrow q$

Existence and uniqueness are proof obligations (trivial for explicit mappings).

c. Example: the function $double$ can be defined by a domain axiom $\mathcal{D}\,double = \mathbb{Z}$ together with a mapping axiom $n \in \mathcal{D}\,double \Rightarrow double\,n = 2 \cdot n$.

d. Example: the function $halve$ can be defined by

- Domain axiom $\mathcal{D}\,halve = \{n : \mathbb{Z} \mid n/2 \in \mathbb{Z}\}$
  Equivalently: $n \in \mathcal{D}\,halve \equiv n \in \mathbb{Z} \wedge n/2 \in \mathbb{Z}$
- Mapping axiom $n \in \mathcal{D}\,halve \Rightarrow halve\,n = n/2$
  Equivalently (implicit): $n \in \mathcal{D}\,halve \Rightarrow n = double\,(halve\,n)$

7.2.1 Equality for functions

Henceforth, f, g, etc. are metasymbols for functions, unless stated otherwise.

a. *Leibniz's principle* particularizes to $x = y \Rightarrow f\,x = f\,y$; more relevant: $f = g \Rightarrow f\,x = g\,x$ and $f = g \Rightarrow \mathcal{D}\,f = \mathcal{D}\,g$. With "guards" for arguments

$$\boxed{f = g \;\Rightarrow\; \mathcal{D}\,f = \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \wedge x \in \mathcal{D}\,g \Rightarrow f\,x = g\,x) \qquad (2)}$$

By (WC⇒): $(p \Rightarrow f = g) \Rightarrow p \Rightarrow \mathcal{D}\,f = \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x)$

b. *Function extensionality* as the converse of Leibniz's principle: with new x,

$$\boxed{\text{(strict inf. rule)}\ \frac{p \Rightarrow \mathcal{D}\,f = \mathcal{D}\,g \,\wedge\, (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x)}{p \Rightarrow f = g} \qquad (3)}$$

Role of p is proof-technical, esp. chaining calculations

$$\boxed{\begin{array}{lll} p & \Rightarrow\ \langle\text{Calculations}\rangle & \mathcal{D}\,f = \mathcal{D}\,g \,\wedge\, (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \\ & \Rightarrow\ \langle\text{Extensionality}\rangle & f = g \end{array}}$$

Warning: such a proof is for $p \Rightarrow f = g$

not for $\mathcal{D}\,f = \mathcal{D}\,g \,\wedge\, (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \Rightarrow f = g$.

7.2.2 Function expressions, operators and their calculation rules

a. Four kinds of function expressions: functions as more than first-class objects

- Two kinds already introduced for simple expressions:
  Identifiers (variables, constants) and *Applications*(e.g., f^{-} and $f \circ g$).
- Two new kinds, *fully completing our language syntax* (nothing more!)
  - Tuplings of the form e, e', e''; domain axiom: $\mathcal{D}\,(e, e', e'') = \{0, 1, 2\}$, mapping: $(e, e', e'')\,0 = e$ and $(e, e', e'')\,1 = e'$ and $(e, e', e'')\,2 = e''$.
  - Abstractions of the form (assuming $v \not\in \varphi\, X$)

$$\boxed{v : X \mathrel{\dot\wedge} p \,.\, e}$$

v is a variable, X a set expression, p a proposition, e any expression. The *filter* $\mathrel{\dot\wedge} p$ is optional, and $v : X \,.\, e$ stands for $v : X \mathrel{\dot\wedge} 1 \,.\, e$.

$$\boxed{\begin{array}{lr} \textsc{Domain axiom:}\ \ d \in \mathcal{D}\,(v : X \mathrel{\dot\wedge} p \,.\, e) \equiv d \in X \wedge p[^{v}_{d} & (4) \\ \textsc{Mapping axiom:}\ d \in \mathcal{D}\,(v : X \mathrel{\dot\wedge} p \,.\, e) \Rightarrow (v : X \mathrel{\dot\wedge} p \,.\, e)\, d = e[^{v}_{d} & (5) \end{array}}$$

Substitution rules are entirely analogous to those of lambda calculus.

b. Some examples regarding abstraction

- Consider $n : \mathbb{Z} \mathbin{\dot\wedge} n \geq 0 \,.\, 2 \cdot n$ The domain axiom yields

$$\boxed{\begin{aligned} m \in \mathcal{D}\,(n : \mathbb{Z} \mathbin{\dot\wedge} n \geq 0 \,.\, 2 \cdot n) &\equiv \langle \text{Domain axm} \rangle \quad m \in \mathbb{Z} \wedge (n \geq 0)[^n_m \\ &\equiv \langle \text{Substitution} \rangle \quad m \in \mathbb{Z} \wedge m \geq 0 \\ &\equiv \langle \text{Definition } \mathbb{N} \rangle \quad m \in \mathbb{N} \end{aligned}}$$

Hence $\mathcal{D}\,(n : \mathbb{Z} \mathbin{\dot\wedge} n \geq 0 \,.\, 2 \cdot n) = \mathbb{N}$ by set extensionality.
If $x + y \in \mathcal{D}\,(n : \mathbb{Z} \mathbin{\dot\wedge} n \geq 0 \,.\, 2 \cdot n)$ or, equivalently, $x + y \in \mathbb{N}$,

$$\boxed{\begin{aligned} (n : \mathbb{Z} \mathbin{\dot\wedge} n \geq 0 \,.\, 2 \cdot n)\,(x + y) &= \langle \text{Mapping axm} \rangle \quad (2 \cdot n)[^n_{x+y} \\ &= \langle \text{Substitution} \rangle \quad 2 \cdot (x + y) \end{aligned}}$$

- Similarly, $\boxed{\mathit{double} = n : \mathbb{Z} \,.\, 2 \cdot n \quad \text{and} \quad \mathit{halve} = n : \mathbb{Z} \mathbin{\dot\wedge} n/2 \in \mathbb{Z} \,.\, n/2}$
- Defining the *constant function definer* $({}^\bullet)$ by

$$\boxed{X \,{}^\bullet\, e = v : X \,.\, e, \ \text{assuming } v \not\in \varphi\, e} \tag{6}$$

and the *empty function* ε and the *single point function definer* $\mapsto$ by

$$\boxed{\varepsilon := \emptyset \,{}^\bullet\, e \quad \text{and} \quad x \mapsto y = \iota\, x \,{}^\bullet\, y} \tag{7}$$

c. Remark Abstractions look unlike common mathematics. Yet, we shall show their use in synthesizing traditional notations in a formally correct and more general way, *while preserving easily recognizable form and meaning*.

For instance, $\sum n : S \,.\, n^2$ will denote the sum of all n^2 as n “ranges” over S. What used to be vague intuitive notions will acquire formal calculation rules.

d. Instantiating function equality with $f := v : X \mathbin{\underset{\cdot}{\wedge}} p \,.\, d$ and $g := v : Y \mathbin{\underset{\cdot}{\wedge}} q \,.\, e$ yields:

THEOREM, EQUALITY FOR ABSTRACTIONS

By Leibniz: $(v : X \mathbin{\underset{\cdot}{\wedge}} p \,.\, d) = (v : Y \mathbin{\underset{\cdot}{\wedge}} q \,.\, e)$

$$\Rightarrow (v \in X \wedge p \equiv v \in Y \wedge q) \wedge (v \in X \wedge p \Rightarrow d = e)$$

By extensionality: $(v \in X \wedge p \equiv v \in Y \wedge q) \wedge (v \in X \wedge p \Rightarrow d = e)$

$$\vdash (v : X \mathbin{\underset{\cdot}{\wedge}} p \,.\, d) = (v \in Y \mathbin{\underset{\cdot}{\wedge}} q \,.\, e)$$

The extensionality part can itself be made clearer by factorizing it as follows

Extensionality, *domain* part:

$$v \in X \wedge p \equiv v \in Y \wedge q \ \vdash\ (v : X \mathbin{\underset{\cdot}{\wedge}} p \,.\, e) = (v : Y \mathbin{\underset{\cdot}{\wedge}} q \,.\, e)$$

Extensionality, *mapping* part:

$$v \in X \wedge p \Rightarrow d = e \ \vdash\ (v : X \mathbin{\underset{\cdot}{\wedge}} p \,.\, d) = (v : X \mathbin{\underset{\cdot}{\wedge}} p \,.\, e)$$

e. Operators for functions

- Terminology: an *operator* is an identifier for a function.
  A *functional* or *higher-order function* is a function over functions.
- Rough classification of functionals
  - *Generic functionals* take arbitrary functions as arguments.
    They are extensively discussed in later "chapters".
  - *Specialized functionals* introduced in particular subject areas.

Remark: In the sequel, we shall occasionally write
$\mathcal{T}$ for the type (or set) universe,
$\mathcal{F}$ for the function universe
$\mathcal{U}$ for the universe.
Various axiomatizations (e.g., Foster) support universal sets (without paradoxes).
We leave the choice open, and just assume $\mathcal{T}$, $\mathcal{F}$ and $\mathcal{U}$ to be "well-behaved", i.e., containing no elements leading to contradictions.

That non-well-behaved sets are excluded is illustrated by the Russell set R defined by $\mathsf{R} = \{X : \mathcal{T} \mid X \notin X\}$, hence $X \in \mathsf{R} \equiv X \in \mathcal{T} \wedge X \notin X$. Instantiation $X := \mathsf{R}$ yields the equation $\mathsf{R} \in \mathsf{R} \equiv \mathsf{R} \in \mathcal{T} \wedge \mathsf{R} \notin \mathsf{R}$ whose only solution implies $\mathsf{R} \notin \mathcal{T}$.

7.3 Generic functionals, a sneak preview

7.3.0 Principle

a. Motivation: twofold

- In a functional formalism (as used here), well-designed functionals can be shared by considerably more mathematical objects than usual.
- Supporting point-free formulations and smooth conversion between pointwise and point-free formulations.

b. Shortcoming of similar operators in traditional mathematics: restrictions on the arguments, e.g.,

- The usual $f \circ g$ requires $\mathcal{R}\, g \subseteq \mathcal{D}\, f$, in which case $\mathcal{D}\,(f \circ g) = \mathcal{D}\, g$
- The usual f^- requires f injective, in which case $\mathcal{D}\, f^- = \mathcal{R}\, f$

c. Approach used here: no restrictions on the arguments, but refine domain of the result function such that its mapping definition does not contain out-of-domain applications for values in this domain (guarded)

7.3.1 Some generic functionals (only as needed for point-free predicate calculus)

a. Filtering operators (infix)

$\downarrow$ *Set filtering* $\quad X \downarrow P = \{x : X \cap \mathcal{D}\, P \mid P\, x\}$ (8)

$\downarrow$ *Function filtering* $\quad f \downarrow P = x \in \mathcal{D}\, f \cap \mathcal{D}\, P \mathrel{\dot\wedge} P\, x \,.\, f\, x$ (9)

$\rceil$ *Function restriction* $\quad f \rceil X = f \downarrow (X^{\bullet}\, 1) = x : \mathcal{D}\, f \cap X \,.\, f\, x$ (10)

Restriction is the familiar(coarser) variant of $\downarrow$. Shorthand: d_e for $d \downarrow e$.

b. *Function composition* (—∘—): $\quad g \circ f = x : \mathcal{D}\, f \mathrel{\dot\wedge} f\, x \in \mathcal{D}\, g \,.\, g\,(f\, x)$ (11)

c. *Direct extension* extends operators over a set X to X-valued functions.

$\overline{\underline{}}$ *Monadic* $\quad \overline{g}\, f = x : \mathcal{D}\, f \mathrel{\dot\wedge} f\, x \in \mathcal{D}\, g \,.\, g\,(f\, x)$ so $\overline{g}\, f = g \circ f$ (12)

$\widehat{\underline{}}$ *Dyadic* $\quad f \mathbin{\widehat{\star}} g = x : \mathcal{D}\, f \cap \mathcal{D}\, g \mathrel{\dot\wedge} (f\, x, g\, x) \in \mathcal{D}\,(\star) \,.\, (f\, x) \star (g\, x)$(13)

$\overset{\leftharpoonup}{\underline{}}$ *Left* $\quad f \mathbin{\overset{\leftharpoonup}{\star}} e = f \mathbin{\widehat{\star}} (\mathcal{D}\, f^{\bullet}\, e)$ (14)

$\overset{\rightharpoonup}{\underline{}}$ *Right* $\quad e \mathbin{\overset{\rightharpoonup}{\star}} f = (\mathcal{D}\, f^{\bullet}\, e) \mathbin{\widehat{\star}} f$ (15)

Legend: arbitrary one-argument function g (prefix), two-argument $\star$ (infix).
Note, e.g., $\mathcal{D}\,(f \mathbin{\widehat{\star}} g) = \{x : \mathcal{D}\, f \cap \mathcal{D}\, g \mid (f\, x, g\, x) \in \mathcal{D}\,(\star)\}$
and $\quad x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) \Rightarrow (f \mathbin{\widehat{\star}} g)\, x = (f\, x) \star (g\, x) \quad$ (less compact).

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

8 Functional Predicate Calculus: calculating with quantifiers

8.0 Deriving basic calculation rules and metatheorems

8.0.0 Predicates and quantifiers

8.0.1 Introductory application to function equality and function types

8.0.2 Direct consequences of the axioms and the duality principle

8.0.3 Some distributivity and monotonicity rules

8.0.4 Case analysis, generalized Shannon expansion and distributivity rules

8.0.5 Instantiation, generalization and their use in proving equational laws

8.0.6 Direct consequences of the axioms and the duality principle

8.1 Expanding the toolkit of calculation rules

8.1.0 Abstractions synthesizing common notations

8.1.1 Selected rules for $\forall$

8.1.2 The one-point rule

8.1.3 Swapping quantifiers/dummies and function comprehension

8.1.4 Useful quick calculation techniques

8.1.5 Two final remarks

8.0 Deriving basic calculation rules and metatheorems

8.0.0 Predicates and quantifiers

a. Definition: a *predicate* is any function P satisfying $x \in \mathcal{D}\,P \Rightarrow P\,x \in \mathbb{B}$.

b. Definition: the *quantifiers* $\forall$ and $\exists$ are predicates over predicates defined by

$$\boxed{\textsc{Axioms:}\ \ \forall\,P \equiv P = \mathcal{D}\,P^{\bullet}\,1 \ \ \text{and}\ \ \exists\,P \equiv P \neq \mathcal{D}\,P^{\bullet}\,0} \qquad (16)$$

Legend: read $\forall\,P$ as “everywhere P” and $\exists\,P$ as “somewhere P”.

c. Remarks

- Simple definition, intuitively clear to engineers/applied mathematicians.
- Calculation rules equally obvious, but derived axiomatically in a moment.
- Point-free style for clarity; familiar forms by taking $x : X\,.\,p$ for P, as in
  $\forall\,x : X\,.\,p$, read: “all x in X satisfy p”,
  $\exists\,x : X\,.\,p$, read: “some x in X satisfy p”.

8.0.1 Introductory application to function equality and function types

a. Function equality is pivotal in the quantifier axioms (16).
Conversely, (16) can unite Leibniz (2) and extensionality (3) for functions.

$$\boxed{\textsc{Theorem, Function equality:}\quad f = g \equiv \mathcal{D}\,f = \mathcal{D}\,g \wedge \forall\,(f \mathbin{\widehat{=}} g) \qquad (17)}$$

Proof: we show ($\Rightarrow$); the second step $\langle$Weakening$\rangle$ is just for saving space.

$$\begin{array}{rcl}
f = g \Rightarrow & \langle \text{Lbnz. (2)} \rangle & \mathcal{D}\,f = \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \\
\Rightarrow & \langle \text{Weakening} \rangle & x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x \\
\equiv & \langle p \equiv (p \equiv 1) \rangle & x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow (f\,x = g\,x) = 1 \\
\equiv & \langle \widehat{\ }\ \text{(13)},\ {}^{\bullet}\ \text{(6)} \rangle & x \in \mathcal{D}\,(f \mathbin{\widehat{=}} g) \Rightarrow (f \mathbin{\widehat{=}} g)\,x = (\mathcal{D}\,(f \mathbin{\widehat{=}} g)\,{}^{\bullet}\,1)\,x \\
\Rightarrow & \langle \text{Extens. (3)} \rangle & (f \mathbin{\widehat{=}} g) = \mathcal{D}\,(f \mathbin{\widehat{=}} g)\,{}^{\bullet}\,1 \\
\equiv & \langle \text{Axiom } \forall \text{ (16)} \rangle & \forall\,(f \mathbin{\widehat{=}} g)
\end{array}$$

Step $\langle$Extens. (3)$\rangle$ tacitly used $\mathcal{D}\,h = \mathcal{D}\,h \cap \mathcal{D}\,((\mathcal{D}\,h)\,{}^{\bullet}\,1)$. Restoring equality $\mathcal{D}\,f = \mathcal{D}\,g$ discarded by weakening yields $f = g \Rightarrow \mathcal{D}\,f = \mathcal{D}\,g \wedge \forall\,(f \mathbin{\widehat{=}} g)$.

Proving ($\Leftarrow$) is the symmetric counterpart (exercise).

b. Our function concept has no (unique) *codomain* associated with it.

Yet, we can specify an approximation or restriction on the images.

Two familiar operators for expressing *function types* (i.e., sets of functions) are formalized here; more refined alternatives are shown later.

$\rightarrow$	*function arrow*	$f \in X \rightarrow Y \ \equiv\ \mathcal{D}\,f = X \wedge \forall\, x : \mathcal{D}\,f \,.\, f\,x \in Y$	(18)
$\nrightarrow$	*partial arrow*	$f \in X \nrightarrow Y \ \equiv\ \mathcal{D}\,f \subseteq X \wedge \forall\, x : \mathcal{D}\,f \,.\, f\,x \in Y$	(19)

Note: Functions of type $X \nrightarrow Y$ are often called “partial”. This is a misnomer: they are proper functions, the type just specifies the domain more loosely.

In fact, here are some simple relationships.

- $X \nrightarrow Y = \bigcup\,(S : \mathcal{P}\,X \,.\, S \rightarrow Y)$
- $|X \nrightarrow Y| = \sum\,(k : 0 \,..\, n \,.\, \binom{n}{k} \cdot m^k) = (m+1)^n = |X \rightarrow (Y \cup \iota\,\bot)|$
  for finite X and Y with $n := |X|$ and $m := |Y|$, since $|X \rightarrow Y| = m^n$.

8.0.2 Direct consequences of the axioms and the duality principle

a. Elementary properties by “head calculation”

- For constant predicates: $\forall\,(X^\bullet 1) \;\equiv\; 1$ and $\exists\,(X^\bullet 0) \;\equiv\; 0$ (by (6, 16))
- For the empty predicate: $\forall\,\varepsilon \;\equiv\; 1$ and $\exists\,\varepsilon \;\equiv\; 0$ (since $\varepsilon = \emptyset^\bullet 1 = \emptyset^\bullet 0$)

b. Theorem, Duality: $\forall\,(\overline{\neg}\,P) \;\equiv\; (\overline{\neg}\,\exists)\,P$ (20)

proof:

$$\begin{array}{lcll}
\forall\,(\overline{\neg}\,P) & \equiv & \langle\text{Def. }\forall\text{ (16), Lemma a (21)}\rangle & \overline{\neg}\,P = \mathcal{D}\,P^\bullet 1 \\
& \equiv & \langle\text{Lemma b (22)}\rangle & P = \overline{\neg}\,(\mathcal{D}\,P^\bullet 1) \\
& \equiv & \langle\text{Lemma c (23), }1 \in \mathcal{D}\,\neg\rangle & P = \mathcal{D}\,P^\bullet(\neg\,1) \\
& \equiv & \langle\neg\,1 = 0\text{, definition }\exists\text{ (16)}\rangle & \neg\,(\exists\,P) \\
& \equiv & \langle\text{Defin. }\overline{}\text{ and }\exists\,P \in \mathcal{D}\,\neg\rangle & \overline{\neg}\,\exists\,P
\end{array}$$

The lemmata used are stated below, their proofs are routine.

Lemma a $\mathcal{D}\,(\overline{\neg}\,P) = \mathcal{D}\,P$ (21)

Lemma b: $\overline{\neg}\,P = Q \;\equiv\; P = \overline{\neg}\,Q$ (22)

Lemma c: $x \in \mathcal{D}\,g \Rightarrow \overline{g}\,(X^\bullet x) = g \circ (X^\bullet x) = X^\bullet (g\,x)$ (23)

8.0.3 Some distributivity and monotonicity rules

a. $\boxed{\textsc{Theorem, Collecting } \forall/\wedge\text{:}\quad \forall\, P \wedge \forall\, Q \Rightarrow \forall\,(P \mathbin{\widehat{\wedge}} Q)}$ (24)

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textsc{Proof:}\ \ \forall\, P \wedge \forall\, Q} \\
\equiv \langle \text{Defin. } \forall \rangle & P = \mathcal{D}\, P^{\bullet} 1 \wedge Q = \mathcal{D}\, Q^{\bullet} 1 \\
\Rightarrow \langle \text{Leibniz} \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\,(\mathcal{D}\, P^{\bullet} 1 \mathbin{\widehat{\wedge}} \mathcal{D}\, Q^{\bullet} 1) \\
\equiv \langle \text{Defin. } \widehat{\ } \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\, x : \mathcal{D}\, P \cap \mathcal{D}\, Q\,.\,(\mathcal{D}\, P^{\bullet} 1)\, x \wedge (\mathcal{D}\, Q^{\bullet} 1)\, x \\
\equiv \langle \text{Defin. } {}^{\bullet}) \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\, x : \mathcal{D}\, P \cap \mathcal{D}\, Q\,.\, 1 \wedge 1 \\
\equiv \langle \forall\,(X^{\bullet} 1) \rangle & \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv 1
\end{array}
$$

Here is a summary of similar theorems , dual theorems and corollaries.

Theorem, Collecting $\forall/\wedge$:	$\forall\, P \wedge \forall\, Q \Rightarrow \forall\,(P \mathbin{\widehat{\wedge}} Q)$
Theorem, Splitting $\forall/\wedge$:	$\mathcal{D}\, P = \mathcal{D}\, Q \Rightarrow \forall\,(P \mathbin{\widehat{\wedge}} Q) \Rightarrow \forall\, P \wedge \forall\, Q$
Theorem, Distributivity $\forall/\wedge$:	$\mathcal{D}\, P = \mathcal{D}\, Q \Rightarrow (\forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\, P \wedge \forall\, Q)$
Theorem, Collecting $\exists/\vee$:	$\mathcal{D}\, P = \mathcal{D}\, Q \Rightarrow \exists\, P \vee \exists\, Q \Rightarrow \exists\,(P \mathbin{\widehat{\vee}} Q)$
Theorem, Splitting $\exists/\vee$:	$\exists\,(P \mathbin{\widehat{\vee}} Q) \Rightarrow \exists\, P \vee \exists\, Q$
Theorem, Distributivity $\exists/\vee$:	$\mathcal{D}\, P = \mathcal{D}\, Q \Rightarrow (\exists\,(P \mathbin{\widehat{\vee}} Q) \equiv \exists\, P \vee \exists\, Q)$

b. Properties for equal predicates; monotonicity rules

$$\boxed{\begin{array}{lll} \textsc{Equal pred.}\ \backslash\forall: & \mathcal{D}\,P = \mathcal{D}\,Q \;\Rightarrow\; \forall\,(P \mathrel{\widehat{\equiv}} Q) \Rightarrow (\forall\,P \equiv \forall\,Q) & (25) \\ \textsc{Equal pred.}\ \backslash\exists: & \mathcal{D}\,P = \mathcal{D}\,Q \;\Rightarrow\; \forall\,(P \mathrel{\widehat{\equiv}} Q) \Rightarrow (\exists\,P \equiv \exists\,Q) & (26) \\ \textsc{Monotony}\ \forall/\Rightarrow: & \mathcal{D}\,Q \subseteq \mathcal{D}\,P \;\Rightarrow\; \forall\,(P \mathrel{\widehat{\Rightarrow}} Q) \Rightarrow (\forall\,P \Rightarrow \forall\,Q) & (27) \\ \textsc{Monotony}\ \exists/\Rightarrow: & \mathcal{D}\,P \subseteq \mathcal{D}\,Q \;\Rightarrow\; \forall\,(P \mathrel{\widehat{\Rightarrow}} Q) \Rightarrow (\exists\,P \Rightarrow \exists\,Q) & (28) \end{array}}$$

- Proof outlines (intended as exercises with hints):
  - (25) and (26): function equality (17), Leibniz, (T⇒), or monotony.
  - (27): shunting $\forall\,(P \mathrel{\widehat{\Rightarrow}} Q)$ and $\forall\,P$, expanding $\forall\,P$ by (16), Leibniz.
  - (28): from (27) via contrapositivity and duality.
- Importance: crucial in chaining proof steps:

$$\boxed{\forall\,(P \mathrel{\widehat{\Rightarrow}} Q) \quad \text{justifies} \quad \forall\,P \Rightarrow \forall\,Q \quad \text{and} \quad \exists\,P \Rightarrow \exists\,Q}$$

provided the corresponding set inclusion for the domains holds.

c. Constant predicates, general form

We saw $\forall\,(X^{\bullet}1) \;\equiv\; 1$ and $\exists\,(X^{\bullet}0) \;\equiv\; 0$. How about $\forall\,(X^{\bullet}0)$ and $\exists\,(X^{\bullet}1)$? *Beware of immature intuition!* Formal proof to the rescue, develop intuition!

$$\boxed{\textsc{Theorem, Constant predicate under } \forall \quad \forall\,(X^{\bullet}p) \;\equiv\; X = \emptyset \vee p} \tag{29}$$

$$
\begin{array}{lll}
\textsc{Proof:}\;\; \forall\,(X^{\bullet}p) & \equiv\langle\text{Def. } \forall \text{ and } {}^{\bullet}\rangle & X^{\bullet}p = X^{\bullet}1 \\
& \Rightarrow\langle\text{Leibniz (2)}\rangle & x \in X \Rightarrow (X^{\bullet}p)\,x = (X^{\bullet}1)\,x \\
& \equiv\langle\text{Definition } {}^{\bullet}\rangle & x \in X \Rightarrow p = 1 \\
& \equiv\langle\text{Prop. calc.}\rangle & (x \in X \;\equiv\; 0) \vee p \\
& \equiv\langle x \in \emptyset \;\equiv\; 0\rangle & (x \in X \;\equiv\; x \in \emptyset) \vee p \\
& \Rightarrow\langle\text{Set extensionality}\rangle & X = \emptyset \vee p
\end{array}
$$

$$
\begin{array}{lll}
X = \emptyset \vee p & \equiv\langle p = 1 \;\equiv\; p\rangle & X = \emptyset \vee p = 1 \\
& \Rightarrow\langle\text{Leibniz (L=)}\rangle & (X^{\bullet}p = X^{\bullet}1 \;\equiv\; \emptyset^{\bullet}p = \emptyset^{\bullet}1) \vee X^{\bullet}p = X^{\bullet}1 \\
& \equiv\langle \emptyset^{\bullet}p = \emptyset^{\bullet}x\rangle & X^{\bullet}p = X^{\bullet}1 \vee X^{\bullet}p = X^{\bullet}1 \\
& \equiv\langle\text{Idempot. } \vee\rangle & X^{\bullet}p = X^{\bullet}1 \\
& \equiv\langle\text{Axm. } \forall \text{ (16)}\rangle & \forall\,(X^{\bullet}p)
\end{array}
$$

So $\forall\,\varepsilon \;\equiv\; 1$ and $\forall\,(X^{\bullet}1) \;\equiv\; 1$ and $\forall\,(X^{\bullet}0) \;\equiv\; X = \emptyset$. Dual: $\exists\,(X^{\bullet}p) \;\equiv\; X \neq \emptyset \wedge p$, so $\exists\,\varepsilon \;\equiv\; 0$ and $\exists\,(X^{\bullet}0) \;\equiv\; 0$ and $\exists\,(X^{\bullet}1) \;\equiv\; X \neq \emptyset$.

8.0.4 Case analysis, generalized Shannon expansion and distributivity rules

a. Recall propositional metatheorems: (i) case analysis $p_0^v \wedge p_1^v \Rightarrow p$ (or $p_0^v, p_1^v \vdash p$), (ii) Shannon, e.g., $p \equiv (v \Rightarrow p_1^v) \wedge (\neg\, v \Rightarrow p_0^v)$ and $p \equiv (v \wedge p_1^v) \vee (\neg\, v \wedge p_0^v)$. We want this power for predicate calculus.

b. A little technicality: for expression e and variable v, let D_e^v denote the domain. Informal definition: largest X s.t. $d \in X \Rightarrow (e_d^v$ only in-domain applications)

Lemma, Case analysis: If $\mathbb{B} \subseteq \mathsf{D}_P^v$ and $v \in \mathbb{B}$ then

$$\boxed{\forall\, P_0^v \wedge \forall\, P_1^v \Rightarrow \forall\, P} \tag{30}$$

c. Theorem, Shannon expansion If $\mathbb{B} \subseteq \mathsf{D}_P^v$ and $v \in \mathbb{B}$ then

$$\boxed{\begin{array}{lcl} \forall\, P & \equiv & (v \Rightarrow \forall\, P_1^v) \wedge (\neg\, v \Rightarrow \forall\, P_0^v) \\ \forall\, P & \equiv & (v \vee \forall\, P_0^v) \wedge (\neg\, v \vee \forall\, P_1^v) \\ \forall\, P & \equiv & (v \wedge \forall\, P_1^v) \vee (\neg\, v \wedge \forall\, P_0^v) \end{array}} \tag{31}$$

and other variants. Proofs by case analysis.

d. Application examples: Distributivity and pseudodistributivity theorems

Left distributivity $\Rightarrow/\forall$: $p \Rightarrow \forall\, P \equiv \forall\,(p \overset{\rightharpoonup}{\Rightarrow} P)$ (32)

Right distributivity $\Rightarrow/\exists$: $\exists\, P \Rightarrow p \equiv \forall\,(P \overset{\leftharpoonup}{\Rightarrow} p)$ (33)

Distributivity of $\vee/\forall$: $p \vee \forall\, P \equiv \forall\,(p \overset{\rightharpoonup}{\vee} P)$ (34)

Pseudodistributivity $\wedge/\forall$: $(p \wedge \forall\, P) \vee \mathcal{D}\, P = \emptyset \equiv \forall\,(p \overset{\rightharpoonup}{\wedge} P)$ (35)

Proof: exercises

e. Clearly, theorems, (30) and (31) also hold if $\forall$ is replaced everywhere by $\exists$.

This yields a collection of similar laws, also obtainable by duality.

Examples:

Left pseudodistr. $\Rightarrow/\exists$: $(p \Rightarrow \exists\, P) \wedge \mathcal{D}\, P \neq \emptyset \equiv \exists\,(p \overset{\rightharpoonup}{\Rightarrow} P)$

Right pseudodist. $\Rightarrow/\forall$: $(\forall\, P \Rightarrow p) \wedge \mathcal{D}\, P \neq \emptyset \equiv \exists\,(P \overset{\leftharpoonup}{\Rightarrow} p)$

Pseudodistribut. $\vee/\exists$: $(p \vee \exists\, P) \wedge \mathcal{D}\, P \neq \emptyset \equiv \exists\,(p \overset{\rightharpoonup}{\vee} P)$

Distributivity of $\wedge/\exists$: $p \wedge \exists\, P \equiv \exists\,(p \overset{\rightharpoonup}{\wedge} P)$

8.0.5 Instantiation, generalization and their use in proving equational laws

a. Theorem, Instantiation and Generalization (note: (37), assumes new v)

Instantiation: $\forall\,P \Rightarrow e \in \mathcal{D}\,P \Rightarrow P\,e$ (36)

Generalization: $p \Rightarrow v \in \mathcal{D}\,P \,\Rightarrow P\,v \;\vdash\; p \Rightarrow \forall\,P$ (37)

Proofs: in (2) and (3), let $f := P$ and $g := \mathcal{D}\,P\,{}^{\bullet}1$, then apply (6) and (16).

b. Metatheorem, $\forall$-introduction/removal again assuming new v,

$\boxed{p \Rightarrow \forall\,P \text{ is a theorem iff } p \Rightarrow v \in \mathcal{D}\,P \Rightarrow P\,v \text{ is a theorem.}}$ (38)

Significance: for $p = 1$, this reflects typical implicit use of generalization: to prove $\forall\,P$, prove $v \in \mathcal{D}\,P \Rightarrow P\,v$, or assume $v \in \mathcal{D}\,P$ and prove $P\,v$.

c. Metatheorem, Witness assuming new v

$\boxed{\exists\,P \Rightarrow p \text{ is a theorem iff } v \in \mathcal{D}\,P \Rightarrow P\,v \Rightarrow p \text{ is a theorem.}}$ (39)

Significance: this formalizes the following well-known informal proof scheme: to prove $\exists\,P \Rightarrow p$, "take" a v in $\mathcal{D}\,P$ s.t. $P\,v$ (the "witness") and prove p.

d. We allow weaving generalization (37) into a calculation chain as follows.

Metatheorem, generalization of the consequent : assuming new v,

$$\boxed{\begin{array}{lll} p & \Rightarrow \langle\text{Calculation to } v \in \mathcal{D}\,P \Rightarrow P\,v\rangle & v \in \mathcal{D}\,P \Rightarrow P\,v \\ & \Rightarrow \quad \langle\text{Generalizing the consequent}\rangle & \forall\,P \end{array}} \qquad (40)$$

Expected warning: this proof is for $p \Rightarrow \forall\,P$, *not* $(v \in \mathcal{D}\,P \Rightarrow P\,v) \Rightarrow \forall\,P$. We use it only for deriving calculation rules; it is rarely (if ever) needed beyond.

e. Application example: proving a very important theorem.

$$\boxed{\text{Theorem, Trading under } \forall\text{:} \quad \forall\,P_Q \equiv \forall\,(Q \mathrel{\widehat{\Rightarrow}} P)} \qquad (41)$$

Proof: We show ($\Rightarrow$), the reverse being analogous.

$$\boxed{\begin{array}{llll} \forall\,P_Q & \Rightarrow & \langle\text{Instantiation (36)}\rangle & v \in \mathcal{D}\,(P_Q) \Rightarrow P_Q\,v \\ & \equiv & \langle\text{Definition} \downarrow \text{(81)}\rangle & v \in \mathcal{D}\,P \cap \mathcal{D}\,Q \wedge Q\,v \Rightarrow P\,v \\ & \equiv & \langle\text{Shunting } \wedge \text{ to } \Rightarrow\rangle & v \in \mathcal{D}\,P \cap \mathcal{D}\,Q \Rightarrow Q\,v \Rightarrow P\,v \\ & \equiv & \langle\text{Axiom } \widehat{\ }\text{, remark}\rangle & v \in \mathcal{D}\,(Q \mathrel{\widehat{\Rightarrow}} P) \Rightarrow (Q \mathrel{\widehat{\Rightarrow}} P)\,v \\ & \Rightarrow & \langle\text{Gen. conseq. (40)}\rangle & \forall\,(Q \mathrel{\widehat{\Rightarrow}} P) \end{array}}$$

The remark in question is $v \in \mathcal{D}\,P \cap \mathcal{D}\,Q \Rightarrow (Q\,v, P\,v) \in \mathcal{D}\,(\Rightarrow)$.

f. Remarks

- Reversion principle: any proof starting with instantiation, continuing with *equivalences* $(\equiv)$ *only*, and concluding with generalization of the consequent, can be reversed trivially, so the reverse proof can be omitted.
- The dual of trading under $\forall$, viz., $\forall\, P_Q \equiv \forall\, (Q \mathbin{\widehat{\Rightarrow}} P)$, is the following

$$\boxed{\textsc{Theorem, Trading under } \exists\text{:}\quad \exists\, P_Q \equiv \exists\, (Q \mathbin{\widehat{\wedge}} P)} \qquad (42)$$

Proof: Observinfing software reuse and the algebraic style, we use (41).

$$\boxed{\begin{array}{lccl}
\exists\, P_Q & \equiv & \langle \text{Duality (20)} \rangle & \widehat{\neg}\forall\, (\widehat{\neg}\, P_Q) \\
 & \equiv & \langle \widehat{\neg}\, P_Q = (\widehat{\neg}\, P)_Q \rangle & \widehat{\neg}\forall\, (\widehat{\neg}\, P)_Q \\
 & \equiv & \langle \text{Trading } \forall \text{ (41)} \rangle & \widehat{\neg}\forall\, (Q \mathbin{\widehat{\Rightarrow}} \widehat{\neg}\, P) \\
 & \equiv & \langle q \Rightarrow p \equiv \neg\, q \vee p \rangle & \widehat{\neg}\forall\, (\widehat{\neg}\, Q \mathbin{\widehat{\vee}} \widehat{\neg}\, P) \\
 & \equiv & \langle \text{De Morgan} \rangle & \widehat{\neg}\forall\, (\widehat{\neg}\, (Q \mathbin{\widehat{\wedge}} P)) \\
 & \equiv & \langle \text{Duality (20)} \rangle & \exists\, (Q \mathbin{\widehat{\wedge}} P)
\end{array}}$$

Note: direct extensions of propositional operators are simple counterparts of the basic rules. Hence we use the latter in the justification brackets.

8.1 Expanding the toolkit of calculation rules

8.1.0 Abstractions synthesizing common notations

a. Abstractions are useful for synthesizing or “repairing” common notations.

Example: let $R := v : X \,.\, r$ and $P := v : X \,.\, p$ in the trading theorems (41, 42)

$$\boxed{\begin{array}{lcl} \forall\,(v : X \mathbin{\underset{\cdot}{\wedge}} r \,.\, p) & \equiv & \forall\,(v : X \,.\, r \Rightarrow p) \\ \exists\,(v : X \mathbin{\underset{\cdot}{\wedge}} r \,.\, p) & \equiv & \exists\,(v : X \,.\, r \wedge p). \end{array}} \tag{43}$$

For readers not yet fully comfortable with direct extensions: a *direct* proof for (43) instead of presenting it an instance of the general formulation (41).

$$\boxed{\begin{array}{lll} \multicolumn{3}{l}{\forall\,(v : X \mathbin{\underset{\cdot}{\wedge}} r \,.\, p)} \\ \Rightarrow & \langle\text{Instantiation (36)}\rangle & v \in \mathcal{D}\,(v : X \mathbin{\underset{\cdot}{\wedge}} r \,.\, p) \Rightarrow (v : X \mathbin{\underset{\cdot}{\wedge}} r \,.\, p)\, v \\ \equiv & \langle\text{Abstraction (4), (5)}\rangle & v \in X \wedge r \Rightarrow p \\ \equiv & \langle\text{Shunting } \wedge \text{ to } \Rightarrow\rangle & v \in X \Rightarrow r \Rightarrow p \\ \Rightarrow & \langle\text{Gen. conseq. (40)}\rangle & \forall\,(v : X \,.\, r \Rightarrow p) \end{array}}$$

The converse follows from the reversion principle.

b. Further example: summary of some selected axioms and theorems

Table for $\forall$	General form	Form with $P := v : X\,.\,p$ and $v \not\in \varphi\, q$
Definition	$\forall\, P \;\equiv\; P = \mathcal{D}\, P^{\bullet}\, 1$	$\forall\,(v : X\,.\,p) \;\equiv\; (v : X\,.\,p) = (v : X\,.\,1)$
Instantiation	$\forall\, P \Rightarrow e \in \mathcal{D}\, P \Rightarrow P\, e$	$\forall\,(v : X\,.\,p) \Rightarrow e \in X \Rightarrow p[^v_e$
Generalization	$v \in \mathcal{D}\, P \Rightarrow P\, v \vdash \forall\, P$	$v \in X \Rightarrow p \vdash \forall\,(v : X\,.\,p)$
L-dstr. $\Rightarrow/\forall$	$\forall\,(q \overset{\rightharpoonup}{\Rightarrow} P) \;\equiv\; q \Rightarrow \forall\, P$	$\forall\,(v : X\,.\,q \Rightarrow p) \;\equiv\; q \Rightarrow \forall\,(v : X\,.\,p)$

Table for $\exists$	General form	Form with $P := v : X\,.\,p$ and $v \not\in \varphi\, q$
Definition	$\exists\, P \;\equiv\; P \neq \mathcal{D}\, P^{\bullet}\, 0$	$\exists\,(v : X\,.\,p) \;\equiv\; (v : X\,.\,p) \neq (v : X\,.\,0)$
$\exists$-introduction	$e \in \mathcal{D}\, P \Rightarrow P\, e \Rightarrow \exists\, P$	$e \in X \Rightarrow p[^v_e \Rightarrow \exists\,(v : X\,.\,p)$
Distrib. $\exists/\wedge$	$\exists\,(q \overset{\rightharpoonup}{\wedge} P) \;\equiv\; q \wedge \exists\, P$	$\exists\,(v : X\,.\,q \wedge p) \;\equiv\; q \wedge \exists\,(v : X\,.\,p)$
R-dstr. $\Rightarrow/\exists$	$\forall\,(P \overset{\leftharpoonup}{\Rightarrow} q) \;\equiv\; \exists\, P \Rightarrow q$	$\forall\,(v : X\,.\,p \Rightarrow q) \;\equiv\; \exists\,(v : X\,.\,p) \Rightarrow q$

The general form is the point-free one; the pointwise variants are instantiations.

8.1.1 Selected rules for $\forall$

a. A few more important rules in algebraic style.

Legend: P and Q: *predicates*; R: higher-order predicate (function such that $R\,v$ is a predicate for any v in $\mathcal{D}\,R$): S: relation (predicate on pairs).

The *currying* operator $—^{\mathsf{C}}$ transforms any function f with domain of the form $X \times Y$ into a higher-order function f^{C} defined by $f^{\mathsf{C}} = v : X \,.\, y : Y \,.\, f\,(v, y)$.

Merge rule:	$P \copyright Q \Rightarrow \forall\,(P \mathbin{\dot{\cup}} Q) = \forall\,P \wedge \forall\,Q$
Transposition:	$\forall\,(\forall \circ R) = \forall\,(\forall \circ R^{\mathsf{T}})$
Nesting:	$\forall\,S = \forall\,(\forall \circ S^{\mathsf{C}})$
Composition rule:	$\forall\,P \equiv \forall\,(P \circ f)$ provided $\mathcal{D}\,P \subseteq \mathcal{R}\,f$
One-point rule:	$\forall\,P_{=e} \equiv e \in \mathcal{D}\,P \Rightarrow P\,e$

The *range* operator $\mathcal{R}$, defined by $y \in \mathcal{R}\,f \equiv \exists\,x : \mathcal{D}\,f \,.\, f\,x = y$, will be discussed later. Some (but not all) proofs will also appear when appropriate.

b. Similar rules using dummies.

Legend: p, q and r: $\mathbb{B}$-valued *expressions*.
Assume the usual restrictions on types and free occurrences are satisfied.

DOMAIN SPLIT:	$\forall\,(x : X \cup Y \,.\, p) \;\equiv\; \forall\,(x : X \,.\, p) \wedge \forall\,(x : Y \,.\, p)$
DUMMY SWAP:	$\forall\,(x : X \,.\, \forall\, y : Y \,.\, p) \;\equiv\; \forall\,(y : Y \,.\, \forall\, x : X \,.\, p)$
NESTING:	$\forall\,((x, y) : X \times Y \,.\, p) \;\equiv\; \forall\,(x : X \,.\, \forall\, y : Y \,.\, p)$
DUMMY CHANGE:	$\forall\,(y : Y \,.\, p) \;\equiv\; \forall\,(x : \mathcal{D}\, f \,.\, p[^{y}_{f\,x})$ if $y \in Y \;\equiv\; \exists\, x : \mathcal{D}\, f \,.\, f\, x = y$
ONE-POINT RULE:	$\forall\,(x : X \underset{\cdot}{\wedge} x = e \,.\, p) \;\equiv\; e \in X \Rightarrow p[^{x}_{e}$

8.1.2 The one-point rule

a. Significance: largely ignored by theoreticians, very often useful in practice. Also: instantiation $(\forall P \Rightarrow e \in \mathcal{D}\,P \Rightarrow P\,e)$ has the same r.h.s., but the one-point rule is an equivalence, hence stronger. The proof is also instructive

b. $\boxed{\text{THEOREM, ONE-POINT RULE:}\quad \forall P_{=e} \equiv e \in \mathcal{D}\,P \Rightarrow P\,e}$ (44)

PROOF We use mutual implication. For brevity, let X stand for $\mathcal{D}\,P$ in

$$\boxed{\begin{array}{lllll}
 & \forall P_{=e} & & & \\
 & & \equiv & \langle\text{Filter, trading}\rangle & \forall\, x : X \,.\, x = e \Rightarrow P\,x \\
 & & \Rightarrow & \langle\text{Instant. } x := e\rangle & e \in X \Rightarrow P\,e \\
1 & & \equiv & \langle\text{Leibniz (2)}\rangle & \forall\, x : X \,.\, x = e \Rightarrow (x \in X \Rightarrow P\,x \equiv e \in X \Rightarrow P\,e) \\
 & & \equiv & \langle 1 \Rightarrow p \equiv p\rangle & \forall\, x : X \,.\, x = e \Rightarrow (P\,x \equiv e \in X \Rightarrow P\,e) \\
 & & \Rightarrow & \langle\text{Weakening}\rangle & \forall\, x : X \,.\, x = e \Rightarrow (e \in X \Rightarrow P\,e) \Rightarrow P\,x \\
 & & \equiv & \langle\text{Shunting}\rangle & \forall\, x : X \,.\, (e \in X \Rightarrow P\,e) \Rightarrow x = e \Rightarrow P\,x \\
 & & \equiv & \langle\text{L-dstr. } \Rightarrow/\forall\rangle & (e \in X \Rightarrow P\,e) \Rightarrow \forall\, x : X \,.\, x = e \Rightarrow P\,x \\
 & & \equiv & \langle\text{Filter, trading}\rangle & (e \in X \Rightarrow P\,e) \Rightarrow \forall P_{=e}.
\end{array}}$$

c. Final remarks on the one-point rule

- Point-free and pointwise forms for ∀:

$$\boxed{\begin{array}{rcl} \forall\, P_{=e} & \equiv & e \in \mathcal{D}\, P \Rightarrow P\, e \\ \forall\,(x : X\, .\, x = e \Rightarrow p) & \equiv & e \in X \Rightarrow p[^x_e \end{array}}$$

- Duals for ∃ (both styles):

$$\boxed{\begin{array}{rcl} \exists\, P_{=e} & \equiv & e \in \mathcal{D}\, P \wedge P\, e \\ \exists\,(x : X\, .\, x = e \wedge p) & \equiv & e \in X \wedge p[^x_e \end{array}}$$

- Investigating what happens when implication in $x = e \Rightarrow P\,x$ is reversed yields a one-directional variant (better a half pint than an empty cup).

Theorem, Half-pint rule:

$$\forall\,(x : \mathcal{D}\, P\, .\, P\, x \Rightarrow x = e) \;\Rightarrow\; \exists\, P \Rightarrow P\, e \tag{45}$$

Hint for proof: start with Leibniz and weakening to $x = e \Rightarrow P\,x \Rightarrow P\,e$.

8.1.3 Swapping quantifiers/dummies and function comprehension

a. A simple swapping rule (“homogeneous” $=$ same kind of quantifier)

Theorem, Homogeneous swapping:

$$\begin{aligned} \forall\,(x\!:\!X\,.\,\forall\,y\!:\!Y\,.\,p) &\equiv \forall\,(y\!:\!Y\,.\,\forall\,x\!:\!X\,.\,p) \\ \exists\,(x\!:\!X\,.\,\exists\,y\!:\!Y\,.\,p) &\equiv \exists\,(y\!:\!Y\,.\,\exists\,x\!:\!X\,.\,p) \end{aligned} \qquad (46)$$

b. Heterogeneous swapping: this is less evident, and direction-dependent

Theorem, Moving $\forall$ outside $\exists$:

$$\exists\,(y\!:\!Y\,.\,\forall\,x\!:\!X\,.\,p) \Rightarrow \forall\,(x\!:\!X\,.\,\exists\,y\!:\!Y\,.\,p) \qquad (47)$$

Proof: subtle but easy with Gries’s hint: to prove $p \Rightarrow q$, prove $p \vee q \equiv q$.

The converse (moving $\exists$ outside $\forall$) is not a theorem but an axiom

Axiom, function comprehension:

$$\forall\,(x\!:\!X\,.\,\exists\,y\!:\!Y\,.\,p) \Rightarrow \exists\,f\!:\!X \rightarrow Y\,.\,\forall\,x\!:\!X\,.\,p[^{y}_{f\,x}. \qquad (48)$$

The other direction ($\Leftarrow$) is easy to prove.

8.1.4 Useful quick calculation techniques

a. Motivation We have derived the rules that are most needed in practice, giving them suggestive mnemonic names for easy reference (as in Dijkstra, Gries e.a.). *These names prove extremely valuable in becoming conversant with the rules!*

However, no list of rules is complete or always at hand. Fortunately, there are:

b. Some quick calculation techniques (brief overview)

- Case analysis (30) and Shannon's theorem (31) are very effective.
- Spot-checking plausibility with proposition pairs. Principle: tuples are functions; for propositional p and q, the pair p, q is a predicate and

$$\boxed{\forall\,(p,q) \equiv p \wedge q \quad \text{and} \quad \exists\,(p,q) \equiv p \vee q.} \tag{49}$$

Plausibility of formulas involving a predicate P can be quickly tested for the special case $\mathcal{D}\,P = \mathbb{B}$, for which $\forall\,P = P\,0 \wedge P\,1$ and $\exists\,P = P\,0 \vee P\,1$. Warning: in spot-checking, $\mathcal{D}\,P = \emptyset$ must not be overlooked.

Note: all rules derived in our formalism are "robust" w.r.t. the empty domain.

c. Example of spot-checking with pairs: let the “DUT” be the rather non-intuitive

$$\boxed{\forall\,(x : \mathcal{D}\,P\,.\,P\,x \Rightarrow p) \;\;\equiv\;\; \exists\,(x : \mathcal{D}\,P\,.\,P\,x) \Rightarrow p \qquad \text{assuming new } x}$$

For $\mathcal{D}\,P = \mathbb{B}$ the DUT yields $\forall\,(x : \mathbb{B}\,.\,(P_0, P_1)\,x \Rightarrow p) \;\;\equiv\;\; \exists\,(P_0, P_1) \Rightarrow p$ or

$$\boxed{(P_0 \Rightarrow p) \wedge (P_1 \Rightarrow p) \;\;\equiv\;\; P_0 \vee P_1 \Rightarrow p}$$

which is clear on sight (proposition algebra, case analysis) and boosts intuition.

Check for the empty domain: $1 \;\equiv\; 0 \Rightarrow p$, which is correct.

d. Another example of spot-checking the empty domain: the similar conjecture

$$\boxed{\forall\,(x : \mathcal{D}\,P\,.\,P\,x \wedge p) \;\;\equiv\;\; \forall\,(x : \mathcal{D}\,P\,.\,P\,x) \wedge p \qquad \text{Wrong!}}$$

holds for pairs (exercise) but in general only only insofar as $\mathcal{D}\,P \neq \emptyset$.

If $\mathcal{D}\,P = \emptyset$, we obtain $1 \;\equiv\; 1 \wedge p$, making $p = 0$ a counterexample.

Note: all rules derived in our formalism are “robust” w.r.t. the empty domain.

8.1.5 Two final remarks

a. An important style issue

By now, we have factored out derivations which themselves were not always equational (separating $\equiv$ in $\Rightarrow$ and $\Leftarrow$, the "mutual implication" rule).

The resulting calculation rules are equational, and are mostly used as such in applications. Separating $\equiv$ in $\Rightarrow$ and $\Leftarrow$ is needed rarely, and poor style otherwise, except when the two directions have interesting distinct side results.

b. Omitting universal quantification

In applications, generalization is often used (implicitly) to avoid carrying around outermost universal quantification and bindings in every calculation step.

Implicit quantification by generalization must be used with care: leaving universal quantifiers implicit is a major source of errors in inductive proofs.

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

9 Functional Predicate Calculus: application examples

9.0 Endogenous application examples

9.0.0 The empty domain issue: “Andrew’s challenge” revisited

9.0.1 The function range, with an application to set comprehension

9.1 Exogenous application examples

9.1.0 Limits in mathematical analysis

9.1.1 Adherence as a predicate transformer

9.1.2 Calculating with properties and operators regarding orderings

9.1.3 A case study: when is a greatest lower bound a least element?

9.0 Endogenous application examples

9.0.0 The empty domain issue: “Andrew’s challenge” revisited

a. Introduction: terminology and problem statement

- Informal language with quantifiers: “the variable ‘ranges’ over ...” (range?). Here: not part of the quantifier but exactly the *domain* of the argument. Example: in $\forall\, P$ it is $\mathcal{D}\, P$. The principle is called “Domain modulation”.
- Problem: other quantifier calculi handle the empty domain poorly
- Example: *Andrew’s challenge* (name given by Gries) is proving

$$\boxed{\begin{array}{l}((\exists x \forall y \,|\!: p.x \;\equiv\; p.y) \;\equiv\; ((\exists x \,|\!: q.x) \;\equiv\; ((\forall y \,|\!: p.y))) \quad \equiv \\ ((\exists x \forall y \,|\!: q.x \;\equiv\; q.y) \;\equiv\; ((\exists x \,|\!: p.x) \;\equiv\; ((\forall y \,|\!: q.y)))\end{array}}$$

(notation from source) or, by (A$\equiv$) and (C$\equiv$), proving
$A_p \;\equiv\; A_q$ where $A_p \;\equiv\; (\exists x \forall y \,|\!: p.x \;\equiv\; p.y) \;\equiv\; (\exists x \,|\!: p.x) \;\equiv\; (\forall y \,|\!: p.y)$
Sufficient: prove A_p for arbitrary p.

b. Existing proofs for A_p, i.e.,

$$\boxed{\exists x \forall y \,|\!:\, p.x \;\equiv\; p.y) \;\equiv\; (\exists x \,|\!:\, p.x) \;\equiv\; (\forall y \,|\!:\, p.y)}$$

- Gries's proof: easily found on the web
- Dijkstra EWD 1247, "Beware of the empty range"
  Most instructive, but needs case analysis for empty/nonempty domain
- R. Joshi, K. S. Namjoshi in EWD 1249, "Andrew's Challenge once more"
  Avoids case analysis by not using rules that differ for the empty domain

Remark: the latter proofs use ad hoc abbreviations

$$\boxed{[t] \;\equiv\; \langle \forall x :: t.x \rangle \quad \text{and} \quad \langle t \rangle \;\equiv\; \langle \exists x :: t.x \rangle}$$

Obviated in our formalism by point-free style, since $\forall$ and $\exists$ are functions and $P = x : \mathcal{D}\,P \,.\, P\,x$, hence $\forall\,P \;\equiv\; \forall\,x : \mathcal{D}\,P \,.\, P\,x$ and $\exists\,P \;\equiv\; \exists\,x : \mathcal{D}\,P \,.\, P\,x$.

c. Our variant of the EWD 1249 proof: letting $X := \mathcal{D}\,P$,

$$\boxed{\begin{array}{l}
\exists\, x : X \,.\, \forall\, y : X \,.\, P\,x \;\equiv\; P\,y \\
\equiv \langle\text{Mutual implic.}\rangle\;\; \exists\, x : X \,.\, \forall\, y : X \,.\, (P\,x \Rightarrow P\,y) \wedge (P\,y \Rightarrow P\,x) \\
\equiv \langle\text{Distribut. } \forall/\wedge\rangle\;\; \exists\, x : X \,.\, \forall\,(y : X \,.\, P\,x \Rightarrow P\,y) \wedge \forall\,(y : X \,.\, P\,y \Rightarrow P\,x) \\
\equiv \langle\text{Lft distr. } \Rightarrow/\forall\rangle\;\; \exists\, x : X \,.\, (P\,x \Rightarrow \forall\, P) \wedge \forall\,(y : X \,.\, P\,y \Rightarrow P\,x) \\
\equiv \langle\text{Rght dst. } \Rightarrow/\exists\rangle\;\; \exists\, x : X \,.\, (P\,x \Rightarrow \forall\, P) \wedge (\exists\, P \Rightarrow P\,x) \\
\equiv \langle\text{Property below}\rangle\;\; \exists\, x : X \,.\, (P\,x \wedge \forall\, P) \vee \neg\,(\exists\, P \vee P\,x) \\
\equiv \langle\text{Distribut. } \exists/\vee\rangle\;\; \exists\,(x : X \,.\, P\,x \wedge \forall\, P) \vee \exists\,(x : X \,.\, \neg\,(\exists\, P \vee P\,x)) \\
\equiv \langle\text{Dual } \forall/\exists \text{ (20)}\rangle\;\; \exists\,(x : X \,.\, P\,x \wedge \forall\, P) \vee \neg\,(\forall\, x : X \,.\, \exists\, P \vee P\,x) \\
\equiv \langle\text{Distribut. } \vee/\forall\rangle\;\; \exists\,(x : X \,.\, P\,x \wedge \forall\, P) \vee \neg\,(\exists\, P \vee \forall\, P) \\
\equiv \langle\text{Distribut. } \wedge/\exists\rangle\;\; (\exists\, P \wedge \forall\, P) \vee \neg\,(\exists\, P \vee \forall\, P) \\
\equiv \langle\text{Property below}\rangle\;\; (\exists\, P \Rightarrow \forall\, P) \wedge (\forall\, P \Rightarrow \exists\, P) \\
\equiv \langle\text{Mutual implic.}\rangle\;\; \exists\, P \;\equiv\; \forall\, P
\end{array}}$$

Referenced property: $\boxed{(p \Rightarrow q) \wedge (r \Rightarrow p) \;\equiv\; (p \wedge q) \vee \neg\,(r \vee p)}$.

Proof: on sight, using Shannon.

d. Our proof from lemmata for another, more important property

- Our proof for $\boxed{\exists P \equiv \forall P \equiv \exists x \,{:}\, \mathcal{D}\,P \,.\, \forall y \,{:}\, \mathcal{D}\,P \,.\, P\,x \equiv P\,y}$ is

$$\boxed{\begin{aligned} \exists P \equiv \forall P \quad &\equiv \quad \langle\text{Mut. implic.}\rangle \quad (\forall P \Rightarrow \exists P) \wedge (\exists P \Rightarrow \forall P) \\ &\equiv \quad \langle\text{Lemma A, B}\rangle \quad \mathcal{D}\,P \neq \emptyset \wedge \text{con}\,P \\ &\equiv \quad \langle\text{Lemma C}\rangle \quad \exists x \,{:}\, \mathcal{D}\,P \,.\, \forall y \,{:}\, \mathcal{D}\,P \,.\, P\,x \equiv P\,y \end{aligned}}$$

Here are the lemmata

$$\boxed{\begin{aligned} &\text{LEMMA A:} \quad \forall P \Rightarrow \exists P \equiv \mathcal{D}\,P \neq \emptyset \\ &\text{LEMMA B:} \quad \exists P \Rightarrow \forall P \equiv \text{con}\,P \\ &\text{LEMMA C:} \quad \mathcal{D}\,f \neq \emptyset \wedge \text{con}\,f \equiv \exists x \,{:}\, \mathcal{D}\,f \,.\, \forall y \,{:}\, \mathcal{D}\,f \,.\, f\,x = f\,y \end{aligned}}$$

- Context: con is defined by $\boxed{\text{con}\,f \equiv \forall x \,{:}\, \mathcal{D}\,f \,.\, \forall y \,{:}\, \mathcal{D}\,f \,.\, f\,x = f\,y}$

  We wanted to prove the *Constant Function Theorem*

$$\boxed{\text{THEOREM:} \quad \text{con}\,f \equiv \mathcal{D}\,f = \emptyset \vee \exists x \,{:}\, \mathcal{D}\,f \,.\, f = \mathcal{D}\,f^{\,\bullet}\,f\,x.}$$

This is a consequence of Lemma C (exercise).

- Proofs of the lemmata

$\boxed{\textsc{Lemma a:}\ \forall\,P \Rightarrow \exists\,P \;\equiv\; \mathcal{D}\,P \neq \emptyset}$ Proof:

$$\begin{aligned}
\forall\,P \Rightarrow \exists\,P \;&\equiv\; \langle\text{From} \Rightarrow \text{to} \vee\rangle && \neg\forall\,P \vee \exists\,P\\
&\equiv\; \langle\text{Duality (20)}\rangle && \exists\,(\neg\,P) \vee \exists\,P\\
&\equiv\; \langle\text{Distrib. } \exists/\vee\rangle && \exists\,(\neg\,P \mathbin{\widehat{\vee}} P)\\
&\equiv\; \langle\text{Excl. middle}\rangle && \exists\,(\mathcal{D}\,P\,{}^\bullet 1)\\
&\equiv\; \langle\text{Const. pred.}\rangle && \mathcal{D}\,P \neq \emptyset
\end{aligned}$$

$\boxed{\textsc{Lemma b:}\ \exists\,P \Rightarrow \forall\,P \;\equiv\; \text{con}\,P}$ Proof:

$$\begin{aligned}
&\exists\,P \Rightarrow \forall\,P\\
&\equiv\; \langle\text{Rght-dst. } \Rightarrow/\exists\rangle \quad \forall\,x\,{:}\,\mathcal{D}\,P\,.\,P\,x \Rightarrow \forall\,P\\
&\equiv\; \langle\text{Left-dstr. } \Rightarrow/\forall\rangle \quad \forall\,x\,{:}\,\mathcal{D}\,P\,.\,\forall\,y\,{:}\,\mathcal{D}\,P\,.\,P\,x \Rightarrow P\,y\\
&\equiv\; \langle\text{Remark below}\rangle \quad \forall\,x\,{:}\,\mathcal{D}\,P\,.\,\forall\,y\,{:}\,\mathcal{D}\,P\,.\,P\,x \equiv P\,y\\
&\equiv\; \langle\text{Definition con}\rangle \quad \text{con}\,P
\end{aligned}$$

Remark: uneventful proof steps not written out are: duplicate by idempotency of $\wedge$; rename dummies $x, y := y, x$ and swap $\forall$'s in one copy; combine copies by (D $\forall/\wedge$), and obtain $P\,x \equiv P\,y$ by mutual implication.

$\boxed{\textsc{Lemma c:}\ \ \mathcal{D}\,f \neq \emptyset \wedge \mathsf{con}\,f \ \equiv\ \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\, y \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y}$ Proof:

$$
\begin{array}{lll}
\multicolumn{3}{l}{\exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\, y \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y} \\
\quad\equiv & \langle \text{Idempot. } \wedge \rangle & \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\,(y \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y) \wedge \forall\,(z \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,z) \\
\quad\equiv & \langle \text{Distrib. } \wedge/\forall \rangle & \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\,(y \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y \wedge \forall\, z \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,z) \\
\quad\equiv & \langle \text{Distrib. } \wedge/\forall \rangle & \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\, y \mathbin{:} \mathcal{D}\,f \,.\, \forall\, z \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y \wedge f\,x = f\,z \\
\quad\Rightarrow & \langle \text{Transitiv. } = \rangle & \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\, y \mathbin{:} \mathcal{D}\,f \,.\, \forall\, z \mathbin{:} \mathcal{D}\,f \,.\, f\,y = f\,z \\
\quad\equiv & \langle \text{Definit. con} \rangle & \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \mathsf{con}\,f \\
\quad\equiv & \langle \text{Const. pred.} \rangle & \mathcal{D}\,f \neq \emptyset \wedge \mathsf{con}\,f \\
\\
\multicolumn{3}{l}{\mathcal{D}\,f \neq \emptyset \wedge \mathsf{con}\,f} \\
\quad\equiv & \langle \text{Definit. con} \rangle & \mathcal{D}\,f \neq \emptyset \wedge \forall\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\, y \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y \\
\quad\Rightarrow & \langle \text{Weakening } \textsc{a} \rangle & \exists\, x \mathbin{:} \mathcal{D}\,f \,.\, \forall\, y \mathbin{:} \mathcal{D}\,f \,.\, f\,x = f\,y
\end{array}
$$

The weakening of lemma $\textsc{a}$ is $\mathcal{D}\,P \neq \emptyset \Rightarrow \forall\,P \Rightarrow \exists\,P$,
hence $\mathcal{D}\,P \neq \emptyset \wedge \forall\,P \Rightarrow \exists\,P$.

9.0.1 The function range, with an application to set comprehension

The function range

a. Axiomatic definition of the *function range* operator $\mathcal{R}$

AXIOM, FUNCTION RANGE: $e \in \mathcal{R}\, f \;\equiv\; \exists\, (x \,:\, \mathcal{D}\, f \,.\, f\, x = e)$ (50)

Equivalently, in point-free style: $e \in \mathcal{R}\, f \;\equiv\; \exists\, (f \overset{\leftharpoonup}{=} e)$.

Examples: proving $\forall\, x \,:\, \mathcal{D}\, f \,.\, f\, x \in \mathcal{R}\, f$ and $\mathcal{R}\, f \subseteq Y \;\equiv\; \forall\, x \,:\, \mathcal{D}\, f \,.\, f\, x \in Y$, given that $\subseteq$ is defined by $Z \subseteq Y \;\equiv\; \forall\, z \,:\, Z \,.\, z \in Y$ (exercises).

Consequence for the function arrow: $f \in X \to Y \;\equiv\; \mathcal{D}\, f = X \wedge \mathcal{R}\, f \subseteq Y$.

b. A very useful theorem (point-free variant of "change of quantified variables").

THEOREM, COMPOSITION RULE

FOR $\forall$: $\forall\, P \Rightarrow \forall\, (P \circ f)$ and $\mathcal{D}\, P \subseteq \mathcal{R}\, f \Rightarrow \forall\, (P \circ f) \Rightarrow \forall\, P$ (51)

FOR $\exists$: $\exists\, (P \circ f) \Rightarrow \exists\, P$ and $\mathcal{D}\, P \subseteq \mathcal{R}\, f \Rightarrow \exists\, P \Rightarrow \exists\, (P \circ f)$ (52)

PROOF: We prove the part for $\forall$, the rest follows by duality.

Proof for $\boxed{\text{(i) } \forall\, P \Rightarrow \forall\,(P \circ f) \quad \text{and} \quad \text{(ii) } \mathcal{D}\, P \subseteq \mathcal{R}\, f \Rightarrow (\forall\,(P \circ f) \equiv \forall\, P)}$

We preserve equivalence as long as possible, factoring out a common part.

$$\boxed{\begin{array}{lccl}
\forall\,(P \circ f) & \equiv & \langle \text{Definition } \circ \rangle & \forall\, x : \mathcal{D}\, f \wedge f\, x \in \mathcal{D}\, P \,.\, P\,(f\, x) \\
 & \equiv & \langle \text{Trading sub } \forall \rangle & \forall\, x : \mathcal{D}\, f \,.\, f\, x \in \mathcal{D}\, P \Rightarrow P\,(f\, x) \\
 & \equiv & \langle \text{One-point rule} \rangle & \forall\, x : \mathcal{D}\, f \,.\, \forall\, y : \mathcal{D}\, P \,.\, y = f\, x \Rightarrow P\, y \\
 & \equiv & \langle \text{Swap under } \forall \rangle & \forall\, y : \mathcal{D}\, P \,.\, \forall\, x : \mathcal{D}\, f \,.\, y = f\, x \Rightarrow P\, y \\
 & \equiv & \langle \text{R-dstr. } \Rightarrow/\exists \rangle & \forall\, y : \mathcal{D}\, P \,.\, \exists\,(x : \mathcal{D}\, f \,.\, y = f\, x) \Rightarrow P\, y \\
 & \equiv & \langle \text{Definition } \mathcal{R} \rangle & \forall\, y : \mathcal{D}\, P \,.\, y \in \mathcal{R}\, f \Rightarrow P\, y
\end{array}}$$

Hence $\forall\,(P \circ f) \equiv \forall\, y : \mathcal{D}\, P \,.\, y \in \mathcal{R}\, f \Rightarrow P\, y$.

Proof for part (i)

$$\boxed{\begin{array}{lccl}
\forall\,(P \circ f) & \equiv & \langle \text{Common part} \rangle & \forall\, y : \mathcal{D}\, P \,.\, y \in \mathcal{R}\, f \Rightarrow P\, y \\
 & \Leftarrow & \langle p \Rightarrow q \Rightarrow p \rangle & \forall\, y : \mathcal{D}\, P \,.\, P\, y
\end{array}}$$

Proof for part (ii): assume $\mathcal{D}\, P \subseteq \mathcal{R}\, f$, that is: $\forall\, y : \mathcal{D}\, P \,.\, y \in \mathcal{R}\, f$.

$$\boxed{\begin{array}{lccl}
\forall\,(P \circ f) & \equiv & \langle \text{Common part} \rangle & \forall\, y : \mathcal{D}\, P \,.\, y \in \mathcal{R}\, f \Rightarrow P\, y \\
 & \equiv & \langle \text{Assumption} \rangle & \forall\, y : \mathcal{D}\, P \,.\, P\, y
\end{array}}$$

c. Remarks on the point-wise form of the composition theorem (51)

- $\mathcal{D}\,P \subseteq \mathcal{R}\,f \Rightarrow (\forall\,(P \circ f) \equiv \forall\,P)$ can be written

$$\boxed{\forall\,(y\!:\!Y\,.\,P\,y) \equiv \forall\,x\!:\!X\,.\,P\,(f\,x)}$$

  provided $Y \subseteq \mathcal{D}\,P$ and $X \subseteq \mathcal{D}\,f$ and $Y = \mathcal{R}\,(f \rceil X)$

- Another form is the “dummy change” rule

$$\boxed{\forall\,(y\!:\!Y\,.\,p) \equiv \forall\,(x\!:\!X\,.\,p[^{y}_{f\,x})}$$

  provided $Y \subseteq \mathrm{D}_p^y$ and $X \subseteq \mathcal{D}\,f$ and $Y = \mathcal{R}\,(f \rceil X)$.

Set comprehension with proper bindings and calculation rules

a. Convention: $\boxed{\text{we introduce } \{\!\!-\!\!\} \text{ as an operator } \textit{fully interchangeable} \text{ with } \mathcal{R}}$.

Immediate consequences:

- Formalizes familiar expressions with their expected meaning but without their defects (ambiguity, no formal calculation rules).

  Examples: $\{2,3,5\}$ and $Even = \{m : \mathbb{Z} \,.\, 2 \cdot m\}$

  Notes: tuples are functions, so $\{e, e', e''\}$ denotes a set by its elements. Also, $k \in \{m : \mathbb{Z} \,.\, 2 \cdot m\} \equiv \exists\, m : \mathbb{Z} \,.\, k = 2 \cdot m$ by the range axiom (50).

- The only "custom" *to be discarded* is using $\{\}$ for singletons.

  No loss: preservation would violate Leibniz's principle, e.g.,

  $f = a, b \Rightarrow \{f\} = \{a, b\}$.

  Note: $f = a, b \Rightarrow \{f\} = \{a, b\}$ is fully consistent in our formalism. Yet:

- To avoid baffling the uninitiated: write $\mathcal{R}\, f$, not $\{f\}$, if f is an operator.

  For singleton sets, always use ι, as in $\iota\, 3$.

b. Convention (to cover common forms, without flaws) variants for abstraction

$$\boxed{\begin{array}{lll} e \mid x : X & \text{stands for} & x : X \,.\, e \\ x : X \mid p & \text{stands for} & x : X \mathrel{\underset{\cdot}{\wedge}} p \,.\, x \end{array}}$$

Immediate consequences

- Formalizes expressions like $\{2 \cdot m \mid m : \mathbb{Z}\}$ and $\{m : \mathbb{N} \mid m < n\}$.
- Now binding is always trouble-free, even in examples such as (exercise)

$$\boxed{\begin{array}{lcl} \{n : \mathbb{Z} \mid n \in \mathit{Even}\} & = & \{n : \mathit{Even} \mid n \in \mathbb{Z}\} \\ \{n \in \mathbb{Z} \mid n : \mathit{Even}\} & \neq & \{n \in \mathit{Even} \mid n : \mathbb{Z}\} \end{array}}$$

- All calculation rules follow from predicate calculus by the axiom for $\mathcal{R}$.
- A frequent pattern is captured by the following property

$$\boxed{\textsc{Theorem, Set comprehension:}\quad e \in \{x : X \mid p\} \equiv e \in X \wedge p[^x_e \qquad (53)}$$

$$\begin{array}{lll} \textsc{Proof:}\quad e \in \{x : X \mathrel{\underset{\cdot}{\wedge}} p \,.\, x\} & \equiv \langle \text{Function range (50)} \rangle & \exists\, x : X \mathrel{\underset{\cdot}{\wedge}} p \,.\, x = e \\ & \equiv \langle \text{Trading, one-pt rule} \rangle & e \in X \wedge p[^x_e \end{array}$$

c. A few other calculation examples.

Seeing $\{n:\mathbb{Z} \mid n/2 \in \mathbb{Z}\}$ or $\{n:\mathbb{Z} \mid \exists\, m:\mathbb{Z}\,.\,2\cdot m = n\}$ or $\{2\cdot m \mid m:\mathbb{Z}\}$ as equivalent expressions for the set of even numbers may seem "obvious". However, in a formal setting a proof is required. It is also quite illustrative.

$$
\boxed{
\begin{array}{lll}
\multicolumn{3}{l}{k \in \{n:\mathbb{Z} \mid n/2 \in \mathbb{Z}\}} \\
\quad\equiv & \langle\text{Comprehension (53)}\rangle & k \in \mathbb{Z} \wedge k/2 \in \mathbb{Z} \\
\quad\equiv & \langle\text{One-point rule for } \exists\rangle & k \in \mathbb{Z} \wedge \exists\, m:\mathbb{Z}\,.\,m = k/2 \\
\quad\equiv & \langle m = k/2 \;\equiv\; 2\cdot m = k\rangle & k \in \mathbb{Z} \wedge \exists\, m:\mathbb{Z}\,.\,2\cdot m = k \;\;\text{(*)} \\
\quad\equiv & \langle\text{Comprehension (53)}\rangle & k \in \{n:\mathbb{Z} \mid \exists\, m:\mathbb{Z}\,.\,n = 2\cdot m\} \\
\multicolumn{3}{l}{k \in \{n:\mathbb{Z} \mid n/2 \in \mathbb{Z}\}} \\
\quad\equiv & \langle\text{From (*), distr. } \wedge/\exists\rangle & \exists\, m:\mathbb{Z}\,.\,2\cdot m = k \wedge k \in \mathbb{Z} \\
\quad\equiv & \langle m \in \mathbb{Z} \Rightarrow 2\cdot m \in \mathbb{Z}\rangle & \exists\, m:\mathbb{Z}\,.\,2\cdot m = k \\
\quad\equiv & \langle\text{Function range (50)}\rangle & k \in \{m:\mathbb{Z}\,.\,2\cdot m\}
\end{array}
}
$$

Note that $m = k/2 \;\equiv\; k = 2\cdot m$ holds in the context $k \in \mathbb{Z} \wedge m \in \mathbb{Z}$.
Technicality: in our calculus, all expressions are "guarded".

A similar but subtler example (also used later): the correspondence between

- the set $\{n^- \mid n:\mathbb{N}_{\neq 0}\}$ and
- the predicate $P:\mathbb{R}\rightarrow\mathbb{B}$ with $P\,x \equiv x \neq 0 \wedge x^- \in \mathbb{N}$

Assume $—^-:\mathbb{R}_{\neq 0}\rightarrow\mathbb{R}_{\neq 0}$ and property $x \in \mathbb{R}_{\neq 0} \Rightarrow (x^-)^- = x$ for the multiplicative inverse.

Calculation:

$$
\begin{array}{lcl}
\multicolumn{3}{l}{x \in \{n^- \mid n:\mathbb{N}_{\neq 0}\}} \\
\equiv & \langle\text{Function range (50)}\rangle & \exists\, n:\mathbb{N}_{\neq 0} \,.\, x = n^- \\
\equiv & \langle n \in \mathbb{R}_{\neq 0} \Rightarrow n^- \in \mathbb{R}_{\neq 0}\rangle & \exists\, n:\mathbb{N}_{\neq 0} \,.\, x = n^- \wedge n^- \in \mathbb{R}_{\neq 0} \\
\equiv & \langle e = e' \wedge p[^v_e \equiv e = e' \wedge p[^v_{e'}\rangle & \exists\, n:\mathbb{N}_{\neq 0} \,.\, x = n^- \wedge x \in \mathbb{R}_{\neq 0} \\
\equiv & \langle\text{Leibniz } (\Leftrightarrow)\text{, stated prop.}\rangle & \exists\, n:\mathbb{N}_{\neq 0} \,.\, x^- = n \wedge x \in \mathbb{R}_{\neq 0} \\
\equiv & \langle\text{One-point rule for } \exists\rangle & x \in \mathbb{R}_{\neq 0} \wedge x^- \in \mathbb{N}_{\neq 0} \\
\equiv & \langle a \in \mathbb{N}_{\neq 0} \equiv a \in \mathbb{N} \wedge a \neq 0\rangle & x \in \mathbb{R}_{\neq 0} \wedge x^- \in \mathbb{N} \wedge x^- \neq 0 \\
\equiv & \langle x \in \mathbb{R}_{\neq 0} \Rightarrow x^- \neq 0\rangle & x \in \mathbb{R} \wedge x \neq 0 \wedge x^- \in \mathbb{N} \\
\equiv & \langle\text{Definition of } P\rangle & x \in \mathcal{D}\,P \wedge P\,x
\end{array}
$$

9.1 Exogenous application examples

9.1.0 Limits in mathematical analysis

"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

$$\text{R. c. L. 2 p. di m. 11 L} \qquad \text{for our} \qquad 3\sqrt{2+11i}.$$

Many professional mathematicians to this day use the quantifiers ($\forall$,$\exists$) in a similar fashion,

$$\exists\delta > 0 \text{ s.t. } |f(x) - f(x_0)| < \epsilon \text{ if } |x - x_0| < \delta, \text{ for all } \epsilon > 0,$$

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")

a. Preliminary concept: *adherence* [Lang], here presented formally.

Definition, Adherence (as a set transformer):

$$\mathbf{def}\ \mathsf{Ad} : \mathcal{P}\,\mathbb{R} \to \mathcal{P}\,\mathbb{R}\ \mathbf{with}\ v \in \mathsf{Ad}\,S \equiv \forall\,\epsilon : \mathbb{R}_{>0} \,.\, \exists\,x : S \,.\, |x - v| < \epsilon \qquad (54)$$

b. Limits: a typical informal definition

Let S be a set of numbers and a be adherent to S. Let f be a function defined on S. We shall say that the **limit of** $f(x)$ **as** x **approaches** a **exists,** if there exists a number L having the following property. Given ϵ, there exists a number $\delta > 0$ such that for all $x \in S$ satisfying $|x - a| < \delta$ we have $|f(x) - L| < \epsilon$. If that is the case, we write

$$\lim_{\substack{x \to a \\ x \in S}} f(x) = L.$$

Serge Lang, *Undergraduate Analysis*

c. Limits: a typical informal proof

Proposition 2.1. *Let S be a set of numbers, and assume that a is adherent to S. Let S' be a subset of S, and assume that a is also adherent to S'. Let f be a function defined on S.*
If $\displaystyle\lim_{\substack{x \to a \\ x \in S}} f(x)$ exists, then $\displaystyle\lim_{\substack{x \to a \\ x \in S'}} f(x)$ also exists, and those limits are equal.
In particular, if the limit exists, it is unique.

Proof. Let L be the first limit. Given ϵ, there exists δ such that whenever $x \in S$ and $|x - a| < \delta$ we have $|f(x) - L| < \epsilon/2$. This applies a fortiori when $x \in S'$, so that L is also the limit for $x \in S'$. If M is also a limit, there exists δ_1 such that whenever $x \in S$ and $|x - a| < \delta_1$ then $|f(x) - M| < \epsilon/2$. If $|x - a| < \min(\delta, \delta_1)$ and $x \in S$, then

$$|L - M| \leq |L - f(x)| + |f(x) - M| < \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.$$

Hence $|L - M|$ is less than any ϵ, and it follows that $|L - M| = 0$, whence $L = M$.”

d. Critique of the informal treatment

- Obvious critique: see quotation from Taylor: obscures proof structure
- Additionally: the poor custom, often found in many mathematics texts, of writing an equality when the underlying concept is actually a relation. Example here: writing $\lim\limits_{\substack{x \to a \\ x \in S}} f(x) = L$ *before* uniqueness is proven.

  Danger: confuses the argument and is error-prone.

  E.g., it suggests fallacious uniqueness proofs like:

  Let L and M satisfy $\lim\limits_{\substack{x \to a \\ x \in S}} f(x) = L$ and $\lim\limits_{\substack{x \to a \\ x \in S}} f(x) = M$, then $L = M$ by transitivity. *Wrong!*

Such flaws are avoided simply by a proper relational formulation, which naturally arises in a formal setting.

e. A proper formalization

Define a relation islim_f (parametrized by any function $f : \mathbb{R} \not\rightarrow \mathbb{R}$) between $\mathbb{R}$ and the set of points adherent to $\mathcal{D}\, f$, that is:

$\mathsf{islim}_f \in \mathbb{R} \times \mathsf{Ad}\,(\mathcal{D}\, f) \to \mathbb{B}$ and, for the mapping,

$$L \,\mathsf{islim}_f\, a \equiv \forall\, \epsilon : \mathbb{R}_{>0} \,.\, \exists\, \delta : \mathbb{R}_{>0} \,.\, \forall\, x : \mathcal{D}\, f \,.\, |x - a| < \delta \Rightarrow |f\, x - L| < \epsilon \qquad (55)$$

Affix convention chosen s.t. $L\,\mathsf{islim}_f\, a$ is read "L is a limit for f in a".

Domain modulation (via $\mathcal{D}\, f$) subsumes S in the conventional formulation.

f. "Proposition 2.1", formalized

Theorem: for any function $f : \mathbb{R} \not\rightarrow \mathbb{R}$, any subset S of $\mathcal{D}\, f$ and any a adherent to S,

(i) $\exists\, (L : \mathbb{R} \,.\, L\,\mathsf{islim}_f\, a) \Rightarrow \exists\, (L : \mathbb{R} \,.\, L\,\mathsf{islim}_{f \rceil S}\, a)$,

(ii) $\forall\, L : \mathbb{R} \,.\, \forall\, M : \mathbb{R} \,.\, L\,\mathsf{islim}_f\, a \wedge M\,\mathsf{islim}_{f \rceil S}\, a \Rightarrow L = M$.

Remark: one must first prove $a \in \mathsf{Ad}\,(\mathcal{D}\,f)$ and $a \in \mathsf{Ad}\,(\mathcal{D}\,f \rceil S)$ to ensure that the image definition of islim holds for all expressions of interest.

This proof, and the proof of (i), are routine applications of monotonicity. As a hint for (i), first prove $\forall\, L : \mathbb{R} \,.\, L \,\mathsf{islim}_f\, a \Rightarrow L \,\mathsf{islim}_{f \rceil S}\, a$.

The calculational proof of (ii) is very interesting: (a) clear explicit structure, (b) role of adherence, not even mentioned in the conventional proof, emerges.

Here is the proof: for arbitrary a in $\mathsf{Ad}\,S$, L and M in $\mathbb{R}$, and using the abbreviation $b\,R\,\delta \;\equiv\; \forall\, x : S \,.\, |x - a| < \delta \Rightarrow |f\,x - b| < \epsilon$ (given a and ϵ),

$$
\begin{array}{lcl}
\multicolumn{3}{l}{L \,\mathsf{islim}_f\, a \wedge M \,\mathsf{islim}_{f \rceil S}\, a} \\
\Rightarrow & \langle \text{Hint given for (i)} \rangle & L \,\mathsf{islim}_{f \rceil S}\, a \wedge M \,\mathsf{islim}_{f \rceil S}\, a \\
\equiv & \langle \text{(55)}, \mathcal{D}\,(f \rceil S) = S \rangle & \forall\,(\epsilon : \mathbb{R}_{>0} \,.\, \exists\,\delta : \mathbb{R}_{>0} \,.\, L\,R\,\delta) \wedge \forall\,(\epsilon : \mathbb{R}_{>0} \,.\, \exists\,\delta : \mathbb{R}_{>0} \,.\, M\,R\,\delta) \\
\equiv & \langle \text{Distributivity } \forall/\wedge \rangle & \forall\,\epsilon : \mathbb{R}_{>0} \,.\, \exists\,(\delta : \mathbb{R}_{>0} \,.\, L\,R\,\delta) \wedge \exists\,(\delta : \mathbb{R}_{>0} \,.\, M\,R\,\delta) \\
\equiv & \langle \text{Rename, dstr. } \wedge/\exists \rangle & \forall\,\epsilon : \mathbb{R}_{>0} \,.\, \exists\,\delta : \mathbb{R}_{>0} \,.\, \exists\,\delta' : \mathbb{R}_{>0} \,.\, L\,R\,\delta \wedge M\,R\,\delta' \\
\Rightarrow & \langle \text{Closeness lemma} \rangle & \forall\,\epsilon : \mathbb{R}_{>0} \,.\, \exists\,\delta : \mathbb{R}_{>0} \,.\, \exists\,\delta' : \mathbb{R}_{>0} \,.\, a \in \mathsf{Ad}\,S \Rightarrow |L - M| < 2 \cdot \epsilon \\
\equiv & \langle \text{Hypoth. } a \in \mathsf{Ad}\,S \rangle & \forall\,\epsilon : \mathbb{R}_{>0} \,.\, \exists\,\delta : \mathbb{R}_{>0} \,.\, \exists\,\delta' : \mathbb{R}_{>0} \,.\, |L - M| < 2 \cdot \epsilon \\
\equiv & \langle \text{Const. pred. sub } \exists \rangle & \forall\,\epsilon : \mathbb{R}_{>0} \,.\, |L - M| < 2 \cdot \epsilon \\
\equiv & \langle \text{Vanishing lemma} \rangle & L - M = 0 \\
\equiv & \langle \text{Leibniz, group } + \rangle & L = M
\end{array}
$$

For the sake of completeness, here are the auxiliary lemmata.

$$\boxed{\textsc{Closeness lemma:}\quad L\,R\,\delta \wedge M\,R\,\delta' \Rightarrow a \in \mathsf{Ad}\,S \Rightarrow |L - M| < 2\cdot\epsilon}$$

Proof:

$$
\begin{array}{lll}
\multicolumn{3}{l}{L\,R\,\delta \wedge M\,R\,\delta'} \\
\equiv & \langle\text{Expanding } b\,R\,\delta\rangle & \forall\,(x : S\,.\,|x - a| < \delta \Rightarrow |f\,x - L| < \epsilon) \wedge \\
 & & \forall\,(x : S\,.\,|x - a| < \delta' \Rightarrow |f\,x - M| < \epsilon) \\
\equiv & \langle\text{Distributiv. } \forall/\wedge\rangle & \forall\,x : S\,.\,(|x - a| < \delta \Rightarrow |f\,x - L| < \epsilon) \\
 & & \qquad\quad \wedge\ \ (|x - a| < \delta' \Rightarrow |f\,x - M| < \epsilon) \\
\Rightarrow & \langle\text{Monotonic. } \wedge/\Rightarrow\rangle & \forall\,x : S\,.\,|x - a| < \delta \wedge |x - a| < \delta' \\
 & & \qquad \Rightarrow\ \ |f\,x - L| < \epsilon \wedge |f\,x - M| < \epsilon \\
\equiv & \langle\text{Def. } \curlywedge,\ |-x| = |x|\rangle & \forall\,x : S\,.\,|x - a| < \delta \curlywedge \delta' \Rightarrow |L - f\,x| < \epsilon \wedge |f\,x - M| < \epsilon \\
\Rightarrow & \langle\text{Monotonic. } +/<\rangle & \forall\,x : S\,.\,|x - a| < \delta \curlywedge \delta' \Rightarrow |L - f\,x| + |f\,x - M| < 2\cdot\epsilon \\
\Rightarrow & \langle\text{Triangle inequal.}\rangle & \forall\,x : S\,.\,|x - a| < \delta \curlywedge \delta' \Rightarrow |L - M| < 2\cdot\epsilon \\
\equiv & \langle\text{R-distribut. } \Rightarrow/\exists\rangle & \exists\,(x : S\,.\,|x - a| < \delta \curlywedge \delta') \Rightarrow |L - M| < 2\cdot\epsilon \\
\Rightarrow & \langle\text{Def. Ad, } \delta \curlywedge \delta' > 0\rangle & a \in \mathsf{Ad}\,S \Rightarrow |L - M| < 2\cdot\epsilon
\end{array}
$$

The *vanishing lemma* formalizes and generalizes the intuitive argument "Hence $|L - M|$ is less than any ϵ, and it follows that $|L - M| = 0$".

$$\boxed{\textsc{Vanishing lemma:}\quad \forall\, c : \mathbb{R}_{>0} \,.\, \forall\, x : \mathbb{R} \,.\, \forall\, (\epsilon : \mathbb{R}_{>0} \,.\, |x| < c \cdot \epsilon) \Rightarrow x = 0}$$

Proof: for any $c : \mathbb{R}_{>0}$ and $x : \mathbb{R}$,

$$\begin{array}{lcl}
\multicolumn{3}{l}{\forall\, \epsilon : \mathbb{R}_{>0} \,.\, |x| < c \cdot \epsilon} \\
\equiv & \langle \text{Trading under } \forall \rangle & \forall\, \epsilon : \mathbb{R} \,.\, \epsilon > 0 \Rightarrow |x| < c \cdot \epsilon \\
\Rightarrow & \langle \text{Instant. } \epsilon := |x|/c \rangle & |x|/c \in \mathbb{R} \Rightarrow |x|/c > 0 \Rightarrow |x| < c \cdot |x|/c \\
\equiv & \langle \text{Assump., arithm.} \rangle & |x| > 0 \Rightarrow |x| < |x| \\
\equiv & \langle y = y \Rightarrow \neg\, (y < y) \rangle & \neg\, (|x| > 0) \\
\equiv & \langle \neg\, (|x| > 0) \;\equiv\; x = 0 \rangle & x = 0
\end{array}$$

Note how the calculational style exposes hidden facts, mostly coinciding with the axioms of a *norm* [Lang]: for real a and vectors x, y (numbers for 1D)

(i) $0 \leq |x|$ and $|x| = 0 \;\equiv\; x = 0$,

(ii) $|a \cdot x| = |a| \cdot |x|$,

(iii) $|x + y| \leq |x| + |y|$.

g. Wrapping up the limit example

- The limit operator lim: for any argument $f : \mathbb{R} \nrightarrow \mathbb{R}$, lim f is defined by

$$\textsc{Definition:}\quad \begin{array}{l} \lim f \in \{a : \mathsf{Ad}\,(\mathcal{D}\, f) \mid \exists\, b : \mathbb{R} \,.\, b\, \mathsf{islim}_f\, a\} \to \mathbb{R} \\ \forall\, a : \mathcal{D}\,(\lim f) \,.\, (\lim f\, a)\, \mathsf{islim}_f\, a \end{array} \qquad (56)$$

- Well-definedness follows from function comprehension and uniqueness.
- Observations
  - lim is a functional, not an ad hoc abstractor
  - It supports point-free expressions like lim $f\, a$.
  - Domain modulation covers left, right and two-sided limits, e.g., given

$$\boxed{\mathbf{def}\ f : \mathbb{R} \to \mathbb{R}\ \mathbf{with}\ f\, x = (x \geq 0)\,?\,0 \vdash x + 1}$$

  we have lim $f_{<0}\, 0 = 1$ and lim $f_{\geq 0}\, 0 = 0$, whereas $0 \notin \mathcal{D}\,(\lim f_{\leq 0})$.
  - Conventional notations by macros from lim, e.g., for real expression e,
  $\lim\limits_{x \to a} e$ stands for lim $(x : \mathbb{R} \,.\, e)\, a$ and
  $\lim\limits_{x \underset{<}{\to} a} e$ stands for lim $(x : \mathbb{R}_{<a} \,.\, e)\, a$

9.1.1 Adherence as a predicate transformer

a. Purpose: show analogy with formulations and reasoning in computer science.

Other advantage: predicate expressions more compact than set expressions

Principle: $\mathcal{P}\,X$ is isomorphic to $X \to \mathbb{B}$.

b. Alternative definition of adherence

Definition, Adherence (as a predicate transformer):

$$\textbf{def}\ \mathsf{ad} : (\mathbb{R} \to \mathbb{B}) \to (\mathbb{R} \to \mathbb{B})\ \textbf{with}$$
$$\mathsf{ad}\,P\,v \ \equiv\ \forall\,\epsilon : \mathbb{R}_{>0}\,.\,\exists\,x : \mathbb{R}_P\,.\,|x - v| < \epsilon \tag{57}$$

c. Illustration: usual concepts of open and closed sets as predicates on predicates.

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

give rise to interesting theorems exposing the relationship with adherence.

- LEMMA: $\forall P : \mathbb{R} \to \mathbb{B} \,.\, \forall v : \mathbb{R} \,.\, P\,v \Rightarrow \mathsf{ad}\,P\,v$

  Proof: easy.

- THEOREM, CLOSURE PROPERTY: $\mathsf{closed}\,P \;\equiv\; \mathsf{ad}\,P = P$

  Proof:

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

- **Theorem, Idempotency of adherence:** $\mathsf{ad} \circ \mathsf{ad} = \mathsf{ad}$

  Proof: by function equality and the definition of $\circ$, proving $\mathsf{ad}\,(\mathsf{ad}\,P)\,v \;\equiv\; \mathsf{ad}\,P\,v$ for arbitrary $v:\mathbb{R}$ suffices.

  Instantiating $P\,v \Rightarrow \mathsf{ad}\,P\,v$ yields $\mathsf{ad}\,P\,v \Rightarrow \mathsf{ad}\,(\mathsf{ad}\,P)\,v$.

  For the converse,

$$\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{ad}\,(\mathsf{ad}\,P)\,v} \\
\equiv & \langle\text{Definition ad}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}_{\mathsf{ad}\,P}\,.\,|x-v|<\epsilon \\
\equiv & \langle\text{Trading sub }\exists\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}\,.\,\mathsf{ad}\,P\,x \wedge |x-v|<\epsilon \\
\equiv & \langle\text{Definition ad}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}\,.\,\forall\,(\epsilon:\mathbb{R}\,.\,\exists\,z:\mathbb{R}_P\,.\,|z-x|<\epsilon) \wedge |x-v|<\epsilon \\
\Rightarrow & \langle\text{Instantiation}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}\,.\,\exists\,(z:\mathbb{R}_P\,.\,|z-x|<\epsilon) \wedge |x-v|<\epsilon \\
\equiv & \langle\text{Distrib. }\wedge/\exists\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}\,.\,\exists\,z:\mathbb{R}_P\,.\,|z-x|<\epsilon \wedge |x-v|<\epsilon \\
\equiv & \langle\text{Monot. }+/<\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}\,.\,\exists\,z:\mathbb{R}_P\,.\,|z-x|+|x-v|<2\cdot\epsilon \\
\Rightarrow & \langle\text{Triangle ineq.}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,x:\mathbb{R}\,.\,\exists\,z:\mathbb{R}_P\,.\,|z-v|<2\cdot\epsilon \\
\equiv & \langle\text{Dummy swap}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,z:\mathbb{R}_P\,.\,\exists\,x:\mathbb{R}\,.\,|z-v|<2\cdot\epsilon \\
\equiv & \langle\text{Const. pred.}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,z:\mathbb{R}_P\,.\,|z-v|<2\cdot\epsilon \\
\equiv & \langle\text{Dummy chng.}\rangle & \forall\,\epsilon:\mathbb{R}_{>0}\,.\,\exists\,z:\mathbb{R}_P\,.\,|z-v|<\epsilon \\
\equiv & \langle\text{Definition ad}\rangle & \mathsf{ad}\,P\,v.
\end{array}$$

9.1.2 Calculating with properties and operators regarding orderings

a. Conventions and definitions $\boxed{\mathsf{pred}_X := X \rightarrow \mathbb{B}}$ and $\boxed{\mathsf{rel}_X := X^2 \rightarrow \mathbb{B}}$

Potential properties over rel_X, formalizing each by a predicate $P : \mathsf{rel}_X \rightarrow \mathbb{B}$.

Characteristic	P	Image, i.e., $P\,R \equiv$ formula below
reflexive	Refl	$\forall\, x : X \,.\, x\,R\,x$
irreflexive	Irfl	$\forall\, x : X \,.\, \neg\,(x\,R\,x)$
symmetric	Symm	$\forall\,(x, y) : X^2 \,.\, x\,R\,y \Rightarrow y\,R\,x$
asymmetric	Asym	$\forall\,(x, y) : X^2 \,.\, x\,R\,y \Rightarrow \neg\,(y\,R\,x)$
antisymmetric	Ants	$\forall\,(x, y) : X^2 \,.\, x\,R\,y \Rightarrow y\,R\,x \Rightarrow x = y$
transitive	Trns	$\forall\,(x, y, z) : X^3 \,.\, x\,R\,y \Rightarrow y\,R\,z \Rightarrow x\,R\,z$
total	Totl	$\forall\,(x, y) : X^2 \,.\, x\,R\,y \vee y\,R\,x$
equivalence	EQ	Trns $R \wedge$ Refl $R \wedge$ Symm R
preorder	PR	Trns $R \wedge$ Refl R
partial order	PO	PR $R \wedge$ Ants R
total order	TO	PO $R \wedge$ Totl R
quasi order	QO	Trns $R \wedge$ Irfl R (also called strict p.o.)

b. Two formulations for extremal elements (note: we write $\prec$ rather than R)

- Chracterization by set-oriented formulation of type $\boxed{\mathsf{rel}_X \to X \times \mathcal{P}\,X \to \mathbb{B}}$

  Example: $\boxed{\mathit{ismin}\_\_ \ \text{with}\ x \mathbin{\mathit{ismin}_\prec} S \ \equiv\ x \in S \wedge \forall\, y : X \,.\, y \prec x \Rightarrow y \notin S}$

- Chracterization by predicate transformers of type $\boxed{\mathsf{rel}_X \to \mathsf{pred}_X \to \mathsf{pred}_X}$

Name	*Symbol*	*Type*: $\mathsf{rel}_X \to \mathsf{pred}_X \to \mathsf{pred}_X$. *Image*: below
Lower bound	lb	$\mathsf{lb}_\prec P\, x \ \equiv\ \forall\, y : X \,.\, P\, y \Rightarrow x \prec y$
Least	lst	$\mathsf{lst}_\prec P\, x \ \equiv\ P\, x \wedge \mathsf{lb}\, P\, x$
Minimal	min	$\mathsf{min}_\prec P\, x \ \equiv\ P\, x \wedge \forall\, y : X \,.\, y \prec x \Rightarrow \neg\, (P\, y)$
Upper bound	ub	$\mathsf{ub}_\prec P\, x \ \equiv\ \forall\, y : X \,.\, P\, y \Rightarrow y \prec x$
Greatest	gst	$\mathsf{gst}_\prec P\, x \ \equiv\ P\, x \wedge \mathsf{ub}\, P\, x$
Maximal	max	$\mathsf{max}_\prec P\, x \ \equiv\ P\, x \wedge \forall\, y : X \,.\, x \prec y \Rightarrow \neg\, (P\, y)$
Least ub	lub	$\mathsf{lub}_\prec \ =\ \mathsf{lst}_\prec \circ \mathsf{ub}_\prec$
Greatest lb	glb	$\mathsf{glb}_\prec \ =\ \mathsf{gst}_\prec \circ \mathsf{lb}_\prec$

This is the preferred formulation, used henceforth.

c. Familiarization properties (avoiding wrong connotations)

No element can be both minimal and least:

$$\boxed{\neg\,(\mathsf{min}_{\prec}\, P\, x \wedge \mathsf{lst}_{\prec}\, P\, x)}$$

Reflexivity precludes minimal elements:

$$\boxed{\mathsf{Refl}\ (\prec) \Rightarrow \neg\,(\mathsf{min}_{\prec}\, P\, x)}$$

However, it is easy to show

$$\boxed{\mathsf{min}_{\overline{\prec}} = \mathsf{lst}_{\prec}}$$

given the relation transformer

$$\boxed{\overline{\underline{\quad}} : \mathsf{rel}_X \to \mathsf{rel}_X \quad \text{with} \quad x \mathrel{\overline{\prec}} y \equiv \neg\,(y \prec x)}$$

Example: if X is the set $\mathbb{N}$ of natural numbers with $\leq$, then $\boxed{\mathsf{min}_{<} = \mathsf{lst}_{\leq}}$.

d. Calculational reasoning about extremal elements

DEFINITION, MONOTONICITY:

A predicate $P : \mathsf{pred}_X$ is *monotonic* w.r.t. a relation $\_\!\_ \prec \_\!\_ : \mathsf{rel}_X$ iff

$$\forall\,(x, y) : X^2 \,.\, x \prec y \Rightarrow P\,x \Rightarrow P\,y \qquad (58)$$

THEOREM, PROPERTIES OF EXTREMAL ELEMENTS:

For any $\_\!\_ \prec \_\!\_ : \mathsf{rel}_X$ and $P : \mathsf{pred}_X$,

(a) If $\prec$ is reflexive, then $\forall\,(y : X \,.\, x \prec y \Rightarrow P\,y) \Rightarrow P\,x$

(b) If $\prec$ is transitive, then $\mathsf{ub}_\prec\,P$ is monotonic w.r.t. $\prec$

(c) If P is monotonic w.r.t. $\prec$, then

$$\mathsf{lst}_\prec\,P\,x \;\equiv\; P\,x \wedge \forall\,(y : X \,.\, P\,y \;\equiv\; x \prec y)$$

(d) If $\prec$ is reflexive and transitive, then

$$\mathsf{lub}_\prec\,P\,x \;\equiv\; \forall\,(y : X \,.\, \mathsf{ub}\,P\,y \;\equiv\; x \prec y)$$

(e) If $\prec$ is antisymmetric, then

$$\mathsf{lst}_\prec\,P\,x \wedge \mathsf{lst}_\prec\,P\,y \Rightarrow x = y \text{ (uniqueness).} \qquad (59)$$

Replacing lb by ub and so on yields complementary theorems (straightforward).

Proof (or outline). For (a), instantiate the antecedent with $y := x$.

For (b), assume $\prec$ transitive and prove $x \prec y \Rightarrow \mathsf{ub}_\prec\, P\,x \Rightarrow \mathsf{ub}_\prec\, P\,y$ shunted.

$$\begin{array}{lcll}
\mathsf{ub}_\prec\, P\,x & \Rightarrow & \langle p \Rightarrow q \Rightarrow p\rangle & x \prec y \Rightarrow \mathsf{ub}_\prec\, P\,x \\
 & \equiv & \langle \text{Definition ub}\rangle & x \prec y \Rightarrow \forall\, z : X\,.\,P\,z \Rightarrow z \prec x \wedge 1 \\
 & \equiv & \langle p \Rightarrow e[^v_1 = e[^v_p\rangle & x \prec y \Rightarrow \forall\, z : X\,.\,P\,z \Rightarrow z \prec x \wedge x \prec y \\
 & \Rightarrow & \langle \text{Transitiv. } \prec\rangle & x \prec y \Rightarrow \forall\, z : X\,.\,P\,z \Rightarrow z \prec y \\
 & \equiv & \langle \text{Definition ub}\rangle & x \prec y \Rightarrow \mathsf{ub}_\prec\, P\,y
\end{array}$$

For (c), assume P monotonic and calculate $\mathsf{lst}_\prec\, P\,x$

$$\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{lst}_\prec\, P\,x} \\
\equiv & \langle \text{Defin. lst, lb}\rangle & P\,x \wedge \forall\, y : X\,.\,P\,y \Rightarrow x \prec y \\
\equiv & \langle \text{Modus Pon.}\rangle & P\,x \wedge (P\,x \Rightarrow \forall\, y : X\,.\,P\,y \Rightarrow x \prec y) \\
\equiv & \langle \text{L-dstr. } \Rightarrow/\forall\rangle & P\,x \wedge \forall\, y : X\,.\,P\,x \Rightarrow P\,y \Rightarrow x \prec y \\
\equiv & \langle \text{Monoton. } P\rangle & P\,x \wedge \forall\, y : X\,.\,(P\,x \Rightarrow P\,y \Rightarrow x \prec y) \wedge (P\,x \Rightarrow x \prec y \Rightarrow P\,y) \\
\equiv & \langle \text{L-dstr. } \Rightarrow/\wedge\rangle & P\,x \wedge \forall\, y : X\,.\,P\,x \Rightarrow (P\,y \Rightarrow x \prec y) \wedge (x \prec y \Rightarrow P\,y) \\
\equiv & \langle \text{Mut. implic.}\rangle & P\,x \wedge \forall\, y : X\,.\,P\,x \Rightarrow (P\,y \equiv x \prec y) \\
\equiv & \langle \text{L-dstr. } \Rightarrow/\forall\rangle & P\,x \wedge (P\,x \Rightarrow \forall\, y : X\,.\,P\,y \equiv x \prec y) \\
\equiv & \langle \text{Modus Pon.}\rangle & P\,x \wedge \forall\,(y : X\,.\,P\,y \equiv x \prec y)
\end{array}$$

Proofs (continued)

(d) is a direct consequence from the preceding parts.

(e) is proven in a simple but very typical way.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{lst}_{\prec}\,P\,x \wedge \mathsf{lst}_{\prec}\,P\,y} \\
\quad\equiv & \langle\text{Defin. lst, lb}\rangle & P\,x \wedge \forall\,(y\!:\!X\,.\,P\,y \Rightarrow x \prec y) \wedge P\,y \wedge \forall\,(x\!:\!X\,.\,P\,x \Rightarrow y \prec x) \\
\quad\Rightarrow & \langle\text{Instantiation}\rangle & P\,x \wedge (P\,y \Rightarrow x \prec y) \wedge P\,y \wedge (P\,x \Rightarrow y \prec x) \\
\quad\equiv & \langle\text{Rearranging}\rangle & P\,x \wedge (P\,x \Rightarrow y \prec x) \wedge P\,y \wedge (P\,y \Rightarrow x \prec y) \\
\quad\equiv & \langle\text{Modus Pon.}\rangle & P\,x \wedge y \prec x \wedge P\,y \wedge x \prec y \\
\quad\Rightarrow & \langle\text{Weakening}\rangle & y \prec x \wedge x \prec y \\
\quad\Rightarrow & \langle\text{Antisymm.}\rangle & x = y
\end{array}
$$

Remark: as we did for limits, we use the predicate transformers to define functionals $\bigsqcap$ (and $\bigsqcup$) mapping X-valued functions to the glb (and lub) of their range, for cases where existence and uniqueness are satisfied. The predicate used is then the *range predicate* $\mathsf{R}\_ : (Y \to X) \to X \to \mathbb{B}$ is defined by $\mathsf{R}_f\,x \equiv x \in \mathcal{R}\,f$.

Infix operators like $\sqcup$ are defined by variadic application: $x \sqcup y = \bigsqcup\,(x, y)$.

Observe how predicate calculus enhances similarity with the proofs about analysis

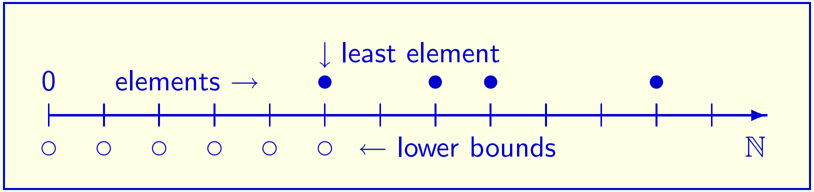
9.1.3 A case study: when is a greatest lower bound a least element?

a. Purpose: show how the formal rules help where semantic intuition lacks.

General orderings are rather abstract, examples are difficult to construct, and may also have hidden properties not covered by the assumptions.

Original motivation: in a minitheory for recursion.

- The greatest lower bound operator $\wedge$, defined as the particularization of $\sqcap$ to $\mathbb{R}' := \mathbb{R} \cup \{-\infty, +\infty\}$ with ordering $\leq$, is important in analysis.
- The recursion theory required least elements of nonempty subsets of $\mathbb{N}$. For these, it appears "intuitively obvious" that *both concepts coincide*.



Is this really obvious? The diagram gives no clue as to which axioms of $\mathbb{N}, \leq$ are involved, and so is useless for generalization. Formal study exposes the properties of natural numbers used and show generalizations.

b. The linear diagram suggests totality is the key.
This is misleading: $\mathbb{R}$ is totally ordered, yet

$$\boxed{P : \mathbb{R} \rightarrow \mathbb{B} \text{ with } \forall\, x : \mathcal{D}\, P \,.\, P\, x \;\equiv\; x \neq 0 \wedge x^{-} \in \mathbb{N}}$$

has g.l.b. 0 but no least element. To show this, we prove $\text{lb}_{\le}\, P\, x \Rightarrow x \le 0$ by contrapositivity, and also prove the converse. For any $x : \mathbb{R}$,

$$\begin{array}{lll}
0 < x & & \\
\quad\equiv & \langle \text{Type }{}^{-}\text{; by def.: } a \le \lceil a \rceil \rangle & 0 < x^{-} < (\lceil x^{-} \rceil + 1) \wedge (\lceil x^{-} \rceil + 1) \in \mathbb{N} \\
\quad\equiv & \langle 0 < a < b \;\equiv\; 0 < b^{-} < a^{-} \rangle & 0 < (\lceil x^{-} \rceil + 1)^{-} < x \wedge (\lceil x^{-} \rceil + 1) \in \mathbb{N} \\
\quad\Rightarrow & \langle \exists\text{-intr.: } \; y := (\lceil x^{-} \rceil + 1)^{-} \rangle & \exists\, y : \mathbb{R} \,.\, y \neq 0 \wedge y^{-} \in \mathbb{N} \wedge y < x \\
\quad\equiv & \langle \text{Dual, } \; \neg\, (y < x) \;\equiv\; x \le y \rangle & \neg\, (\forall\, y : \mathbb{R} \,.\, y \neq 0 \wedge y^{-} \in \mathbb{N} \Rightarrow x \le y \\
\quad\equiv & \langle \text{Definitions of } P \text{ and lb} \rangle & \neg\, (\text{lb}_{\le}\, P\, x) \\
x \le 0 & & \\
\quad\equiv & \langle \forall\, y : \mathbb{R} \,.\, P\, y \Rightarrow 0 < y \rangle & \forall\, y : \mathbb{R} \,.\, P\, y \Rightarrow 0 < y \wedge x \le 0 \\
\quad\Rightarrow & \langle \text{Trans. } <\text{, weaken conseq.} \rangle & \forall\, y : \mathbb{R} \,.\, P\, y \Rightarrow x \le y
\end{array}$$

Hence $\text{lb}_{\le}\, P\, x \;\equiv\; x \le 0$. Using this to prove $\neg\, (\text{lst}_{\le}\, P\, x)$ is straightforward.

c. Certain theorems needed for $\mathbb{N}$ with $\leq$ required no assumptions. Examples:

THEOREM, QUANTIFIED SHUNTING:
for any $P : X \to \mathbb{B}$, $Q : Y \to \mathbb{B}$, $\text{—}R\text{—} : X \times Y \to \mathbb{B}$,

$$\forall\,(x : X\,.\,P\,x \Rightarrow \forall\,(y : Y\,.\,Q\,y \Rightarrow x\,R\,y))$$
$$\equiv \forall\,(y : Y\,.\,Q\,y \Rightarrow \forall\,(x : X\,.\,P\,x \Rightarrow x\,R\,y))$$

PROOF: by left distributivity of $\Rightarrow/\forall$, shunting and dummy swap.

LEMMA: $\forall\,x : X\,.\,P\,x \Rightarrow \forall\,y : X\,.\,\mathsf{lb}_{\prec}\,P\,y \Rightarrow y \prec x$

HINT: weaken $\forall\,x : X\,.\,\mathsf{lb}_{\prec}\,P\,x \equiv \forall\,y : X\,.\,P\,y \Rightarrow x \prec y$ from $\equiv$ to $\Rightarrow$.

THEOREM, LEAST ELEMENTS AS G.L.B.S: $\mathsf{lst}_{\prec}\,P\,x \equiv P\,x \wedge \mathsf{glb}_{\prec}\,P\,x$.

PROOF:

$$
\begin{array}{ll}
 & \mathsf{lst}_{\prec}\,P\,x \\
\equiv & \langle \text{lst, lemma} \rangle \quad P\,x \wedge \mathsf{lb}_{\prec}\,P\,x \wedge (P\,x \Rightarrow \forall\,y : X\,.\,\mathsf{lb}_{\prec}\,P\,y \Rightarrow y \prec x) \\
\equiv & \langle \text{Mod. Pon.} \rangle \quad P\,x \wedge \mathsf{lb}_{\prec}\,P\,x \wedge \forall\,y : X\,.\,\mathsf{lb}_{\prec}\,P\,y \Rightarrow y \prec x \\
\equiv & \langle \text{Defin. glb} \rangle \quad P\,x \wedge \mathsf{glb}_{\prec}\,P\,x
\end{array}
$$

d. THEOREM, G.L.B.S AS LEAST ELEMENTS (FOR $\mathbb{N}$): $\mathsf{glb}_{\leq}\, P\, n \Rightarrow \mathsf{lst}_{\leq}\, P\, n$

PROOF: with abbreviations L for $\mathsf{lb}_{\leq}\, P$ and K for $\mathsf{lst}_{\leq}\, P$ and I for $\mathsf{glb}_{\leq}\, P$,

$$\begin{array}{lcll}
I\, n & \Rightarrow & \langle \text{Weakening} \rangle & \forall\, m : \mathbb{N} \,.\, L\, m \Rightarrow m \leq n \\
 & \equiv & \langle \text{Contraposit} \rangle & \forall\, m : \mathbb{N} \,.\, n < m \Rightarrow \neg L\, m \\
 & \equiv & \langle \text{Definition } L \rangle & \forall\, m : \mathbb{N} \,.\, n < m \Rightarrow \neg \forall\, k : \mathbb{N} \,.\, P\, k \Rightarrow m \leq k \\
 & \equiv & \langle \text{Duality } \forall/\exists \rangle & \forall\, m : \mathbb{N} \,.\, n < m \Rightarrow \exists\, k : \mathbb{N} \,.\, P\, k \wedge k < m \\
 & \Rightarrow & \langle \text{Weakening} \rangle & \forall\, m : \mathbb{N} \,.\, n < m \Rightarrow \exists\, P \\
 & \equiv & \langle \text{R-dstr. } \Rightarrow/\exists \rangle & \exists\, (m : \mathbb{N} \,.\, n < m) \Rightarrow \exists\, P \\
 & \equiv & \langle \text{Property } \alpha \rangle & \exists\, P \\
 & \Rightarrow & \langle \text{Property } \beta \rangle & \exists\, K \\
I\, n & \equiv & \langle I\, n \Rightarrow \exists\, K \rangle & \exists\, K \wedge I\, n \\
 & \Rightarrow & \langle \text{Property } \gamma \rangle & \exists\, K \wedge \forall\, m : \mathbb{N} \,.\, I\, m \Rightarrow m = n \\
 & \Rightarrow & \langle K\, n \Rightarrow I\, n \rangle & \exists\, K \wedge \forall\, m : \mathbb{N} \,.\, K\, m \Rightarrow m = n \\
 & \Rightarrow & \langle \text{Half-pt. rule} \rangle & K\, n
\end{array}$$

Property α is $\forall\, n : \mathbb{N} \,.\, \exists\, m : \mathbb{N} \,.\, n < m$ (weak).

Property β is well-foundedness: $\forall\, P : \mathbb{N} \rightarrow \mathbb{B} \,.\, \exists\, P \Rightarrow \exists\, (\mathsf{lst}_{\leq}\, P)$.

Property γ is (Antisymmetry $\leq$), hence $\forall\, n : \mathbb{N} \,.\, I\, n \Rightarrow \forall\, m : \mathbb{N} \,.\, I\, m \Rightarrow m = n$.

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

10 Well-foundedness, induction principles and applications

10.0 Induction over the natural numbers

10.1 Induction and well-founded relations

10.2 Application: inductive definitions over natural numbers

10.3 Application: inductive definitions over other structures

(in a later version)

10.4 Application: inductive reasoning about programs

10.0 Induction over the natural numbers

a. **Induction over natural numbers as transitivity and modus ponens**

Let $\prec$ be a transitive relation on A and $f : \mathbb{N} \rightarrow A$ satisfy

$$\boxed{\forall\, n : \mathbb{N} \,.\, f\, n \prec f\,(n+1)} \tag{60}$$

Theorems $f\,0 \prec f\,1$, $f\,1 \prec f\,2$ until $f\,n \prec f\,(n+1)$ are instantiations of (60). They imply $f\,0 \prec f\,(n+1)$ by transitivity and MP, yielding the expectation

$$\boxed{\forall\,(n : \mathbb{N} \,.\, f\, n \prec f\,(n+1)) \;\Rightarrow\; \forall\,(n : \mathbb{N} \,.\, f\,0 \prec f\,(n+1))} \tag{61}$$

Proof depends on axiomatization $\mathbb{N}$. If f is real-valued and $\prec$ is the usual $\leq$,

$$\boxed{\forall\,(n : \mathbb{N} \,.\, f\, n \leq f\,(n+1)) \;\Rightarrow\; \forall\,(n : \mathbb{N} \,.\, f\,0 \leq f\,(n+1))}$$

Specializing f to a $\mathbb{B}$-valued function (predicate) $P : \mathbb{N} \rightarrow \mathbb{B}$, and $\leq$ to $\Rightarrow$

$$\boxed{\forall\,(n : \mathbb{N} \,.\, P\, n \Rightarrow P\,(n+1)) \;\Rightarrow\; \forall\,(n : \mathbb{N} \,.\, P\,0 \Rightarrow P\,(n+1))}$$

or, by the rules of the predicate calculus (exercise),

$$\boxed{\forall\,(n : \mathbb{N} \,.\, P\, n \Rightarrow P\,(n+1)) \;\Rightarrow\; P\,0 \Rightarrow \forall\, P} \tag{62}$$

b. **Weak induction over natural numbers**

The preceding formula (62) yields the familiar weak induction principle

$$\boxed{P\,0 \wedge \forall\,(n\,{:}\,\mathbb{N}\,.\,P\,n \Rightarrow P\,(n+1)) \ \equiv\ \forall\,P} \tag{63}$$

Note that $\forall\,P \ \equiv\ \forall\,n\,{:}\,\mathbb{N}\,.\,P\,n$ since $P = n\,{:}\,\mathbb{N}\,.\,P\,n$.

Proof pattern: proving $\forall\,P$ is equivalent to proving

- **(BC)** Base case: prove $P\,0$
- **(IC)** Inductive case: prove $\forall\,(n\,{:}\,\mathbb{N}\,.\,P\,n \Rightarrow P\,(n+1))$.

In practice, the proof of **(IC)** is further (often tacitly) reduced in two stages.

(i) To prove **(IC)**, prove $n \in \mathbb{N} \Rightarrow P\,n \Rightarrow P\,(n+1)$, generalize implicitly.

(ii) To prove $n \in \mathbb{N} \Rightarrow P\,n \Rightarrow P\,(n+1)$ assume the antecedents $n \in \mathbb{N}$ and $P\,n$ (*induction hypothesis*, IH) and prove $P\,(n+1)$ (*induction step*, IS).

c. Illustration by a classical “textbook example”

$\boxed{\text{Theorem:}\quad \forall\, n : \mathbb{N} \,.\, \sum\,(i : \square\, n \,.\, 2 \cdot i + 1) = n^2}$

Proof: let $\boxed{P : \mathbb{N} \rightarrow \mathbb{B} \text{ with } P\,n \equiv \sum\,(i : \square\, n \,.\, 2 \cdot i + 1) = n^2}$ and prove $\boxed{\forall\, P}$.

- **(BC)** $\boxed{\text{Proving } P\,0}$, i.e., $\sum\,(i : \square\, 0 \,.\, 2 \cdot i + 1) = 0^2$

$$\boxed{\begin{array}{rcll} \sum\,(i : \square\, 0 \,.\, 2 \cdot i + 1) & = & \langle\text{Empty rule}\rangle & 0 \\ & = & \langle\text{Arithmetic}\rangle & 0^2 \end{array}}$$

- **(IC)** First method: by $\boxed{\text{proving } P\,n \Rightarrow P\,(n+1) \text{ for arbitrary } n : \mathbb{N}}$

$$\boxed{\begin{array}{lcl} P\,n & & \\ \equiv & \langle\text{Def. } P\rangle & \sum\,(i : \square\, n \,.\, 2 \cdot i + 1) \;=\; n^2 \\ \Rightarrow & \langle\text{Leibniz}\rangle & \sum\,(i : \square\, n \,.\, 2 \cdot i + 1) + 2 \cdot n + 1 \;=\; n^2 + 2 \cdot n + 1 \\ \equiv & \langle\text{Domain split}\rangle & \sum\,(i : \square\, (n+1) \,.\, 2 \cdot i + 1) \;=\; n^2 + 2 \cdot n + 1 \\ \equiv & \langle\text{Arithmetic}\rangle & \sum\,(i : \square\, (n+1) \,.\, 2 \cdot i + 1) \;=\; (n+1)^2 \\ \equiv & \langle\text{Def. } P\rangle & P\,(n+1) \end{array}}$$

Remarks:

- *Domain split* is the formula (derived from the axioms for $\sum$)

$$\boxed{\sum (i : 0 \mathinner{..} n \,.\, f\, i) = \sum (i : 0 \mathinner{..} n - 1 \,.\, f\, i) + f\, n}$$

for any numeric function f satisfying $0 \mathinner{..} n \subseteq \mathcal{D}\, f$.

- The second proof for $\boxed{P\, n \Rightarrow P\,(n+1)}$ uses the deduction theorem

(IH) $\boxed{\text{Assume } P\, n\text{, i.e., } \sum\, (i : \Box\, n \,.\, 2 \cdot i + 1) = n^2}$

(IS) $\boxed{\text{Proof of } P\,(n+1)\text{, i.e., } \sum\, (i : \Box\, (n+1) \,.\, 2 \cdot i + 1) = (n+1)^2}$

$$\boxed{\begin{aligned} &\textstyle\sum\, (i : \Box\, (n+1) \,.\, 2 \cdot i + 1) \\ &\quad = \quad \langle \text{Domain split} \rangle \quad \textstyle\sum\, (i : \Box\, n \,.\, 2 \cdot i + 1) + 2 \cdot n + 1 \\ &\quad = \quad \langle \text{IH} \rangle \quad n^2 + 2 \cdot n + 1 \\ &\quad = \quad \langle \text{Arithmetic} \rangle \quad (n+1)^2 \end{aligned}}$$

Note that the IH is $P\, n$, certainly *not* $P\, n \equiv \sum\, (i : \Box\, n \,.\, 2 \cdot i + 1) = n^2$, which trivially holds by virtue of the definition of P.

d. **Warning**

- When proving $P\,n \Rightarrow P\,(n+1)$ by assuming $P\,n$ and proving $P\,(n+1)$,

  $\boxed{P\,n \text{ is not a theorem, and hence may not be instantiated}}$

  This is the most frequent mistake made by students in proofs by induction.
- Instantiation may be useful in case several quantified variables are involved. Applying induction principle to one variable may suffice, but requires care.
- Method: to prove $\boxed{\forall\,(m,n) : \mathbb{N}^2 \,.\, Q\,(m,n)}$ for given $Q : \mathbb{N}^2 \to \mathbb{B}$, define

  $$\boxed{P : \mathbb{N} \to \mathbb{B} \quad \text{with} \quad P\,n \equiv \forall\,m : \mathbb{N} \,.\, Q\,(m,n)}$$

  and prove $\forall\,P$ by proving $P\,0$ and $P\,n \Rightarrow P\,(n+1)$ for arbitrary $n : \mathbb{N}$. The (IH) is $P\,n$ is $\forall\,m : \mathbb{N} \,.\, Q\,(m,n)$, allowing instantiation of m, not n.
- Safeguard: always explicitly introduce a predicate, with all quantifiers. Mention instantiatiions and generalizations explicitly.

e. **Illustration: proving a property of the Fibonacci numbers**

Given $\boxed{\mathbf{def}\ \mathsf{F}\_ : \mathbb{N} \to \mathbb{N}\ \mathbf{with}\ \mathsf{F}_0 = 0 \wedge \mathsf{F}_1 = 1 \wedge \mathsf{F}_{n+2} = \mathsf{F}_{n+1} + \mathsf{F}_n}$

To prove $\boxed{\mathsf{F}_{m+n+1} = \mathsf{F}_{m+1} \cdot \mathsf{F}_{n+1} + \mathsf{F}_m \cdot \mathsf{F}_n}$ for any m and n in $\mathbb{N}$,

we define $\boxed{P : \mathbb{N} \to \mathbb{B} \text{ with } P\,n \equiv \forall\, m : \mathbb{N} \,.\, \mathsf{F}_{m+n+1} = \mathsf{F}_{m+1} \cdot \mathsf{F}_{n+1} + \mathsf{F}_m \cdot \mathsf{F}_n}$

and prove $\forall\, P$ by induction.

- **(BC)** Proof of $P\,0$, i.e., $\forall\, m : \mathbb{N} \,.\, \mathsf{F}_{m+1} = \mathsf{F}_{m+1} \cdot \mathsf{F}_1 + \mathsf{F}_m \cdot \mathsf{F}_0$ (easy)
- **(IC)** $\boxed{\text{Assume } P\,n\text{, i.e., } \forall\, m : \mathbb{N} \,.\, \mathsf{F}_{m+n+1} = \mathsf{F}_{m+1} \cdot \mathsf{F}_{n+1} + \mathsf{F}_m \cdot \mathsf{F}_n \qquad \text{(IH)}}$

  $\boxed{\text{Prove } P\,(n+1)\text{, i.e., } \forall\, m : \mathbb{N} \,.\, \mathsf{F}_{m+(n+1)+1} = \mathsf{F}_{m+1} \cdot \mathsf{F}_{(n+1)+1} + \mathsf{F}_m \cdot \mathsf{F}_{n+1}}$

  For armitrary $m : \mathbb{N}$, calculate $\mathsf{F}_{m+(n+1)+1}$.

$$\boxed{\begin{aligned}
\mathsf{F}_{m+(n+1)+1} &= \langle \text{Assoc. } + \rangle & &\mathsf{F}_{(m+1)+n+1} \\
&= \langle \text{Instant. IH} \rangle & &\mathsf{F}_{m+2} \cdot \mathsf{F}_{n+1} + \mathsf{F}_{m+1} \cdot \mathsf{F}_n \\
&= \langle \text{Def. F} \rangle & &(\mathsf{F}_{m+1} + \mathsf{F}_m) \cdot \mathsf{F}_{n+1} + \mathsf{F}_{m+1} \cdot \mathsf{F}_n \\
&= \langle \text{Arithmetic} \rangle & &\mathsf{F}_{m+1} \cdot (\mathsf{F}_{n+1} + \mathsf{F}_n) + \mathsf{F}_m \cdot \mathsf{F}_{n+1} \\
&= \langle \text{Def. F} \rangle & &\mathsf{F}_{m+1} \cdot \mathsf{F}_{n+2} + \mathsf{F}_m \cdot \mathsf{F}_{n+1}
\end{aligned}}$$

f. **Strong induction over natural numbers**

$$\boxed{\forall\,(n\!:\!\mathbb{N}\,.\,\forall\,(i\!:\!\square\, n\,.\,P\,i) \Rightarrow P\,n) \quad\equiv\quad \forall\,P} \tag{64}$$

No separate base case since $P\,0$ is already implied by the l.h.s.:

$$\boxed{\begin{array}{lll}
\multicolumn{3}{l}{\forall\,(n\!:\!\mathbb{N}\,.\,\forall\,(i\!:\!\square\, n\,.\,P\,i) \Rightarrow P\,n)} \\
\quad\Rightarrow & \langle\text{Instantiation } n := 0\rangle & \forall\,(i\!:\!\square\, 0\,.\,P\,i) \Rightarrow P\,0 \\
\quad\equiv & \langle\text{Empty rule}\rangle & 1 \Rightarrow P\,0 \\
\quad\equiv & \langle\text{Left identity} \Rightarrow\rangle & P\,0,
\end{array}}$$

which allows rewriting the basic form (64) as follows (exercise).

$$\boxed{P\,0 \wedge \forall\,(n\!:\!\mathbb{N}\,.\,\forall\,(i\!:\!0\,..\,n\,.\,P\,i) \Rightarrow P\,(n+1)) \quad\equiv\quad \forall\,P} \tag{65}$$

Form (65) is useful when a base case is appropriate for some reason.

Usually it is more elegant to avoid a separate base case, and directly use (64).

10.1 Induction and well-founded relations

10.1.0 Conventions and definitions

a. **Shorthands** auxiliary operators (the first row: for for any set X)

$$\boxed{\begin{array}{cccr} \mathsf{pred}_X = X \to \mathbb{B} & \text{and} & \mathsf{rel}_X = X^2 \to \mathbb{B} & (66) \\ \mathbf{def}\ \mathsf{Pred} := \bigcup X : \mathcal{T} \,.\, \mathsf{pred}_X & \text{and} & \mathbf{def}\ \mathsf{Rel} := \bigcup X : \mathcal{T} \,.\, \mathsf{rel}_X & (67) \end{array}}$$

Observe that any $R : \mathsf{Rel}$ satisfies $\mathsf{tgt}\, R = \mathsf{src}\, R$.

b. **Formulations with sets versus predicates**

The formulations are fully equivalent. Switching between formulations:

$$\boxed{\begin{array}{lr} \mathbf{def}\ \mathsf{set}\_ : \mathcal{T} \ni X \to \mathsf{pred}_X \to \mathcal{P}\, X\ \mathbf{with}\ \mathsf{set}_X\, P = \{x : X \mid P\, x\} & (68) \\ \mathbf{def}\ \mathsf{prd}\_ : \mathcal{T} \ni X \to \mathcal{P}\, X \to \mathsf{pred}_X\ \mathbf{with}\ \mathsf{prd}_X\, S\, x \equiv x \in S & (69) \end{array}}$$

Given set X, $\mathsf{set}_X \in \mathsf{pred}_X \to \mathcal{P}\, X$, maps predicates on X to subsets of X, whereas $\mathsf{prd}_X \in \mathcal{P}\, X \to \mathsf{pred}_X$ is the inverse (proof: exercise).

Sets are more common, predicates are more calculationally convenient.

c. Relations and extremal elements

Formulation in set-oriented style.

$\boxed{\text{Definition, minimal and least elements of a relation}}$ (70)

Let $\_\_ \prec \_\_ : \mathsf{Rel}$ and $X := \mathsf{src}\,(\prec)$. For any subset S of X and any x in X,

- x is *minimal in* S iff x belongs to S and every y in X satisfying $y \prec x$ does not belong to S. Formally

$$\boxed{x \; ismin_{\prec} \; S \quad \equiv \quad x \in S \wedge \forall\, y : X \,.\, y \prec x \Rightarrow y \notin S}$$

- x is a *lower bound for* S iff all elements y of S satisfy $x \prec y$. Formally:

$$\boxed{x \; islb_{\prec} \; S \quad \equiv \quad \forall\, y : S \,.\, x \prec y}$$

- x is a *least element of* S iff x is in S and is a lower bound for S:

$$\boxed{x \; isleast_{\prec} \; S \quad \equiv \quad x \in S \wedge x \; islb_{\prec} \; S}$$

The type of these operators is (exercise) $\boxed{\mathsf{Rel} \ni R \to \mathsf{src}\, R \times \mathcal{P}\,(\mathsf{src}\, R) \to \mathbb{B}}$.

More elegant is the following formulation based on *predicate transformers*

DEFINITION, MINIMAL AND LEAST ELEMENTS OF A RELATION

Let $—\prec— : \mathsf{Rel}$ and $X := \mathsf{src}\,(\prec)$. For any predicate P on X and x in X,

- x is *minimal for* P iff x satisfies P and every $y : X$ satisfying $y \prec x$ does not satisfy P. Formally

$$\boxed{\mathsf{min}_{\prec}\, P\, x \quad \equiv \quad P\, x \wedge \forall\, y : X\, .\, y \prec x \Rightarrow \neg\, (P\, y)}$$

- x is a *lower bound for* P iff $x \prec y$ for all elements $y : X$ satisfying P:

$$\boxed{\mathsf{lb}_{\prec}\, P\, x \quad \equiv \quad \forall\, y : X\, .\, P\, y \Rightarrow x \prec y}$$

- x is a *least element for* P iff x satisfies P and is a lower bound for P:

$$\boxed{\mathsf{lst}_{\prec}\, P\, x \quad \equiv \quad P\, x \wedge \mathsf{lb}\, P\, x}$$

Here the type of all operators is $\boxed{\mathsf{Rel} \ni R \rightarrow \mathsf{pred}_{\mathsf{src}\, R} \rightarrow \mathsf{pred}_{\mathsf{src}\, R}}$.

10.1.1 Well-foundedness and supporting induction

a. $\boxed{\text{DEFINITION, WELL-FOUNDEDNESS}}$ (71)

A relation $\prec : X^2 \rightarrow \mathbb{B}$ is *well-founded* iff every nonempty subset of X has a minimal element.

$$\boxed{\mathsf{WF}\,(\prec) \quad \equiv \quad \forall\, S : \mathcal{P}\, X \,.\, S \neq \emptyset \Rightarrow \exists\, x : X \,.\, x \; ismin_{\prec}\, S}$$

b. $\boxed{\text{DEFINITION, SUPPORTING INDUCTION}}$ (72)

A relation $\prec\ : X^2 \rightarrow \mathbb{B}$ *supports induction* iff $\mathsf{SI}\ (\prec)$, wwith definition

$$\boxed{\mathsf{SI}\ (\prec) \quad \equiv \quad \forall\, P : \mathsf{pred}_X \,.\, \forall\, (x : X \,.\, \forall\, (y : X_{\prec\, x} \,.\, P\, y) \Rightarrow P\, x) \Rightarrow \forall\, x : X \,.\, P\, x}$$

c. $\boxed{\text{THEOREM, EQUIVALENCE OF WF AND SI}: \quad \mathsf{WF}\,(\prec) \;\equiv\; \mathsf{SI}\ (\prec)}$ (73)

PROOF: next slide

$$
\begin{array}{ll}
 & \text{WF}\,(\prec) \\
\equiv & \langle \text{Definition WF (71) and } S \neq \emptyset \;\equiv\; \exists\, x : S \,.\, 1 \rangle \\
 & \forall\, S : \mathcal{P}\, X \,.\, \exists\,(x : S \,.\, 1) \Rightarrow \exists\,(x : X \,.\, x \mathrel{\mathit{ismin}_{\prec}} S) \\
\equiv & \langle S = X \cap S,\ \text{trading} \rangle \\
 & \forall\, S : \mathcal{P}\, X \,.\, \exists\,(x : X \,.\, x \in S) \Rightarrow \exists\,(x : X \,.\, x \mathrel{\mathit{ismin}_{\prec}} S) \\
\equiv & \langle \text{Definition } \mathit{ismin} \text{ (70)} \rangle \\
 & \forall\, S : \mathcal{P}\, X \,.\, \exists\,(x : X \,.\, x \in S) \Rightarrow \exists\,(x : X \,.\, x \in S \wedge \forall\, y : X \,.\, y \prec x \Rightarrow y \notin S) \\
\equiv & \langle p \Rightarrow q \;\equiv\; \neg\, q \Rightarrow \neg\, p \rangle \\
 & \forall\, S : \mathcal{P}\, X \,.\, \neg\,(\exists\, x : X \,.\, x \in S \wedge \forall\, y : X \,.\, y \prec x \Rightarrow y \notin S) \Rightarrow \neg\,(\exists\, x : X \,.\, x \in S) \\
\equiv & \langle \text{Duality } \forall/\exists\text{, De Morgan} \rangle \\
 & \forall\, S : \mathcal{P}\, X \,.\, \forall\,(x : X \,.\, x \notin S \vee \neg\,(\forall\, y : X \,.\, y \prec x \Rightarrow y \notin S)) \Rightarrow \forall\, x : X \,.\, x \notin S \\
\equiv & \langle \vee \text{ to } \Rightarrow\text{, i.e., } a \vee \neg\, b \;\equiv\; b \Rightarrow a \rangle \\
 & \forall\, S : \mathcal{P}\, X \,.\, \forall\,(x : X \,.\, \forall\,(y : X \,.\, y \prec x \Rightarrow y \notin S) \Rightarrow x \notin S) \Rightarrow \forall\, x : X \,.\, x \notin S \\
\equiv & \langle \text{Change of variables: } S = \{x : X \mid \neg\,(P\, x)\} \text{ and } P\, x \;\equiv\; x \notin S \rangle \\
 & \forall\, P : X \to B \,.\, \forall\,(x : X \,.\, \forall\,(y : X \,.\, y \prec x \Rightarrow P\, y) \Rightarrow P\, x) \Rightarrow \forall\, x : X \,.\, P\, x \\
\equiv & \langle \text{Trading, def. SI (72)} \rangle \\
 & \text{SI}\,(\prec)
\end{array}
$$

10.1.2 Particular instances of well-founded induction

a. **Induction over** $\mathbb{N}$ Here we prove earlier principles axiomatically.

One of the axioms for natural numbers is:

$$\boxed{\text{Every nonempty subset of } \mathbb{N} \text{ has a } \textit{least} \text{ element under } \leq.}$$

Equivalently, every nonempty subset of $\mathbb{N}$ has a *minimal* element under $<$.

Strong induction over $\mathbb{N}$ follows by instantiating (73) with $<$ for $\prec$

$$\boxed{\forall\,(n\,{:}\,\mathbb{N}\,.\,P\,n) \;\equiv\; \forall\,(n\,{:}\,\mathbb{N}\,.\,\forall\,(m\,{:}\,\mathbb{N}\,.\,m < n \Rightarrow P\,m) \Rightarrow P\,n)}$$

Weak induction over $\mathbb{N}$ can be obtained in two ways.

- By proving that the relation $\prec$ defined by $\boxed{m \prec n \equiv m + 1 = n}$ is well-founded, and deducing from the general form in (72) that

$$\boxed{\forall\,(n\,{:}\,\mathbb{N}\,.\,P\,n) \;\equiv\; P\,0 \wedge \forall\,(n\,{:}\,\mathbb{N}\,.\,P\,n \Rightarrow P\,(n+1)).}$$

- By showing weak induction to be logically equivalent to strong induction.

b. **Structural induction**

- Over *sequences*: list prefix is well-founded and yields

THEOREM, STRUCTURAL INDUCTION FOR LISTS:
for any set A and any $P : A^* \to \mathbb{B}$,

$$\forall\,(x : A^* \,.\, P\,x) \;\equiv\; P\,\varepsilon \wedge \forall\,(x : A^* \,.\, P\,x \Rightarrow \forall\, a : A \,.\, P\,(a >\!\!- x)) \qquad (74)$$

Suffices for proving most properties about functional programs with lists.

- Other example: structural induction over expressions

THEOREM, STRUCTURAL INDUCTION OVER EXPRESSIONS (75)

Given the grammar

$$\begin{array}{rcl} \mathit{expression} & ::= & \mathit{constant} \mid \mathit{variable} \mid \mathit{application} \mid \mathit{abstraction}. \\ \mathit{application} & ::= & \underline{(}\mathit{expression}\ \mathit{expression}\underline{)} \mid \underline{(}\mathit{expression}\ \mathit{cop}''\ \mathit{expression}\underline{)}. \\ \mathit{abstraction} & ::= & \underline{(}\underline{\lambda}\mathit{variable}\underline{.}\mathit{expression}\underline{)}. \end{array}$$

generating the principal syntactic category E of expressions, and the auxiliary categories C, V defined separately in the usual way.

Theorem, structural induction over expressions, continuation

Then, for any predicate $P : E \longrightarrow \mathbb{B}$ on expressions, the following induction principle holds.

$$\boxed{\begin{aligned}
&\forall\,(e\!:\!E\,.\,P\,e) \quad \equiv \\
&\quad \forall\,(c\!:\!C\,.\,P\,c) \;\wedge\; \forall\,(v\!:\!V\,.\,P\,v) \wedge \\
&\quad \forall\,(e,e')\!:\!E^2\,.\,P\,e \wedge P\,e' \Rightarrow P\,[\![(e\,e')]\!] \wedge \\
&\qquad\qquad\qquad\qquad\qquad\qquad \forall\,(\star\!:\!C''\,.\,P\,[\![(e \star e')]\!]) \wedge \\
&\qquad\qquad\qquad\qquad\qquad\qquad \forall\,(v\!:\!V\,.\,P\,[\![(\lambda v.e)]\!])
\end{aligned}}$$

The proof of this theorem is rather detailed.

When using this theorem for an inductive proof, the obligations can be arranged in the familiar way.

(BC) *Base case:* $\boxed{\text{prove } P\,[\![c]\!] \text{ and } P\,[\![v]\!]}$;

(IC) *Inductive case:* $\boxed{\text{assuming } P\,e' \text{ and } P\,e}$ (*induction hypothesis*),
$\boxed{\text{prove } P\,[\![(e\,e')]\!] \text{ and } P\,[\![(e \star e')]\!] \text{ and } P\,[\![(\lambda v.e)]\!]}$ (*induction step*).

10.2 Application: inductive definitions over natural numbers

10.2.0 Principle

A general form of a recursive definition for functions over natural numbers: given $a : A$ and $g : \mathbb{N} \to A \to A$,

$$\boxed{\textbf{def}\ f_{\_} : \mathbb{N} \to A\ \textbf{with}\ f_0 = a \wedge \forall\, n : \mathbb{N} \,.\, f_{n+1} = g\, n\, f_n} \tag{76}$$

Equivalently (equivalence proof left as an exercise)

$$\boxed{\textbf{def}\ f_{\_} : \mathbb{N} \to A\ \textbf{with}\ f_n = (n = 0)\,?\,a \dagger g\,(n-1)\,f_{n-1}} \tag{77}$$

A first simple example: the sum of the n first natural numbers can be defined as

$$\boxed{\textbf{def}\ \mathit{snat} : \mathbb{N} \to \mathbb{N}\ \textbf{with}\ \mathit{snat}\,0 = 0 \wedge \forall\, n : \mathbb{N} \,.\, \mathit{snat}\ (n+1) = n + \mathit{snat}\,n.}$$

It is left as an easy exercise to prove by induction that

$$\boxed{\mathit{snat}\ n = \tfrac{n\cdot(n-1)}{2} \quad \text{and} \quad \mathit{snat}\ n = \textstyle\sum\,(i : \square\, n \,.\, i).}$$

10.2.1 Another example

The n-fold composition $f \uparrow n$ (or f^n) of a function $f : A \rightarrow A$ is defined by

$$\boxed{f^0 = id_A \;\; \text{and} \;\; \forall\, n : \mathbb{N} \,.\, f^{n+1} = f^n \circ f} \qquad (78)$$

$\boxed{\text{Example theorem: for any } f : A \rightarrow A, \quad \forall\, n : \mathbb{N} \,.\, \forall\, m : \mathbb{N} \,.\, f^{m+n} = f^m \circ f^n}$

Proof: $\boxed{\text{defining } P : \mathbb{N} \rightarrow \mathbb{B} \text{ with } P\, n = \forall\, m : \mathbb{N} \,.\, f^{m+n} = f^m \circ f^n}$ we prove $\boxed{\forall\, P}$

(BC) $\boxed{\text{Proof of } P\, 0}$, i.e., $\boxed{\forall\, (m : \mathbb{N} \,.\, f^{m+0} = f^m \circ f^0)}$ (easy)

(IC) $\boxed{\text{Proof of } P\, n \Rightarrow P\,(n+1)}$: we assume $P\, n$ (IH)

and prove $P\,(n+1)$, i.e., $\forall\, m : \mathbb{N} \,.\, f^{m+(n+1)} = f^m \circ f^{n+1}$. For any $m : \mathbb{N}$,

$$\boxed{\begin{aligned} f^{m+(n+1)} &= \langle \text{Assoc. } + \rangle && f^{(m+n)+1} \\ &= \langle \text{Defin. } f^n \rangle && f^{(m+n)} \circ f \\ &= \langle \text{Ind. Hyp.} \rangle && (f^m \circ f^n) \circ f \\ &= \langle \text{Assoc. } \circ \rangle && f^m \circ (f^n \circ f) \\ &= \langle \text{Defin. } f^n \rangle && f^m \circ f^{(n+1)} \end{aligned}}$$

10.2.2 Higher-order recursion on natural numbers

a. **Preamble:** useful auxiliary operator obviating ellipsis: *listto* $\boxed{\ldots}$
Not defined here. Property: for any integers m and n satisfying $m \leq n$,

$$\boxed{(m \ldots n) \in \square\,(n - m) \rightarrow \mathbb{Z} \quad \text{and} \quad \forall\, i : \mathcal{D}\,(m \ldots n)\,.\,(m \ldots n)\, i = m + i}$$

Examples: $3 \ldots 7 = 3, 4, 5, 6$. Instead of $x_m, x_{m+1}, \ldots, x_{n-1}$ we write $x \circ m \ldots n$, and instead of $x_m + x_{m+1} + \cdots + x_{n-1}$ we write $\sum\,(x \circ m \ldots n)$.

b. **Higher order recursion**

Typical case: $f\,n$ depends not on $f\,(n-1)$ only but also on preceding values.

The general form of a *k-th order recursive definition* is

$$\boxed{\mathbf{def}\ f\_\_ : \mathbb{N} \rightarrow A\ \mathbf{with}\ f_{<k} = a \wedge \forall\, n : \mathbb{N}\,.\, f_{n+k} = g\, n\, (f \circ (n \ldots n + k)) \qquad (79)}$$

where $a\_\_ : A^k$ and $g : \mathbb{N} \rightarrow A^k$ are assumend given. Equivalently (exercise!),

$$\boxed{\mathbf{def}\ f\_\_ : \mathbb{N} \rightarrow A\ \mathbf{with}\ f_n = (n < k)\,?\, a_n \dagger g\,(n - k)\,(f \circ (n - k \ldots n))}$$

where $a\_\_ : A^k$ and $g : \mathbb{N} \rightarrow A^k$ are assumend given.

c. **Example** of 2nd order recursion: defining the Fibonacci function.

$$\boxed{\textbf{def}\ \mathsf{F}\_ : \mathbb{N} \to \mathbb{N}\ \textbf{with}\ \mathsf{F}_n = (n < 2)\,?\, n \nmid \mathsf{F}_{n-1} + \mathsf{F}_{n-2}}$$

d. **Transformation to a first order recursive definition on k-tuples** Let

$$\boxed{\textbf{def}\ F\_ : \mathbb{N} \to A^k\ \textbf{with}\ F_n\, i = f_{n+i},}$$

then F satisfies the recursion equation

$$\boxed{\forall\, n : \mathbb{N} \,.\, F_n = (n = 0)\,?\, a \nmid G\,(n-1)\,F_{n-1}}$$

where G is related to g by

$$\boxed{\textbf{def}\ G : \mathbb{N} \to A^k \to A^k\ \textbf{with}\ G\,n\,x\,i = (i < k - 1)\,?\, x\,(i+1) \nmid g\,n\,x}$$

The proof is left as an excercise in formal derivation. Clearly $\boxed{f_n = F_n\,0}$.

e. **Application to the Fibonacci numbers** yields $\boxed{F_n = (\mathsf{F}_n, \mathsf{F}_{n+1})}$ and hence

$$\boxed{\begin{aligned} F_n &= (n = 0)\,?\,(0,1) \nmid G\,(n-1)\,F_{n-1} \\ G\,n\,x\,i &= (i < 1)\,?\,x\,(i+1) \nmid \textstyle\sum x \end{aligned}}$$

The definition for G can be rewritten $\boxed{G\,n\,x\,i = (i = 0)\,?\,x\,1 \nmid x\,0 + x\,1}$ or

$$\boxed{G\,n\,x = x \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}}$$

Since $G\,n\,x$ does not depend on n, clearly

$$\boxed{F_n = F_0 \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n}$$

which is an explicit solution without irrational numbers. Observe that

$$\boxed{\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^3 = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^4 = \begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix}}$$

10.3 Application: inductive definitions over other structures

THIS SECTION TO BE COMPLETED IN A LATER VERSION

10.4 Application: inductive reasoning about programs

For programs expressed in a functional language (e.g., Haskell), the framework developed thus far suffices, and examples of inductive reasoning will be found in the exercises.

Programs expressed in imperative languages require develop some additional theory in our framework (denotational semantics, axiomatic semantics).

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

11 Funmath: unifying continuous and discrete mathematics

11.0 Motivation: functional mathematics as the unifying principle

11.1 Syntax: the four basic constructs

11.2 Pragmatics: elastic and variadic operators

11.3 Formal calculation: calculational logic and generic functionals

11.0 Motivation: functional mathematics as the unifying principle

- **Principle:** (re)defining mathematical objects, whenever feasible, as functions.
  Remark: found especially advantageous where this is not yet a commonplace
- **Epistemological advantages**
  - Conceptual: uniformity in treatment while respecting essential differences
  - Practical: sharing general-purpose (generic) operators over functions
- **Example: sequences** (generic for tuples, lists, etc., "homogeneous" or not)
  - Why this example? "interface" between discrete and continuous math
  - Wide ramifications: (also in the design of mathematical software)
    - ∗ Removal of all conventions having poor calculational properties
      Worst kind of violation: against Leibniz's principle. Example: ellipsis

      $$\boxed{a_0 + a_1 + \cdots + a_7 \text{ where } a_i = i^2 \text{ yields } 0 + 1 + \cdots + 49}$$

    - ∗ Replacement by well-defined operators and algebraic calculation rules

- **Notational ramifications of “functional mathematics”**
  - Entire notational framework needs/uses only four constructs:

identifier	function application	abstraction	tuple denotation
n, $succ$, $+$, $\forall$	$f\,n$, $x+y$, $\forall\, x : X \,.\, p$	$x : X \wedge p \,.\, e$	a, b, c

  - Properties:
    - ∗ Synthesizes common conventions, minus ambiguities/inconsistencies
    - ∗ Adds forms of expression (e.g., point-free) and precise calculation rules
  - Case example: the **function range** operator $\mathcal{R}$
    - ∗ Common practice: overloaded use of $\in$ for binding, e.g., $\forall\, x \in X \,.\, p$
      Problem: $\{m \in \mathbb{N} \mid m < n\}$ and $\{2 \cdot m \mid m \in \mathbb{N}\}$ may seem harmless,
      but: what about $\{x \in X \mid x \in Y\}$? Is it $X \cap Y$ or a subset of $\mathbb{B}$?
    - ∗ Solution: *range* operator with axiom $y \in \mathcal{R}\, f \ \equiv\ \exists\, x : \mathcal{D}\, f \,.\, y = f\, x$.
    - ∗ Equivalent symbol: $\{\text{—}\}$. Application to tuples and abstractions yields familiar form and meaning. Examples: $\{a, b, c\}$ and $\{n : \mathbb{Z} \,.\, 2 \cdot n\}$
    - ∗ Using $x : X \mid p$ as syntactic sugar for $x : X \wedge p \,.\, x$ does the same for expressions like $\{m : \mathbb{N} \mid m < n\}$.

- **Benefits to mathematical practice**
  - Uniformly unambiguous and consistent conventions
  - Extreme simplicity: only 4 constructs, no ad hoc notations
  - Formal calculation rules, convenient for everyday practice
- **Benefits to programming** (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

11.1 Syntax: the four basic constructs

We have already seen all of them. Here are some additional remarks

a. **Identifier:** any (string of) symbol(s) *except* markers, parentheses, keywords.

- Markers are: binding colon (.), filter mark ($\wedge$), abstraction dot (.)
  Keywords are: **def**, **spec**, **where**.
- Identifiers are *introduced* by *bindings* of the form $\boxed{i : X \wedge p}$
  Here i may be a list of identifiers. The *filter* $\wedge\ p$ (or **with** p) is optional
  Example: $n : \mathbb{N}$ and $n : \mathbb{Z} \wedge n \geq 0$ are interchangeable.
- *Constants* are identifiers introduced by *definitions*, of the form $\boxed{\mathbf{def}\ binding}$
  Example: $\boxed{\mathbf{def}\ roto : \mathbb{R}_{\geq 0}\ \mathbf{with}\ roto^2 = 2}$.
  The scope is *global*. Existence and uniqueness are proof obligations.
- These obligations do not apply to *specifications*, of the form $\boxed{\mathbf{spec}\ binding}$.
- Well-established symbols (e.g., $\mathbb{B}$, $\Rightarrow$, $\mathbb{R}$, $+$, $\sqrt{\ }$) are predefined constants.

b. **Abstractions** are of the form $\boxed{\textit{binding} \,.\, \textit{expression}}$

The identifiers introduced are *variables*, with scope limited to the abstraction.

An abstraction denotes a *function*: writing f for $v : X \wedge p \,.\, e$,

- the domain axiom is $\boxed{d \in \mathcal{D}\, f \;\equiv\; d \in X \wedge p[^v_d}$ and
- the mapping axiom is $\boxed{d \in \mathcal{D}\, f \Rightarrow f\, d = e[^v_d}$.

Here $e[^v_d$ is e with d substituted for v. Example: $n : \mathbb{Z} \,.\, 2 \cdot n$.

c. **Tupling** is of the form $\boxed{e, e', e''}$ (any length, say, n)

Tupling enotes a function with

- domain $\boxed{0..n-1}$, e,g., $\boxed{\mathcal{D}\,(e, e', e'') = 0..2}$
- mapping as in $\boxed{(e, e', e'')\, 0 = e \text{ and } (e, e', e'')\, 1 = e' \text{ and } (e, e', e'')\, 2 = e''}$

d. **Function application** has the form $\boxed{f\,e}$ in the default *prefix* syntax.

A function identifier is called *operator*

For an operator, other conventions can be specified by dashes in its binding, e.g., $\text{—}\star\text{—}$ for infix.

Prefix has precedence over infix.

Parentheses are used for overriding precedence rules, *never* as an operator.

Special forms:

- *Partial application:* if $\star$ is an infix operator, then $(a\star)$ and $(\star b)$ satisfy $(a\star)\,b = a \star b = (\star b)\,a$.
- *Variadic application*, of the form $\boxed{e \star e' \star e'' \star e'''}$, is explained soon.

Macros can define shorthands or sugaring in terms of the basic syntax

Very few are needed.

- Shorthands are $\boxed{d^e \text{ for } d \uparrow e}$ (exponent) and $\boxed{d_e \text{ for } d \downarrow e}$ (filtering).
- Sugaring macros are

  $\boxed{e \mid v : X \wedge p \text{ for } v : X \wedge p \,.\, e}$ and $\boxed{v : X \mid p \text{ for } v : X \wedge p \,.\, v}$

  $\boxed{v := e \text{ for } v : \iota\, e}$, using the *singleton set injector* ι; axiom: $d \in \iota\, e \;\equiv\; d = e$.

  $\boxed{X \ni x \rightarrow Y \text{ for } \times\, x : X \,.\, Y}$, explained later

11.2 Pragmatics: elastic and variadic operators

a. Elastic operators

- Principle: functionals replacing the common ad hoc abstractors, e.g.,

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

Together with function abstraction, they yield familiar expressions

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

For less casual "users", they also yield point-free forms such as

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

and direct applicability to sequences (tuples, lists etc.)

$$\forall\, (x, y) = x \wedge y \qquad \textstyle\sum\, (x, y) = x + y$$

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 style.

b. **Some observations**

- Examples of definitions:
  - Quantifiers: $\boxed{\forall P \;\equiv\; P = \mathcal{D}\,P^{\bullet}\,1}$
  - Summation: $\boxed{\sum \varepsilon = 0 \quad \sum (a \mapsto c) = c \quad \sum (f \mathbin{\dot\cup} g) = \sum f + \sum g \quad (80)}$
    for any a, any numeric c and any number-valued functions f and g with finite nonintersecting (but otherwise arbitrary) domains.

  Note: (80) shows $\sum$ to be a *merge homomorphism*, a generalization of the well-known $h\,(x \mathbin{+\!\!+} y) = h\,x \oplus h\,y$ for data structures x and y.

- Subsumes the $\boxed{\text{“Eindhoven notation”}\;\; Q\,x : P.x : f.x}$.
  Advantages of lastic operators:
  - More general underlying different principle and design
  - Support also the point-free style
  - No algebraic restrictions (associativity, commutativity etc.)
    R. Backhouse also noted that certain restrictions are unnecessary; elastic operators go further
  - Not restricted to discrete math (useful in mathematical analysis).

c. **Variadic operators, elastic extensions and variadic shorthand**

- Variadic operator: elastic operator with meaningful restriction to sequences
- Elastic extension of an infix operator $\star$:
  - Definition: any elastic operator F satisfying

$$\boxed{F\,(x,y) = x \star y}$$

  - Examples:

$$\boxed{\forall\,(x,y) \;\equiv\; x \wedge y \quad \text{and} \quad \sum\,(x,y) = x + y}$$

  - Important: not unique, leaving room for judicious design

- Variadic shorthand:
  - Definition: argument/operator alternation

$$\boxed{x \star y \star z \quad \text{denoting} \quad F\,(x, y, z)}$$

  - Examples: $\boxed{x \wedge y \wedge z \equiv \forall\,(x, y, z) \text{ and } x + y + z = \sum\,(x, y, z)}$

    More interesting: with predicates con (*constant*) and inj (*injective*),

$$\boxed{\begin{array}{lcl} x = y = z & \equiv & \mathsf{con}\,(x, y, z) \\ x \neq y \neq z & \equiv & \mathsf{inj}\,(x, y, z). \end{array}}$$

    Gives $x \neq y \neq z$ useful meaning (impossible with "old" conventions).

  - Choice of F for commonly used $\star$: conservative.

    Example: for associative $\star$, require that the restriction of F to lists is a *list homomorphism*, i.e., for any lists x and y in $\mathcal{D}\,F$,

$$\boxed{F\,(x +\!\!+\, y) = F\,x \star F\,y}$$

11.3 Formal calculation: calculational logic and generic functionals

a. **Calculational logic:** in the chapters on proposition and predicate calculus

b. **Generic functionals:** in the next chapter

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

12 Generic Functionals: definitions

12.0 Design criteria and method for generic functionals

12.1 Functionals designed generically

12.2 Elastic extensions for generic operators

12.3 A generic functional refining function types

12.0 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:** restrictions on the arguments, e.g.,
  - the usual $f \circ g$ requires $\mathcal{R}\,g \subseteq \mathcal{D}\,f$, in which case $\mathcal{D}\,(f \circ g) = \mathcal{D}\,g$
  - the usual f^- requires f injective, in which case $\mathcal{D}\,f^- = \mathcal{R}\,f$
- **Approach used here:**
  - No restrictions on the argument function(s)
  - Instead, refine domain of the result function (say, f) via its domain axiom $x \in \mathcal{D}\,f \;\equiv\; x \in X \wedge p$ ensuring that, in the mapping axiom $x \in \mathcal{D}\,f \Rightarrow q$, q does not contain out-of-domain applications in case $x \in \mathcal{D}\,f$ (guarded)
  - Conservational, i.e.,

    For previously known functionals, if the traditional restriction on the argument is satisfied anyway, the generalization yields the "usual" case.

    For new functionals (extension, transposition, merge etc.): design freedom

12.1 Functionals designed generically

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

$$\boxed{f \downarrow P \;=\; x : \mathcal{D}\, f \cap \mathcal{D}\, P \wedge P\, x \,.\, f\, x} \qquad (81)$$

  Set filtering: $\boxed{x \in X \downarrow P \;\equiv\; x \in X \cap P \wedge P\, x}$.

  Shorthand: a_b for $a \downarrow b$, yielding convenient abbreviations like $f_{<n}$ and $\mathbb{R}_{\geq 0}$.

- **Function restriction ($\rceil$):** the usual domain restriction:

$$\boxed{f \rceil X \;=\; f \downarrow (X^{\bullet} 1).} \qquad (82)$$

- **Composition ($\circ$)** generalizes traditional composition: for *any* functions f and g (without restriction),

$$\boxed{\begin{array}{lcl} x \in \mathcal{D}\,(f \circ g) & \equiv & x \in \mathcal{D}\, g \wedge g\, x \in \mathcal{D}\, f \\ x \in \mathcal{D}\,(f \circ g) & \Rightarrow & (f \circ g)\, x = f\,(g\, x). \end{array}}$$

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

- **Inversion ($\boldsymbol{—^-}$)** For any function f,

$$\boxed{\mathcal{D}\, f^- = \mathsf{Bran}\, f \quad \text{and} \quad x \in \mathsf{Bdom}\, f \Rightarrow f^-\,(f\,x) = x.} \tag{83}$$

For Bdom (*bijectivity domain*) and Bran (*bijectivity range*):

$$\boxed{\begin{array}{rcl} \mathsf{Bdom}\, f & = & \{x : \mathcal{D}\, f \mid \forall\, x' : \mathcal{D}\, f \,.\, f\, x' = f\, x \Rightarrow x' = x\} \\ \mathsf{Bran}\, f & = & \{f\, x \mid x : \mathsf{Bdom}\, f\}. \end{array}} \tag{84}$$

Note that, if the traditional injectivity condition is satisfied, $\mathcal{D}\, f^- = \mathcal{R}\, f$.

- **Dispatching ($\&$) and parallel ($\parallel$)** For any functions f and g,

$$\boxed{\begin{array}{ll} \mathcal{D}\,(f \,\&\, g) = \mathcal{D}\, f \cap \mathcal{D}\, g & \quad x \in \mathcal{D}\,(f \,\&\, g) \Rightarrow (f \,\&\, g)\, x = f\, x, g\, x \\ \mathcal{D}\,(f \parallel g) = \mathcal{D}\, f \times \mathcal{D}\, g & \quad x \in \mathcal{D}\,(f \parallel g) \Rightarrow (f \parallel g)\,(x, y) = f\, x, g\, y \end{array}} \tag{85}$$

- **(Duplex) direct extension ($\widehat{-}$)** For any infix operator $\star$, functions f, g,

$$\boxed{\begin{array}{lcl} x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) & \equiv & x \in \mathcal{D}\,f \cap \mathcal{D}\,g \wedge (f\,x, g\,x) \in \mathcal{D}\,(\star) \\ x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) & \Rightarrow & (f \mathbin{\widehat{\star}} g)\,x = f\,x \star g\,x. \end{array}} \tag{86}$$

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

If $x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow (f\,x, g\,x) \in \mathcal{D}\,(\star)$ then $f \mathbin{\widehat{\star}} g = x : \mathcal{D}\,f \cap \mathcal{D}\,g \,.\, f\,x \star g\,x$.

Example: *equality*: $(f \mathbin{\widehat{=}} g) = x : \mathcal{D}\,f \cap \mathcal{D}\,g \,.\, f\,x = g\,x$

Half direct extension: for any function f and any x,

$$\boxed{f \mathbin{\overleftarrow{\star}} x = f \mathbin{\widehat{\star}} \mathcal{D}\,f^{\bullet}\,x \quad \text{and} \quad x \mathbin{\overrightarrow{\star}} f = \mathcal{D}\,f^{\bullet}\,x \mathbin{\widehat{\star}} f.}$$

Simplex direct extension ($\overline{-}$): recall $\boxed{\overline{f}\,g = f \circ g}$.

- **Function override ($\olessthan$ and $\ogreaterthan$)** For funcs. f and g, $\boxed{g \olessthan f = f \ogreaterthan g}$ and

$$\boxed{\begin{array}{rcl} \mathcal{D}\,(f \ogreaterthan g) & = & \mathcal{D}\,f \cup \mathcal{D}\,g \\ x \in \mathcal{D}\,(f \ogreaterthan g) & \Rightarrow & (f \ogreaterthan g)\,x = x \in \mathcal{D}\,f \mathbin{?} f\,x \nmid g\,x \end{array}}$$

- **Function merge ($\cupdot$)** For any functions f and g,

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

- **Relational functionals: compatibility ($\copyright$), subfunction ($\subseteq$)**

$$\boxed{\begin{array}{rcl} f \copyright g & \equiv & f \rceil \mathcal{D}\,g = g \rceil \mathcal{D}\,f \\ f \subseteq g & \equiv & f = g \rceil \mathcal{D}\,f. \end{array}}$$

Examples of typical algebraic properties

- $\boxed{f \subseteq g \equiv \mathcal{D}\,f \subseteq \mathcal{D}\,g \wedge f \copyright g}$ and $\boxed{f \copyright g \Rightarrow f \ogreaterthan g = f \cupdot g = f \olessthan g}$
- $\subseteq$ is a partial order (reflexive, antisymmetric, transitive)
- For equality: $\boxed{f \copyright g \equiv \forall\,(f \mathrel{\widehat{=}} g)}$ and $\boxed{f = g \equiv \mathcal{D}\,f = \mathcal{D}\,g \wedge f \copyright g}$.

12.2 Elastic extensions for generic operators

- **Function transposition ($—^T$)** The image definition is $\boxed{f^T y\, x = f\, x\, y}$. Making $—^T$ generic requires decision about $\mathcal{D}\, f^T$ for *any* function family f.
  - Intersecting variant ($—^T$) Motivation: $\overset{<}{g}\ f\ =\ g \circ f^T$ uses intersection. This suggests taking $\mathcal{D}\, f^T\ =\ \bigcap x : \mathcal{D}\, f \,.\, \mathcal{D}\,(f\, x)$ or $\mathcal{D}\, f^T\ =\ \bigcap(\mathcal{D} \circ f)$ Hence we define

$$\boxed{f^T = y : \bigcap (\mathcal{D} \circ f) \,.\, x : \mathcal{D}\, f \,.\, f\, x\, y} \tag{87}$$

or, with separate axioms,

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

– Uniting variant $(\!-\!\!-^{U})$, taking $\mathcal{D}\, f^{U} = \bigcup (\mathcal{D} \circ f)$.
Here the design criterion, leads to defining

$$\boxed{f^{U} = y : \bigcup (\mathcal{D} \circ f) \,.\, x : \mathcal{D}\, f \wedge\!\!\!\!\!\cdot\;\, y \in \mathcal{D}\,(f\, x) \,.\, f\, x\, y} \qquad (88)$$

or, with separate axioms,

$$\boxed{\begin{array}{rcl} \mathcal{D}\, f^{U} & = & \bigcup x : \mathcal{D}\, f \,.\, \mathcal{D}\,(f\, x) \\ y \in \mathcal{D}\, f^{U} & \Rightarrow & \mathcal{D}\,(f^{U} y) = \{x : \mathcal{D}\, f \mid y \in \mathcal{D}\,(f\, x)\} \\ & \wedge & (x \in \mathcal{D}\,(f^{U} y) \Rightarrow f^{U} y\, x = f\, x\, y) \end{array}}$$

– Variadic shorthand Observation: $(g \,\&\, h)\, x\, i = (g, h)\, i\, x$ for $i : \{0, 1\}$.
Design decision:

$$\boxed{f \,\&\, g \& \, h = (f, g, h)^{T}}$$

- **Elastic parallel ($\|$)** For any function family F and function f,

$$\boxed{\| \, F\, f \;\; = \;\; x : \mathcal{D}\, F \cap \mathcal{D}\, f \mathrel{\dot{\wedge}} f\, x \in \mathcal{D}\,(F\, x) \,.\, F\, x\,(f\, x)} \tag{89}$$

This can be seen as a typed variant of the well-known **S**-combinator.

- **Elastic merge** For any function family f,

$$\boxed{\begin{array}{c} y \in \mathcal{D}\,(\dot{\bigcup}\, f) \quad \equiv \\ y \in \bigcup\,(\mathcal{D} \circ f) \wedge \forall\,(x, x') : (\mathcal{D}\, f)^2 \,.\, y \in \mathcal{D}\,(f\, x) \cap \mathcal{D}\,(f\, x') \Rightarrow f\, x\, y = f\, x'\, y \\ y \in \mathcal{D}\,(\dot{\bigcup}\, f) \quad \Rightarrow \quad \forall\, x : \mathcal{D}\, f \,.\, y \in \mathcal{D}\,(f\, x) \Rightarrow \dot{\bigcup}\, f\, y = f\, x\, y \end{array}} \tag{90}$$

Interesting examples: for any function g,

$$\boxed{g = \dot{\bigcup}\, x : \mathcal{D}\, g \,.\, x \mapsto g\, x \quad \text{and} \quad g^{-} = \dot{\bigcup}\, x : \mathcal{D}\, g \,.\, g\, x \mapsto x}$$

Illustrates how the generic design criterion leads to fine operator intermeshing.

- **Elastic compatibility** For any function family f

$$\boxed{\textcircled{c}\, f \;\; \equiv \;\; \forall\,(x, y) : (\mathcal{D}\, f)^2 \,.\, f\, x \mathrel{\textcircled{c}} f\, y} \tag{91}$$

In general, $\dot{\cup}$ is not associative, but $\textcircled{c}\,(f, g, h) \Rightarrow (f \mathbin{\dot{\cup}} g) \mathbin{\dot{\cup}} h = f \mathbin{\dot{\cup}} (g \mathbin{\dot{\cup}} h)$.

12.3 A generic functional refining function types

- **Coarse typing: the function arrow** defined by

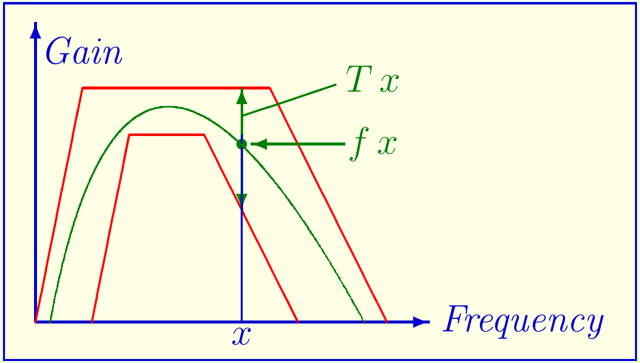
$$\boxed{f \in X \to Y \quad \equiv \quad \mathcal{D}\, f = X \wedge \mathcal{R}\, f \subseteq Y}$$

- **The function approximation paradigm for range refinement**
  - Purpose: formalizing *tolerances* for *functions*
  - Principle: *tolerance function* T specifies for every domain value x the set $T\, x$ of allowable values. Note: $\mathcal{D}\, T$ serves as the domain specification.

    Formalized: a function f *meets* tolerance T iff

$$\boxed{\mathcal{D}\, f = \mathcal{D}\, T \ \wedge \ (x \in \mathcal{D}\, f \cap \mathcal{D}\, T \Rightarrow f\, x \in T\, x).}$$

– Pictorial representation (example: radio frequency filter characteristic).



$$\boxed{\mathcal{D}\,f = \mathcal{D}\,T \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,T \Rightarrow f\,x \in T\,x)}$$

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

$$\boxed{f \in \times T \;\equiv\; \mathcal{D}\, f = \mathcal{D}\, T \wedge \forall\, x : \mathcal{D}\, f \cap \mathcal{D}\, T \,.\, f\, x \in T\, x} \qquad (92)$$

Immediate properties of (**??**):

a. Function *equality* $f = g \equiv \mathcal{D}\, f = \mathcal{D}\, g \wedge \forall\, x : \mathcal{D}\, f \cap \mathcal{D}\, g \,.\, f\, x = g\, x$ yields the "exact approximation"

$$\boxed{f = g \equiv f \in \times (\iota \circ g)}$$

b. (Semi-)pointfree form:

$$\boxed{\times T = \{f : \mathcal{D}\, T \to \bigcup T \mid \forall\, (f \mathbin{\widehat{\in}} T\}}$$

– Commonly used conventions as particularizations

- The usual Cartesian product for a pair of sets A, B is defined by

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

Letting $T := A, B$ in (**??**), calculation yields

$$\boxed{\bigtimes(A, B) = A \times B}$$

If $A \neq \emptyset$ and $B \neq \emptyset$, then $\bigtimes^{-}(A \times B)\,0 = A$ and $\bigtimes^{-}(A \times B)\,1 = B$.

∗ Dependent types: letting $T := a : A\,.\,B_a$,

$$\boxed{\times\,(a : A\,.\,B_a) = \{f : A \to \bigcup\,(a : A\,.\,B_a) \mid \forall\, a : A\,.\,f\,a \in B_a\}}$$

Convenient shorthand: $A \ni a \to B_a$ for $\times\, a : A\,.\,B_a$ Examples:

- Clearer chained dependencies, e.g., $\boxed{A \ni a \to B_a \ni b \to C_{a,b}}$.
- Types for generic operators. Let $\mathcal{F}$ and $\mathcal{T}$ be the function and type universes and fam defined by $f \in \text{fam}\ Y \equiv \mathcal{R}\,f \subseteq Y$,

$$\boxed{\begin{array}{lcl} (\,\circ\,) & \in & \mathcal{F}^2 \ni (f, g) \to \{x : \mathcal{D}\,g \mid g\,x \in \mathcal{D}\,f\} \to \mathcal{R}\,f \\ (\,^{T}) & \in & \text{fam}\ \mathcal{F} \ni f \to \bigcap\,(\mathcal{D} \circ f) \to \mathcal{D}\,f \ni x \to \mathcal{R}\,(f\,x). \end{array}}$$

Next section

0. Introduction: motivation and approach
1. Simple expressions and equational calculation
2. Equational calculation: application examples
3. Point-wise and point-free styles of expression and calculation
4. Point-wise and point-free styles: application examples
5. Binary algebra and proposition calculus: calculation rules
6. Binary algebra and proposition calculus: application examples
7. Sets and functions: equality and calculation
8. Functional Predicate Calculus: calculating with quantifiers
9. Functional Predicate Calculus: application examples
10. Well-foundedness, induction principles and application examples
11. Funmath: unifying continuous and discrete mathematics
12. Generic Functionals: definitions
13. Various examples in continuous and discrete systems modelling

13 Various examples in continuous and discrete systems modelling

13.0 Introduction: role of generic functionals

13.1 Origin: deriving data flow circuits from specifications

13.2 Applications in functional programming

13.3 Aggregate data types and structures

13.4 Overloading and polymorphism

13.5 Formal semantics (from conventional languages to LabVIEW)

13.6 Relational databases in functional style

13.7 Final considerations

13.8 Conclusion

13.0 Introduction: role of generic functionals

13.0.0 Sequences as functions: inheriting the generic functionals

- Intuitively evident: $\boxed{(a, b, c)\,0 = a \text{ and } (a, b, c)\,1 = b}$ etc., yet:
  - traditionally handled as entirely or subtly distinct from functions,
  - in the few exceptions, functional properties left unexploited.
- Examples of (underexploited) functional properties of sequences
  - Function inverse: $\boxed{(a, b, c, d)^{-}\,c = 2}$ (provided $c \notin \{a, b, d\}$)
  - Composition: $\boxed{(0, 3, 5, 7) \circ (2, 3, 1) = 5, 7, 3 \text{ and } f \circ (x, y) = f\,x, f\,y}$
  - Transposition: $\boxed{(f, g)^T x = f\,x, g\,x}$
- Not obtainable by the usual formal treatments of lists, e.g.,
  - recursive definition: $[]$ is a list and, if x is a list, so is $cons\,a\,x$
  - index function separate: $ind\,(cons\,a\,x)\,(n + 1) = ind\,x\,n$ e.g.,
    in Haskell: `ind [a:x] 0 = a` and `ind [a:x] (n + 1) = x n`

13.0.1 Function(al)s for sequences ("user library")

- Domain specification: "block" $\square$

$$\boxed{\square\, n = \{k : \mathbb{N} \mid k < n\} \quad \text{for} \quad n : \mathbb{N} \text{ or } n := \infty}$$

- Length: $\#$ $\quad \boxed{\#\, x = n \;\equiv\; \mathcal{D}\, x = \square\, n, \text{ equivalently: } \#\, x = \square^{-}\,(\mathcal{D}\, x)}$

- Prefix ($>\!\!-$) and concatenation ($+\!\!+$) characterized by domain and mapping:

$$\boxed{\begin{array}{ll} \#\,(a >\!\!- x) = \#\, x + 1 & i \in \mathcal{D}\,(a >\!\!- x) \Rightarrow (i = 0)\,?\, a \nmid x\,(i - 1) \\ \#\,(x +\!\!+ y) = \#\, x + \#\, y & i \in \mathcal{D}\,(x +\!\!+ y) \Rightarrow (i < \#\, x)\,?\, x\, i \nmid y\,(i - \#\, x) \end{array}}$$

- Shift: σ characterized by domain and mapping: for nonempty x,

$$\boxed{\#\,(\sigma\ x) = \#\, x - 1 \qquad i \in \mathcal{D}\,(\sigma\ x) \Rightarrow \sigma\ x\, i = x\,(i + 1)}$$

- The usual induction principle is a *theorem* (not an axiom)

$$\boxed{\forall\,(x : A^{*}\,.\,P\, x) \;\equiv\; P\, \varepsilon \wedge \forall\,(x : A^{*}\,.\,P\, x \Rightarrow \forall\, a : A\,.\,P\,(a >\!\!- x))}$$

13.1 Origin: deriving data flow circuits from specifications

13.1.0 Motivation for showing this topic: a practical need for point-free formulations

- **Point-free formulations** traditionally seen as only relevant to pure theory.

  Any (general) practical formalism needs both point-wise and point-free style.

- **Example: signal flow systems:** assemblies of interconnected components. Dynamical behavior modelled by functionals from input to output signals.

  Here taken as an opportunity to introduce "embryonic" generic functionals, i.e., arising in a specialized context, to be made generic later for general use.

  Extra feature: LabVIEW (a graphical language) taken as an opportunity for

  - Presenting a language with uncommon yet interesting semantics
  - Using it as one of the application examples of our approach (functional description of the semantics using generic functionals)

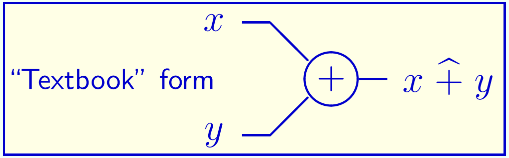
- **Time is not structural**

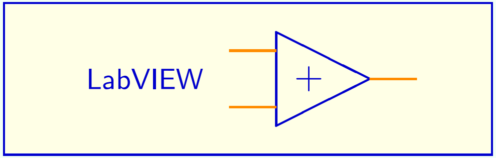
  Hence transformational design = elimination of the time variable

  This was the example area from which our entire formalism emerged.

13.1.1 Basic building blocks for the example

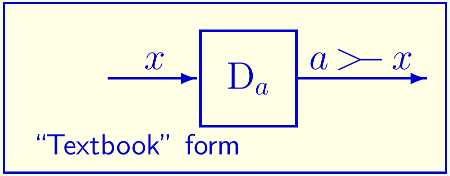
- Memoryless devices realizing arithmetic operations
  - Sum (product, ...) of signals x and y modelled as $\boxed{(x \mathbin{\widehat{+}} y)\,t = x\,t + y\,t}$
  - Explicit *direct extension* operator $\widehat{\ }$ (in engineering often left implicit)

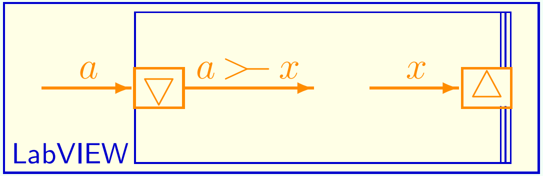




- Memory devices: latches (discrete case), integrators (continuous case)

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





13.1.2 A transformational design example

a. **From specification to realization**

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

$$\boxed{\mathbf{def}\ f : \mathbb{N} \to A\ \mathbf{with}\ f\,n = (n = 0)\,?\,a \dagger g\,(f\,(n-1)) \qquad (93)}$$

- Calculational transformation into the $\boxed{\text{fixpoint equation } f = (\mathrm{D}_a \circ \overline{g}\,)\,f}$

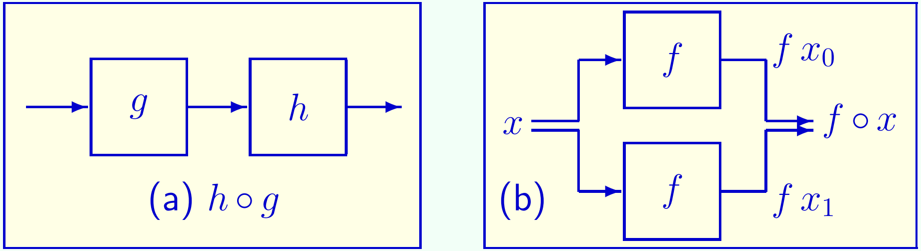
$$\boxed{\begin{aligned}
f\,n =&\quad \langle\text{Def. } f\rangle & (n = 0)\,?\,a \dagger g\,(f\,(n-1))\\
=&\quad \langle\text{Def. } \circ\,\rangle & (n = 0)\,?\,a \dagger (g \circ f)\,(n-1)\\
=&\quad \langle\text{Def. D}\rangle & \mathrm{D}_a\,(g \circ f)\,n\\
=&\quad \langle\text{Def. } \overline{\,-\,}\rangle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

b. **Functionals introduced** (types omitted; designed during the generification)

- Function composition: $\circ$, with mapping $(f \circ g)\,x = f\,(g\,x)$
- Direct extension (1 argument): $\overline{\,-\,}$, with mapping $\overline{g}\ x = g \circ x$

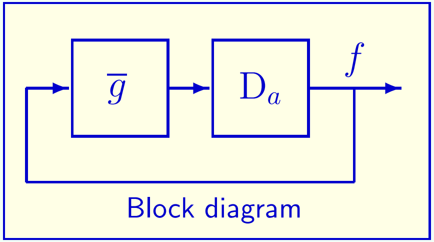
c. **Structural interpretations** of composition and the fixpoint equation

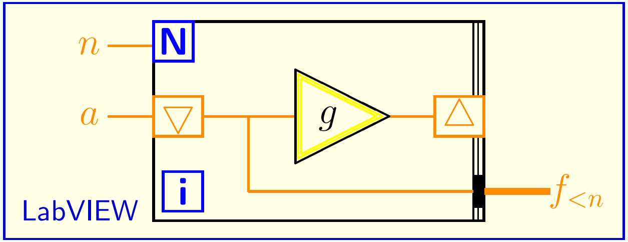
- Structural interpretations of composition: (a) cascading; (b) replication



Example property: $\overline{h \circ g} = \overline{h} \circ \overline{g}$ (proof: exercise)

- Immediate structural solution for the fixpoint equation $\boxed{f = (\mathrm{D}_a \circ \overline{g})\, f}$



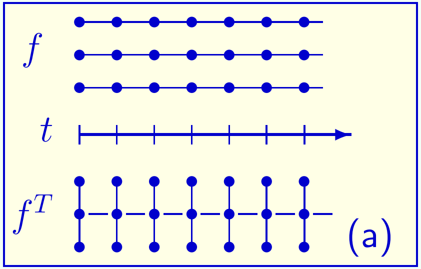


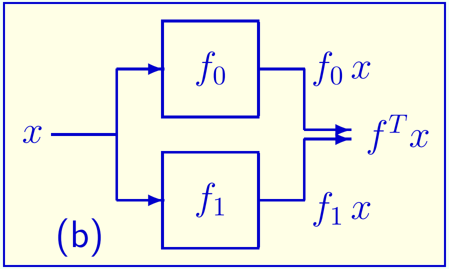
In LabVIEW: extra parameter to obtain prefix of finite length n.

13.1.3 More on the role of generic functionals

a. **A third operator: transposition** (already shown: composition, extension)

- Purpose: swapping the arguments of a functional: $\boxed{f^T y\, x = f\, x\, y}$
  Nomenclature obviously 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 are functions, and up to currying.

b. Calculating with transposition, composition and direct extension

- Duality composition – transposition: assuming x not free in M,

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

Also: $\boxed{f \circ (x, y, z) = f\,x, f\,y, f\,z \quad \text{and} \quad (f, g, h)^T x = f\,x, g\,x, h\,x}$

- Generalizing direct extension to an arbitrary number of arguments:

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

(hints added orally) hence $\boxed{f \mathbin{\widehat{\star}} f' = (\star) \circ (f, f')^T}$ by extensionality.

Therefore we define the generalized direct extension operator $\overset{<}{\rule{1.5em}{0.4pt}}$ by

$$\boxed{\overset{<}{g}\; h = g \circ h^T} \qquad (94)$$

for any function g and any family h of functions.

13.2 Applications in functional programming

Principle: defining objects as functions makes them inherit all generic operators. Moreover, by doing so, nothing is lost (expression styles, calculation rules).

a. **Sequences as functions** with basic *concatenation* operator $\mathbin{+\!\!+}$

$$\boxed{x \mathbin{+\!\!+} y \;=\; i : \square\,(\#\,x + \#\,y)\,.\,(i < \#\,x)\,?\,x\,i \mathbin{\vdash} y\,(i - \#\,x) \qquad \textsf{(95)}}$$

together with the emty function/sequence $\boxed{\varepsilon}$ and the singleton injector $\boxed{\tau}$

$$\boxed{\tau\,x \;\; = 0 \mapsto x} \qquad (96)$$

- Sufficient for derived operators, e.g., prefix $\boxed{>\!\!-}$ and postfix $\boxed{-\!\!<}$

$$\boxed{a \mathbin{>\!\!-} x = \tau\,a \mathbin{+\!\!+} x \quad \text{and} \quad x \mathbin{-\!\!<} a = x \mathbin{+\!\!+} \tau\,a}$$

- Supports existing reasonings, e.g., induction: for any predicate $P : A^* \to \mathbb{B}$,

$$\boxed{\forall\,P \;\;\equiv\;\; P\,\varepsilon \wedge \forall\,x : A^*\,.\,P\,x \Rightarrow \forall\,a : A\,.\,P\,(a \mathbin{>\!\!-} x).}$$

Similarly for infinite structures and coinduction.

b. **Application of generic functionals to sequences**

- Interesting applications of composition and transposition Examples:
  - $\boxed{(0,3,5,7)\circ(2,3,1) = 5,7,3}$ and $\boxed{(0,3,5,7)\circ(2,3,5) = 5,7}$.
  - $\boxed{f\circ(x,y) = f\,x, f\,y}$ subsumes `map f [x, y] = [f x, f y]`.
  - $\boxed{((a,b,c),(a',b',c'))^T = ((a,a'),(b,b'),(c,c'))}$ subsumes `zip [[a,b,c],[a',b',c']] = [[a,a'],[b,b'],[c,c']]`
- Sequences have inverses $\boxed{\text{Example: } (3,3,7)^-\,7 = 2}$
- Some operators over sequences have inverses $\boxed{\text{e.g., } >\!\!\!-^{-}\,(a >\!\!\!- x) = a, x}$

  Application example: $\mathit{head}\,x = {>\!\!\!-}^{-}\,x\,0$ for any nonempty sequence x.
- Pattern matching as equational definition using function inverses

  Recursive definitions like $f\,(a >\!\!\!- x) = h\,(a, f\,x)$: just covered.

  Generalization: definition format $\boxed{f\,(g\,(x,y)) = e\,(x,y)}$.

  Application of f to an actual parameter z satisfies $\boxed{f\,z = e\,(g^-\,z\,0, g^-\,z\,1)}$ for any z in the bijectivity range of g. (also: complex numbers, signals)

c. **Transposition in functional languages**

Defining transposition is usually possible for the special case $\boxed{A \to (B \to C)}$.

Given a function f satisfying $\boxed{f \in A \to (B \to C)}$, the transpose satisfies

$$\boxed{f^T \in B \to (A \to C)}$$

and has property $\boxed{(f^T)^T = f}$.

d. **Direct extension** hasmany application opportunities.

Example (sequences): direct extension as a pairwise map:

$$\boxed{\begin{array}{lrcl} & \mathtt{direx}(*)[[\mathsf{a},\mathsf{b},\mathsf{c}],[\mathsf{a}',\mathsf{b}',\mathsf{c}']] & = & [\mathsf{a}*\mathsf{a}',\mathsf{b}*\mathsf{b}',\mathsf{c}*\mathsf{c}'] \\ \text{Equivalently:} & \mathtt{direx}(*) & = & (\mathtt{map}(*))\ .\ \mathtt{zip} \end{array}}$$

More generally, for generic functionals,

$$\boxed{(\widehat{\star}) = \overline{(\star)} \circ (\&)}$$

13.3 Aggregate data types and structures

a. **Pascal-like records** (ubiquitous in programs) How making them functional?

- Well-known approach: selector functions matching the field labels. (e.g., Haskell)

  Problem: records themselves as arguments, not functions.

- Preferred alternative: generalized functional cartesian product $\times$: records as *functions*, domain: set of field labels from an *enumeration type*. E.g.,

  $$\boxed{Person := \times\,(name \mapsto \mathbb{A}^* \mathbin{\dot\cup} age \mapsto \mathbb{N}),}$$

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

- Syntactic sugar:

  $\boxed{\mathsf{record} : \mathsf{fam}\,(\mathsf{fam}\,\mathcal{T}) \to \mathcal{P}\,\mathcal{F}}$ with $\boxed{\mathsf{record}\,F = \times\,(\dot{\bigcup} F)}$

  Now we can write

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

b. **More about the funcart operator $\times$**

- "Workhorse" for typing all structures unified by functional mathematics.
  - Recall $\boxed{A \rightarrow B = \times(A^{\bullet} B)}$ and $\boxed{A \times B = \times(A, B)}$
  - Array types: for set A and $n : \mathbb{N} \cup \iota\infty$, define

    $$\boxed{A \uparrow n = \square\, n \rightarrow A}$$

    with shorthand A^n. This is the n-fold Cartesian product, since

    $$\boxed{A \uparrow n = \times(\square\, n^{\bullet} A)}$$

  - List types complete the functional unification of aggregates:

    $$\boxed{A^* = \bigcup n : \mathbb{N} \,.\, A^n}$$

- $\times$ Is a genuine functional, not an ad hoc abstractor

  This yields many useful algebraic properties. Most noteworthy: inverse.

  - Choice axiom $\boxed{\times T \neq \emptyset \equiv \forall\, x : \mathcal{D}\,T \,.\, T\,x \neq \emptyset}$ characterizes $\boxed{\text{Bdom}\ \times}$
  - If $\boxed{\times T \neq \emptyset}$, then $\boxed{\times^{-}\,(\times T) = T}$
  - Example: if $A \neq \emptyset$ and $B \neq \emptyset$, then $\boxed{\times^{-}\,(A \times B) = A, B}$, hence

    $$\boxed{\times^{-}(A \times B)\,0 = A \quad \text{and} \quad \times^{-}(A \times B)\,1 = B}$$

  - Explicit image definition: for any nonempty S in the range of $\times$,

    $$\boxed{\times^{-}\,S = x : Dom\,S \,.\, \{f\,x \mid f : S\}} \qquad (97)$$

    where $Dom\,S$ is the common domain of the functions in S extracted, e.g., by $\boxed{Dom\,S = \bigcap f : S \,.\, \mathcal{D}\,f = \bigcap (\mathcal{D} \rceil S)}$.

c. **Other structures** are also defined as functions.

- Example: **trees** are functions whose domains are *branching structures*, i.e., sets of sequences describing the path from the root to a leaf.
  - Covers any branch labeling, e.g., for binary tree: subset of $\mathbb{B}^*$.
  - Classes of trees: specified by restrictions on the branching structures.
  - The $\times$ operator can even specify types for leaves individually.
- Aggregates (as functions) inherit elastic operators for suitable arguments. Example: $\sum s$ sums the "elements" of any number-valued structure s.
- Generic functionals are inherited whatever the image type.

d. **Important for structures: direct extension**

- Dijkstra considers all operators implicitly extended to “structures”
  - Without elaborating the domain (is fixed: the program state space).
  - Even for equality, which becomes a pointwise instead of an “overall” characteristic.
- LabVIEW building blocks are similarly extended (called *polymorphism*).
- Our view
  - Implicit extensions are convenient in a particular area of discourse
  - Broader application range needs finer tuning offered by an explicit generic operator ($^{-}$, $^{\frown}$ or $^{<}$).

13.4 Overloading and polymorphism

- **Terminology**
  - Overloading: using identifier for designating "different" objects.
    Polymorphism: different argument types, formally same image definition.
  - In Haskell: called *ad hoc* and *ad hoc* polymorphism respectively.
  - Considering general overloading also suffices for covering polymorphism.
- **Two main issues in overloading an operator:**
  - Disambiguation of application to all possible arguments
  - Refined typing: reflecting relation between argument and result type

  Covered respectively by:
  - Ensuring that different functions denoted by the operator are *compatible* in the sense of the generic ©-operator.
  - A suitable type operator whose design is discussed next.

- **Option: overloading by explicit parametrization** Trivial with $\times$.
  Example: *binary addition* function adding two binary words of equal length.

  $$\boxed{\mathbf{def}\ binadd\_ : \times n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}\ \mathbf{with}\ binadd_n\,(x, y) = \ldots}$$

  Only the type is relevant. Note: $\boxed{binadd_n \in (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}}$ for any $n : \mathbb{N}$.

- **Option: overloading without auxiliary parameter**
  - $\boxed{\text{Requirement:}}$ operator $\otimes$ with properties $\boxed{\text{exemplified}}$ for $binadd$ by

    $$\boxed{\mathbf{def}\ binadd : \otimes n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}\ \mathbf{with}\ binadd\,(x, y) = \ldots}$$

    - ∗ Note that $\boxed{n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}}$ is a family of function types.
    - ∗ Domain of $binadd$ is $\bigcup n : \mathbb{N} \,.\, (\mathbb{B}^n)^2$, and the type is/should be

      $$\boxed{\bigcup (n : \mathbb{N} \,.\, (\mathbb{B}^n)^2) \ni (x, y) \to \mathbb{B}^{\# x + 1}}$$

    - ∗ This information must be obtainable from $\boxed{\otimes n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}}$.

– An interesting design satisfying the requirement

- ∗ Consider $\boxed{binadd}$ as a *merge* of functions of type $\boxed{(\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}}$ (one for each n).

  The family of functions merged is taken from $\boxed{\times n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}}$

  Requirement: *compatibility*

  (trivially satisfied in this example because of nonintersecting domains).

- ∗ Generalization to arbitrary function type family: yields the desired *function type merge* ($\otimes$) operator

  $$\boxed{\mathbf{def}\ \otimes : \mathsf{fam}\,(\mathcal{P}\,\mathcal{F}) \to \mathcal{P}\,\mathcal{F}\ \mathbf{with}\ \otimes F = \{\dot{\bigcup}\, f \mid f : (\times F)_{\textcircled{c}}\}}$$

  Note: this is not the original form [Van den Beuken], the latter used a non-generic merge to enforce compatibility implicitly.

- ∗ Applications for other purposes than polymorphism shown later.

- **Use in a (declarative) language context** Incremental definition:
  - Regular definitions: $\boxed{\textbf{def}\ x : X\ \textbf{with}\ p_x}$ allow only one definition for x.
  - Overloaded operator definitions: **defo** drops this condition but requires that, for any collection of definitions of the form

    $$\boxed{\textbf{defo}\ f : F_i\ \textbf{with}\ P_i\ f}$$

    (for i in some index set I at the meta-level, but the same f), the derived collection of definitions

    $$\boxed{\textbf{def}\ g_i : F_i\ \textbf{with}\ P_i\ g_i}$$

    is *compatible*: $\boxed{©\, i : I \mathrel{.} g_i}$ (satisfied trivially for nonintersecting domains). Then the given collection defines an operator

    $$\boxed{f : \bigotimes i : I \mathrel{.} F_i \quad \text{with} \quad f = \bigcup i : I \mathrel{.} g_i}$$

- **A rough analogy with Haskell**

  Illustrated by an example from *A Gentle Introduction to Hashell 98*:

```
class Eq a where (==) ::  a -> a -> Bool
instance Eq Integer where x == y = x ‘integerEq‘ y
instance Eq Float where x == y = x ‘floatEq‘ y
```

  This is approximately (and up to currying) rendered by

$$\boxed{\begin{array}{l}\mathbf{def}\ Eq : \mathcal{T} \to \mathcal{P}\,\mathcal{F}\ \mathbf{with}\ Eq\,X = X^2 \to \mathbb{B}\\ \mathtt{defo}\,\mathtt{---}{=}{=}\mathtt{---} : Eq\ Integer\ \mathbf{with}\ x == y\ \equiv\ x\ integerEq\ y\\ \mathtt{defo}\,\mathtt{---}{=}{=}\mathtt{---} : Eq\ Float\ \mathbf{with}\ x == y\ \equiv\ x\ floatEq\ y.\end{array}}$$

  The type of $==$ is $\boxed{Eq\ Integer \otimes Eq\ Float}$ and, nonintersecting domains ensuring compatibility, $\boxed{(==) = (integerEq) \mathbin{\dot{\cup}} (floatEq)}$.

  An essential difference is that Haskell attaches operators to a class.

13.5 Formal semantics (from conventional languages to LabVIEW)

a. **Expressing abstract syntax** (Unification of Meyer's ad hoc conventions)

- For *aggregate constructs* and *list productions*: functional record and *.
  This is $\times$ actually: record $F = \times(\dot{\bigcup} F)$ and $A^* = \bigcup n : \mathbb{N}\,.\, \times(\square\, n \,{}^\bullet A)$.
- For *choice productions* needing disjoint union: generic elastic $\big|$-operator
  For any family F of types,

$$\boxed{\big| F = \bigcup x : \mathcal{D}\, F \,.\, \{x \mapsto y \mid y : F\, x\}} \tag{98}$$

Idea: analogy with $\bigcup F = \bigcup (x : \mathcal{D}\, F \,.\, F\, x) = \bigcup x : \mathcal{D}\, F \,.\, \{y \mid y : F\, x\}$.

Remarks

- Variadic: $\boxed{A \mid B = \big|(A, B) = \{0 \mapsto a \mid a : A\} \cup \{1 \mapsto b \mid b : B\}}$
- Using $i \mapsto y$ rather than the common i, y yields more uniformity.
  Same three type operators can describe directory and file structures.
- For program semantics, disjoint union is often "overengineering".

- Typical examples: (with field labels from an enumeration type)

$$\boxed{\begin{array}{l}
\mathbf{def}\ \mathit{Program} := \textsf{record}\,(\mathit{declarations} \mapsto \mathit{Dlist}, \mathit{body} \mapsto \mathit{Instruction}) \\
\mathbf{def}\ \mathit{Dlist} := D^* \\
\mathbf{def}\ D := \textsf{record}\,(v \mapsto \mathit{Variable}, t \mapsto \mathit{Type}) \\
\mathbf{def}\ \mathit{Instruction} := \mathit{Skip} \cup \mathit{Assignment} \cup \mathit{Compound} \cup \textsf{ etc.}
\end{array}}$$

A few items are left undefined here (easily inferred).

- If disjoint union wanted:

$$\boxed{\mathit{Skip} \mid \mathit{Assignment} \mid \mathit{Compound} \mid \textsf{etc.}}$$

- Instances of programs, declarations, etc. can be defined as

$$\boxed{\mathbf{def}\ p : \mathit{Program}\ \mathbf{with}\ p = \mathit{declarations} \mapsto \mathit{dl} \mathbin{\dot\cup} \mathit{body} \mapsto \mathit{instr}}$$

b. **Semantics** "Functionalizing" Meyer's formulation using generic functionals.

Example: for static semantics, *validity* of declaration lists (no double declarations) and the variable inventory are expressed by

$$\boxed{\begin{array}{l}\mathbf{def}\ Vdcl : Dlist \rightarrow \mathbb{B}\ \mathbf{with}\ Vdcl\, dl = \mathsf{inj}\,(dl^T\, v)\\ \mathbf{def}\ Var : Dlist \rightarrow \mathcal{P}\ Variable\ \mathbf{with}\ Var\, dl = \mathcal{R}\,(dl^T\, v)\end{array}}$$

The *type map* of a valid declaration list (mapping variables to their types) is

$$\boxed{\begin{array}{l}\quad\mathbf{def}\ typmap : Dlist_{Vdcl} \ni dl \rightarrow Var\, dl \rightarrow Tval\ \ \mathbf{with}\\ \qquad\quad typmap\, dl = tval \circ (dl^T\, t) \circ (dl^T\, v)^-\end{array}}$$

Equivalently, $\boxed{typmap\, dl = \bigcup d : \mathcal{R}\, dl \,.\, d\, v \mapsto tval\,(d\, t)}$.

A type map can be used as a context parameter for expressing validity of expressions and instructions, shown next.

c. **Semantics** (continuation) How function *merge* ($\dot{\bigcup}$) obviates case expressions

Example: type (Texp) and type correctness (Vexp) of expressions. Assume

$$\begin{array}{l}
\mathbf{def}\ \mathit{Expr} := \mathit{Constant} \cup \mathit{Variable} \cup \mathit{Applic} \\
\mathbf{def}\ \mathit{Constant} := \mathit{IntCons} \cup \mathit{BoolCons} \\
\mathbf{def}\ \mathit{Applic} := \textsf{record}\,(\mathit{op} \mapsto \mathit{Oper}, \mathit{term} \mapsto \mathit{Expr}, \mathit{term}' \mapsto \mathit{Expr})
\end{array}$$

Letting $\mathit{Tmap} := \bigcup \mathit{dl} : \mathit{Dlist}_{\mathit{Vdcl}} \,.\, \mathit{typmap}\ \mathit{dl}$ and $\mathit{Tval} := \{\mathit{it}, \mathit{bt}, \mathit{ut}\}$, define

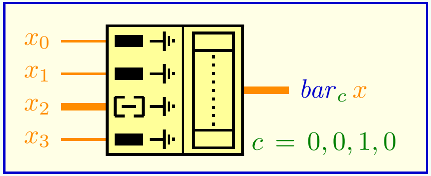
$$\begin{array}{lcl}
\multicolumn{3}{l}{\mathbf{def}\ \mathit{Texp} : \mathit{Tmap} \to \mathit{Expr} \to \mathit{Tval}\ \mathbf{with}} \\
\quad \mathit{Texp}\ \mathit{tm} & = & (c : \mathit{IntCons}\,.\,\mathit{it}) \mathbin{\dot\cup} (c : \mathit{BoolCons}\,.\,\mathit{bt}) \\
 & \dot\cup & (v : \mathit{Variable}\,.\,\mathit{ut}) \olessthan \mathit{tm} \\
 & \dot\cup & (a : \mathit{Applic}\,.\,(a\ \mathit{op} \in \mathit{Arith\_op})\ ?\ \mathit{it} \dagger \mathit{bt})
\end{array}$$

$$\begin{array}{lcl}
\multicolumn{3}{l}{\mathbf{def}\ \mathit{Vexp} : \mathit{Tmap} \to \mathit{Expr} \to \mathbb{B}\ \mathbf{with}} \\
\quad \mathit{Vexp}\ \mathit{tm} & = & (c : \mathit{Constant}\,.\,1) \mathbin{\dot\cup} (v : \mathit{Variable}\,.\,v \in \mathcal{D}\,\mathit{tm}) \\
 & \dot\cup & (a : \mathit{Applic}\,.\,\mathit{Vexp}\ \mathit{tm}\ (a\ \mathit{term}) \wedge \mathit{Vexp}\ \mathit{tm}\ (a\ \mathit{term}')\wedge \\
 & & \mathit{Texp}\ \mathit{tm}\ (a\ \mathit{term}) = \mathit{Texp}\ \mathit{tm}\ (a\ \mathit{term}') \\
 & & = (a\ \mathit{op} \in \mathit{Bool\_op})\ ?\ \mathit{bt} \dagger \mathit{it}))
\end{array}$$

d. **Semantics of data flow languages**

- Already shown: Generic operators for calculations interconnects.
- Semantics: some time ago for Silage (textual), now LabVIEW (graphical)

  Example: LabVIEW block *Build Array* It is generic: configuration parametrizable by menu selection (number and kind of input: *element* or *array*).

  We formalize the configuration by a list in $\mathbb{B}^+$ (0 = *element*, 1 = *array*)

  Semantics example: $\boxed{bar_{0,0,1,0}\,(a, b, (c, d), e) = a, b, c, d, e}$

  Type expression: $\boxed{\mathbb{B}^+ \ni c \to \times\,(i : \mathcal{D}\,c\,.\,(V, V^*)\,(c\,i)) \to V^*}$ (base type V)

  Image definition: $\boxed{bar_c\, x = \mathbin{+\!\!+}\, i : \mathcal{D}\,c\,.\,(\tau\,(x\,i), x\,i)\,(c\,i)}$. Point-free form:

$$\boxed{\begin{array}{l}\textbf{def } bar\_ : \mathbb{B}^+ \ni c \to \bigotimes V : \mathcal{T}\,.\,\times((V, V^*) \circ c) \to V^* \textbf{ with} \\ \qquad bar_c = \mathbin{+\!\!+} \circ \,\|\, ((\tau, id) \circ c).\end{array}}$$

13.6 Relational databases in functional style

a. **Database system** $=$ storing information $+$ convenient user interface
Presentation: offering precisely the information wanted as “virtual tables”.

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
...	...	...	

b. **Relational database** presents the tables as relations.

- Traditional view: rows as tuples (and tuples not seen as functions).
Problem: access only by separate indexing function using numbers.
Patch: “grafting” *attribute names* for column headings.
Disadvantages: model not purely relational, operators on tables ad hoc.
- Functional view; the table rows as *records* using record $F = \times(\dot{\bigcup} F)$
Advantage: embedding in general framework, inheriting algebraic properties and generic operators.

c. **Relational databases as sets of functions using** record $F = \times(\dot{\bigcup} F)$
Example: the table representing *General Course Information*

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
...	...	...	

is declared as $\boxed{GCI : \mathcal{P}\, CID}$, a set of *Course Information Descriptors* with

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

d. **Access to a database:** done by suitably formulated *queries*, such as

(a) Who is the instructor for CS300?

(b) At what time is K. Jason normally teaching a course?

(c) Which courses is R. Barns teaching in the Spring Quarter?

The first query suggests a virtual *subtable* of GCI
The second requires *joining* table GCI with a time table.
All require *selecting* relevant rows.

e. **Formalizing queries**

Basic elements of any *query language* for handling virtual tables:

selection, *projection* and *natural join* [Gries].

Our generic functionals provide this functionality. Convention: record type R.

- $\boxed{\text{Selection } (\sigma)}$ selects in any table $S : \mathcal{P}\, R$ those records satisfying $P : R \to \mathbb{B}$.

  Solution: set filtering $\boxed{\sigma\,(S, P) = S \downarrow P}$.

  Example: $\boxed{GCI \downarrow (r : CID \,.\, r\, \text{code} = \text{CS300})}$ selects the row pertaining to question (a), "Who is the instructor for CS300?".

- $\boxed{\text{Projection } (\pi)}$ yields in any $S : \mathcal{P}\, R$ columns with field names in a set F.

  Solution: restriction $\boxed{\pi\,(S, F) = \{r \rceil F \mid r : S\}}$.

  Example: $\boxed{\pi\,(GCI, \{\text{code}, \text{inst}\})}$ selects the columns for question (a) and $\boxed{\pi\,(GCI \downarrow (r : CID \,.\, r\, \text{code} = \text{CS300}), \iota\, \text{inst})}$ reflects the entire question.

- Join ($\bowtie$) combines tables S, T by uniting the field name sets, rejecting records whose contents for common field names disagree.

  Solution: $\boxed{S \bowtie T = \{ s \mathbin{\dot\cup} t \mid (s,t) : S \times T \mathrel{\dot\land} s \mathbin{\copyright} t \}}$ (function type merge!)

  Example: $\boxed{GCI \bowtie CS}$ combines table GCI with the *course schedule* table CS (e.g., as below) in the desired manner for answering questions

  (b) “At what time is K. Jason normally teaching a course?”

  (c) “Which courses is R. Barns teaching in the Spring Quarter?.

Code	Semester	Day	Time	Location
CS100	Autumn	TTh	10:00	Eng. Bldg. 3.11
MA115	Autumn	MWF	9:00	Pólya Auditorium
CS300	Spring	TTh	11:00	Eng. Bldg. 1.20
. . .	. . .	. . .	. . .	. . .

f. **Algebraic remarks about function type merge**

- Note that $\boxed{S \bowtie T = S \otimes T}$.
- Note also the compatibility property:

$$\boxed{f \copyright g \wedge (f \mathbin{\dot{\cup}} g) \copyright h \;\equiv\; f \copyright g \wedge f \copyright h \wedge g \copyright h}$$

and similarly (by symmetry),

$$\boxed{(g \copyright h) \wedge f \copyright (g \mathbin{\dot{\cup}} h) \;\equiv\; g \copyright h \wedge f \copyright g \wedge f \copyright h}$$

This can be used to show

$$\boxed{\copyright\,(f, g, h) \Rightarrow (f \mathbin{\dot{\cup}} g) \mathbin{\dot{\cup}} h = f \mathbin{\dot{\cup}} (g \mathbin{\dot{\cup}} h)}$$

Hence, although $\dot{\cup}$ is *not* associative, $\otimes$ (and hence $\bowtie$) is associative.

13.7 Final considerations

- **Sobering thought:** even in "functional" programming languages, the most-often used data structures (lists, tuples, etc.) are *not* functions!
  Usual reason: restrictions imposed by the implementation

- **Observations:**
  - Implementation principles and techniques evolve in time
  - Functional view is "downward compatible" with existing definitions
  - Hence this view can still be exploited in *reasoning* about programs!

- **Example** (suggested by Steven Johnson). An interleaving variant of `zip` is

$$\boxed{\texttt{zap [a : x] [b : y]\ \ =\ \ a : b : zap x y}} \tag{99}$$

(intended for infinite sequences). Task: prove that

$$\boxed{\texttt{map f (zap x y) = zap (map f x) (map f y)}} \tag{100}$$

This is usually done by co-induction. Alternative: proving it as an instance of a more general property of generic functionals.

- **Elaboration**

  Recall def. (??), i.e., $\boxed{\mathit{zap}\,(a \mathrel{>\!\!-} x)\,(b \mathrel{>\!\!-} y) = a \mathrel{>\!\!-} b \mathrel{>\!\!-} \mathit{zap}\,x\,y}$

  To prove: (??), i.e., $\boxed{\mathit{map}\,f\,(\mathit{zap}\,x\,y) = \mathit{zap}\,(\mathit{map}\,f\,x)\,(\mathit{map}\,f\,y)}$

  Note: infinite sequences "are" functions with domain $\mathbb{N}$

  - "Quickie" proof for (??) By induction on $\mathbb{N}$, (??) becomes

    $$\boxed{\mathit{zap}\,x\,y\,(2\cdot n) = x\,n \quad \text{and} \quad \mathit{zap}\,x\,y\,(2\cdot n+1) = y\,n}$$

    So (??) follows by the laws for $\circ$ since $\mathit{map}\,f\,x = f \circ x = \overline{f}\,x$.
    Yet: uses a domain variable and case distinction (even, odd argument).

  - Proof from general properties Via $\mathit{zap}\,x\,y\,n = (x\,\frac{n}{2}, y\,\frac{n-1}{2})\,(n \dagger 2)$,

    $$\boxed{\mathit{zap}\,x\,y \quad = \quad \|\,(x \circ \alpha, y \circ \beta)^U \gamma} \qquad (101)$$

    where $\alpha : \{2 \cdot n \mid n : \mathbb{N}\} \to \mathbb{N}$ with $\alpha\,n = n/2$
    and $\quad \beta : \{2 \cdot n + 1 \mid n : \mathbb{N}\} \to \mathbb{N}$ with $\beta\,n = (n-1)/2$,
    and $\quad \gamma : \mathbb{N} \to \mathbb{B}$ with $\gamma\,n = n \dagger 2$.

– Here we prove $\boxed{\mathit{map}\, f\, (\mathit{zap}\, x\, y) = \mathit{zap}\, (\mathit{map}\, f\, x)\, (\mathit{map}\, f\, y) \ \ \textbf{(??)}}$ from $\boxed{\mathit{zap}\, x\, y = \,\|\, (x \circ \alpha, y \circ \beta)^U \gamma \ \ \textbf{(??)}}$ using the following property of the generic functionals: for any function f and function family F

$$\boxed{\overline{\overline{f}}\ (\|\, F^U) \ \ = \ \ \|\, (\overline{\overline{f}}\ F)^U} \tag{102}$$

Proof:

$$\boxed{\begin{array}{rcll}
\mathit{map}\, f\, (\mathit{zap}\, x\, y) & = & \langle \mathit{map}\, f = \overline{f}\,, \textbf{(??)}\rangle & \overline{f}\,(\|\, (x \circ \alpha, y \circ \beta)^U \gamma) \\
 & = & \langle \overline{f}\ (g\, x) = \overline{\overline{f}}\ g\, x\rangle & \overline{\overline{f}}\ (\|\, (x \circ \alpha, y \circ \beta)^U)\, \gamma \\
 & = & \langle \text{Theorem } \textbf{(??)} \rangle & \|\, (\overline{\overline{f}}\ (x \circ \alpha, y \circ \beta))^U\, \gamma \\
 & = & \langle \overline{f}\, (x, y) = f\, x, f\, y\rangle & \|\, (\overline{f}\, (x \circ \alpha), \overline{f}\, (y \circ \beta))^U\, \gamma \\
 & = & \langle \overline{f}\, (g \circ h) = \overline{f}\, g \circ h\rangle & \|\, (\overline{f}\, x \circ \alpha, \overline{f}\, y \circ \beta)^U\, \gamma \\
 & = & \langle \mathit{map}\, f = \overline{f}\,, \textbf{(??)}\rangle & \mathit{zap}\, (\mathit{map}\, f\, x)\, (\mathit{map}\, f\, y)
\end{array}}$$

Remark: more important than the proof of theorem **(??)** itself is establishing theorem **(??)** as a functional generalization thereof.
Also, the fact that defining sequences as functions allows handling “sequences with holes” (such as $x \circ \alpha$ and $y \circ \beta$) effortlessly may also prove useful.

13.8 Conclusion

- Small collection of functionals directly useful in a wide range of applications, from continuous mathematics to programming
- Generic nature and (hence) wide coverage depend on two elements:
  - Unifying view on objects by (re)defining them as functions
  - Judicious specification of the domains for the ‘results’ of by the functionals.

  Additional benefits: point-free style, useful algebraic rules for calculation
- Examples shown for mathematics of software engineering (predicate calculus), aspects of programming languages (formal semantics and unifying design) and quite different application areas (data flow systems and relational data bases).

  Not shown here (yet interesting): examples in non-discrete mathematics.
- Valuable side-effect for organizing human knowledge: exploiting similarities between disparate fields, reducing conceptual and notational overhead, and making the transition easier by capturing analogies formally.