Tutorial — ICTAC 2004

Functional Predicate Calculus and Generic Functionals in Software Engineering

Raymond Boute

INTEC — Ghent University

13:30–13:40	0. Introduction: purpose and approach
13:40–14:30	**Lecture A: Mathematical preliminaries and generic functionals** 1. Preliminaries: formal calculation with equality, propositions, sets 2. Functions and introduction to concrete generic functionals
14:30–15:00	(Half-hour break)
15:00–15:55	**Lecture B: Functional predicate calculus; general applications** 3. Functional predicate calculus: calculating with quantifiers 4. General applications to functions, functionals, relations, induction
15:55–16:05	(Ten-minute break)
16:05–17:00	**Lecture C: Applications in computer and software engineering** 5. Applications of generic functionals in computing science 6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

0 Introduction: Purpose and Approach

0.0 Purpose: strengthening link between theoretical CS and Engineering

- Remark by Parnas:

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

- Observation: difference in practice
  - In classical engineering (electrical, mechanical, civil): established *de facto*
  - In software "engineering": mathematical models rarely used (occasionally in critical systems under the name "Formal Methods")

  C. Michael Holloway: [software designers want to be(come) engineers]

- Causes
  - Different degree of preparation,
  - Divergent mathematical methodology and style

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

    Examples:

From: Blahut / data compacting

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

From: Bracewell / transforms

$$\begin{aligned}
F(s) &= \int_{-\infty}^{+\infty} e^{-|x|} e^{-i2\pi xs} dx \\
&= 2\int_0^{+\infty} e^{-x} \cos 2\pi xs \, dx \\
&= 2 \,\mathrm{Re} \int_0^{+\infty} e^{-x} e^{i2\pi xs} dx \\
&= 2 \,\mathrm{Re} \frac{-1}{i2\pi s - 1} \\
&= \frac{2}{4\pi^2 s^2 + 1}.
\end{aligned}$$

– Major defect: supporting logical arguments highly informal

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

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

0.1 Principle: formal calculation

- Mathematical styles
  - "formal" = manipulating expressions on the basis of their *form*
  - "informal" = manipulating expressions on the basis of their *meaning*
- Advantages of formality
  - Usual arguments: precision, reliability of design etc. well-known
  - Equally (or more) important: *guidance in expression manipulation*
    Calculations guided by the shape of the formulas

    UT FACIANT OPUS SIGNA

    (Maxim of the conferences on *Mathematics of Program Construction*)
- Ultimate goal: *making formal calculation as elegant and practical for logic and computer engineering as shown by calculus and algebra for classical engineering*

Goal has been achieved (illustration; calculation rules introduced later)

Proposition 2.1. for any function $f : \mathbb{R} \not\to \mathbb{R}$, 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 $\forall\, x : S\,.\, |x - a| < \delta \Rightarrow |f\, x - b| < \epsilon$,

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

0.2 Realization of the goal: Functional Mathematics (Funmath)

- Unifying formalism for continuous and discrete mathematics
  - Formalism $=$ notation (language) $+$ formal manipulation rules
- Characteristics
  - Principle: functions as first-class objects and basis for unification
  - Language: very simple (4 constructs only)
    Synthesizes common notations, *without their defects*
    Synthesizes new useful forms of expression, in particular: “point-free”, e.g. $square = times \circ duplicate$ versus $square\, x = x\; times\; x$
  - Formal rules: *calculational*

Why not use “of the shelf” (existing) mathematical conventions?

Answer: too many defects prohibit design of formal calculation rules.

- **Remark:** the need for defect-free notation

  Examples of defects in common mathematical conventions

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

  - Ellipsis, i.e., dots $(\ldots)$ as in $a_0 + a_1 + \cdots + a_n$
    Common use violates Leibniz's principle (substitution of equals for equals)
    Example: $a_i = i^2$ and $n = 7$ yields $0+1+\cdots+49$ (probably not intended!)

  - Summation sign $\sum$ not as well-understood as often assumed.
    Example: error in *Mathematica*: $\sum_{i=1}^{n} \sum_{j=i}^{m} 1 \ = \ \frac{n\cdot(2\cdot m-n+1)}{2}$
    Taking $n := 3$ and $m := 1$ yields 0 instead of the correct sum 1.

  - Confusing function application with the function itself
    Example: $y(t) = x(t) * h(t)$ where $*$ is convolution.
    Causes incorrect instantiation, e.g., $y(t-\tau) = x(t-\tau) * h(t-\tau)$

Examples B: ambiguities in conventions for sets

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

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

  The usual tacit convention is that $\in$ binds x. This seems innocuous, BUT
- Ambiguity is revealed in case p or e is itself of the form $y \in Y$.
  Example: let $Even := \{2 \cdot m \mid m \in \mathbb{Z}\}$ in

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

  Both examples match *both patterns*, thereby illustrating the ambiguity.
- Worse: such defects *prohibit even the formulation of calculation rules*!
  Formal calculation with set expressions rare/nonexistent in the literature.

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

0.3 What you can get out of this

(As for all mathematics: *with regular practice*)

- Ability to calculate with quantifiers as smoothly as usually done with derivatives and integrals

  Note: the same for functionals, pointwise and point-free expressions

- Easier to explore new areas through formalization. Two steps:
  - Formalize concepts using defect-free notation
  - Use formal reasoning to assist "common" intuition

- Also for better understanding other people's work (literature, other sources): formalize while removing defects, use formal calculation for exploration.
  - Traditional student's way: staring at a formula until understanding dawns (if ever)
  - Calculational way: start formal calculation with the formula to "get a feel"

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

1 Formal calculation with equality, propositions, sets

1.0 Simple expressions and equality

1.0.0 Syntax of simple expressions and equational formulas

1.0.1 Substitution and formal calculation with equality

1.1 Pointwise and point-free styles

1.1.0 Lambda calculus as a consistent method for handling dummies

1.1.1 Combinator calculus as an archetype for point-free formalisms

1.2 Calculational proposition logic and binary algebra

1.2.0 Syntax, conventions and calculational logic with implication and negation

1.2.1 Quick calculation rules and proof techniques; rules for derived operators

1.2.2 Binary Algebra and formal calculation with conditional expressions

1.3 Formal calculation with sets via proposition calculus

1.0 Simple expressions and equality

1.0.0 Syntax of simple expressions and equational formulas

a. Syntax of simple expressions Convention: terminal symbols underscored

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

Variables and constants are domain-dependent. Example (arithmetic):

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

Alternative style (Observe the conventions, e.g., $\mathit{variable}$ is a nonterminal, V its syntactic category, v an element of V, mostly using the same initial letter)

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:	$[\![(e \star e')]\!]$	where $\star$ is an operator from C_2 and e and e' expressions

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

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

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: later exercise
- For the time being: IGNORE semantics

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

  The CALCULATION RULES obviate relying on meaning aspects.

1.0.1 Substitution and formal calculation with equality

a. Formalizing substitution
We define a "postfix" operator $[v := d]$ (written after its argument), parametrized by variable v and expression d,
Purpose: $e[v := d]$ is the result of substituting d for w in e.

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

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

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

Remarks

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

b. Example (detailed calculation)

$$\begin{array}{lll}
\multicolumn{3}{l}{(a \cdot succ\, x + y)[x := z \cdot b]} \\
= & \langle\text{Normalize}\rangle & `((a \cdot (succ\, x)) + y)\text{'}[x := (z \cdot b)] \\
= & \langle\text{Rule } \mathbf{S2}\rangle & [\![((a \cdot (succ\, x))[x := (z \cdot b)] + y[x := (z \cdot b)])]\!] \\
= & \langle\text{Rule } \mathbf{S2}\rangle & [\![((a[x := (z \cdot b)] \cdot (succ\, x)[x := (z \cdot b)]) + y[x := (z \cdot b)])]\!] \\
= & \langle\text{Rule } \mathbf{S1}\rangle & [\![((a[x := (z \cdot b)] \cdot (succ\, x[x := (z \cdot b)])) + y[x := (z \cdot b)])]\!] \\
= & \langle\text{Rule } \mathbf{S0}\rangle & [\![((a \cdot (succ\, x[x := (z \cdot b)])) + y[x := (z \cdot b)])]\!] \\
= & \langle\text{Rule } \mathbf{SV}\rangle & `((a \cdot (succ\ (z \cdot b))) + y)\text{'} \\
= & \langle\text{Opt. par.}\rangle & `a \cdot succ\, (z \cdot b) + y\text{'}
\end{array}$$

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

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

c. Formal deduction: general

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

$$\boxed{\dfrac{Prems}{q}}$$

  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 context.)

d. Deduction with equality

The inference rules for equality are:

0. *Instantiation* (strict):	$\dfrac{p}{p[v := e]}$	(α)
1. *Leibniz's principle* (non-strict):	$\dfrac{d' = d''}{e[v := d'] = e[v := d'']}$	(β)
2. *Symmetry of equality* (non-strict):	$\dfrac{e = e'}{e' = e}$	(γ)
3. *Transitivity* of equality (non-strict):	$\dfrac{e = e', e' = e''}{e = e''}$	(δ)

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).

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

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

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 $e_0 = e2$.

1.1 Pointwise and point-free styles

1.1.0 Lambda calculus as a consistent method for handling dummies

a. Syntax of "pure" lambda terms

- Syntax: the expressions are *(lambda) terms*, defined by

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

  We write Λ for the syntactic category. Metavariables: $L\,..\,R$ for terms, u, v, w for variables. Shorthands for certain "useful" terms: **C**, **D**, **I** etc,.

- 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., $\lambda v.MN$ 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 $(\lambda x.(((y(\lambda x.(\lambda y.((xz)(\lambda z.((xy)z))))))y)z))$.

- 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.

b. 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

  (i) In $\lambda x.y(\lambda xy.xz(\lambda z.xyz))yz$, number all occurrences from 0 to 11. Only free occurrences: those of y and z in positions 1, 5, 10, 11.

  (ii) An operator φ for the set of variables that occur free in a term:

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

  Legend: ι for singleton sets, $\cup$ for set union, $\setminus$ for *set difference*.

  (iii) 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.

c. Axioms and substitution rules

- Idea: 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.

- Avoiding name clashes inherent in naïve substitution, as in

$$\begin{array}{rcll} ((\lambda x.(\lambda y.xy))y)x &=& \langle\beta\text{-convers.}\rangle & ((\lambda y.xy)[x := y])x \\ &=& \langle\text{Naïve subst.}\rangle & (\lambda y.yy)x \qquad \text{(wrong!)} \end{array}$$

  Avoidance principle: choice of dummy names is incidental.

- Resulting substitution rules

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

  Variant: $(\lambda v.M)[^w_L = (v = w)\,?\,(\lambda v.M) \nmid (v \notin \varphi\, L)\,?\,(\lambda v.M[^w_L) \nmid (\lambda u.M[^v_u[^w_L)$,
  Checks for name clashes; if there are none, taking new u is unnecessary.

d. Calculation rules and axiom variants

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

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

- 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 $w \notin \varphi\, M$

Certain authors consider α-conversion subsumed by syntactic equality.

- 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 $v \notin \varphi\, M$
(iii)	*Rule ζ* (or **extensionality**): $\dfrac{Mv = Nv}{M = N}$ provided $v \notin \varphi\,(M, N)$

Note: given the basic rules, rule ζ is equivalent to ξ and η combined.
Henceforth we assume all these rules.

e. Redexes and the Church-Rosser property

- 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.
- Normal forms
  - A $\beta\eta$-*normal form* (or *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.
- Church-Rosser property: a term has at most one normal form.

1.1.1 Combinator calculus as an archetype for point-free formalisms

a. Syntax and calculation rules

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

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

- Calculation rules: These 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: $\boxed{\boldsymbol{K} L M = L \quad \text{and} \quad \boldsymbol{S} P Q R = P R (Q R)}$
  - Extensionality: if $ML = NL$ 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 $\boldsymbol{I} := \boldsymbol{S}\boldsymbol{K}\boldsymbol{K}$.

b. Converting lambda terms into combinator terms

- Method (Note: combinators may mix with lambda terms: "Cλ-terms").
  - De-abstractor: for every v, 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 ($\neq v$):	$\widehat{v}w = \boldsymbol{K}w$	(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".

- 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. Start with $\widehat{z}xzy$ separately, to avoid rewriting $\widehat{x}\widehat{y}$.

$$
\begin{array}{rcll}
\widehat{z}xzy & = & \langle\text{Rule S}\rangle & \boldsymbol{S}(\widehat{z}xz)(\widehat{z}y) \\
& = & \langle\text{Rule }\eta\rangle & \boldsymbol{S}x(\widehat{z}y) \\
& = & \langle\text{Rule K}\rangle & \boldsymbol{S}x(\boldsymbol{K}y) \\
\widehat{y}\boldsymbol{S}x(\boldsymbol{K}y) & = & \langle\text{Rule S}\rangle & \boldsymbol{S}(\widehat{y}\boldsymbol{S}x)(\widehat{y}\boldsymbol{K}y) \\
& = & \langle\text{Rule }\eta\rangle & \boldsymbol{S}(\widehat{y}\boldsymbol{S}x)\boldsymbol{K} \\
& = & \langle\text{Rule K}\rangle & \boldsymbol{S}(\boldsymbol{K}(\boldsymbol{S}x))\boldsymbol{K} \\
\widehat{x}\boldsymbol{S}(\boldsymbol{K}(\boldsymbol{S}x))\boldsymbol{K} & = & \langle\text{Rule S}\rangle & \boldsymbol{S}(\widehat{x}\boldsymbol{S}(\boldsymbol{K}(\boldsymbol{S}x)))(\widehat{x}\boldsymbol{K}) \\
& = & \langle\text{Rule S}\rangle & \boldsymbol{S}(\boldsymbol{S}(\widehat{x}\boldsymbol{S})(\widehat{x}\boldsymbol{K}(\boldsymbol{S}x)))(\widehat{x}\boldsymbol{K}) \\
& = & \langle\text{Rule K}\rangle & \boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\widehat{x}\boldsymbol{K}(\boldsymbol{S}x)))(\boldsymbol{K}\boldsymbol{K}) \\
& = & \langle\text{Rule S}\rangle & \boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\boldsymbol{S}(\widehat{x}\boldsymbol{K})(\widehat{x}\boldsymbol{S}x)))(\boldsymbol{K}\boldsymbol{K}) \\
& = & \langle\text{Rule }\eta\rangle & \boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\boldsymbol{S}(\widehat{x}\boldsymbol{K})\boldsymbol{S}))(\boldsymbol{K}\boldsymbol{K}) \\
& = & \langle\text{Rule K}\rangle & \boldsymbol{S}(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{S})(\boldsymbol{S}(\boldsymbol{K}\boldsymbol{K})\boldsymbol{S}))(\boldsymbol{K}\boldsymbol{K})
\end{array}
$$

For *practical* use, we shall define a more convenient set of *generic functionals* which also include types and calculation rules for them. — (this is for later)

1.2 Calculational proposition logic and binary algebra

1.2.0 Syntax, conventions and calculational logic with implication and negation

(Principles familiar, but for practicality we present more calculation rules)

a. The language follows the syntax of simple expressions

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

Variables are chosen near the end of the alphabet, e.g., x, y, z.
Lowercase letters around p, q, r are metavariables standing for propositions.

b. Implication ($\underline{\Rightarrow}$) is at the start the *only* propositional operator (others follow).

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 $(x \Rightarrow y) \Rightarrow z$.

c. Inference rule, axioms and deduction

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

- 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 INS where the premisses are theorems (empty $\mathcal{H}$).

  (iv) 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.

d. Replacing classical formats for deduction by calculational style

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

(i) Typical roof in classical *statement list* style (the numbers are for reference)

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

(ii) Typical proof in classical *sequent* style

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

Criticisms: unnecessary duplications, style far from algebra and calculus

Two steps for achieving calculational style

- Intermediate "stepping stone" (just example, omitting technicalities)

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

- Final step: replace pseudo-calculational "labels" $\Downarrow$ and $\times$ by operators *in* the language (here $\Rightarrow$), via two theorems.
  - Metatheorem, *Transitivity* (T$\Rightarrow$): $\boxed{p \Rightarrow q \,,\, q \Rightarrow r \ \vdash\ p \Rightarrow r}$.
    This subsumes $\Downarrow$-steps since it justifies *chaining* of the form

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

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

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

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

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

$$\boxed{\text{DEDUCTION THEOREM:} \quad \text{If } \mathcal{H} \,\&\, p \vdash q \text{ then } \mathcal{H} \vdash p \Rightarrow q}$$

  - 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

$$\boxed{\text{AXIOM, THE TRUTH CONSTANT:} \quad 1}$$

  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.

h. Completing the calculus with negation ($\neg$), a 1-place function symbol

$$\boxed{\text{AXIOM, CONTRAPOSITIVE:} \quad (\neg\, x \Rightarrow \neg\, y) \Rightarrow y \Rightarrow x \qquad (\mathsf{CP}{\Rightarrow})}$$

Some initial theorems and their proofs

- THEOREM, CONTRADICTORY ANTECEDENTS: $\boxed{\neg\, x \Rightarrow x \Rightarrow y \qquad (\mathsf{CA}{\Rightarrow})}$
  Proof: on sight. Corollary: (meta), contradictory hypotheses: $p\,,\neg\, p \vdash q$
- THEOREM, SKEW IDEMPOTENCY OF "$\Rightarrow$": $\boxed{(\neg\, x \Rightarrow x) \Rightarrow x \qquad (\mathsf{SI}{\Rightarrow})}$
  Proof: subtle; unfortunately seems to need a "rabbit out of a hat".

$$\begin{array}{lll}
1 & \Rightarrow\ \langle \mathsf{CA}{\Rightarrow} \rangle & \neg\, x \Rightarrow x \Rightarrow \neg\,(\neg\, x \Rightarrow x) \\
 & \Rightarrow\ \langle \mathsf{LD}{\Rightarrow} \rangle & (\neg\, x \Rightarrow x) \Rightarrow \neg\, x \Rightarrow \neg\,(\neg\, x \Rightarrow x) \\
 & \Rightarrow\ \langle \mathsf{WC\ by\ CP}{\Rightarrow} \rangle & (\neg\, x \Rightarrow x) \Rightarrow (\neg\, x \Rightarrow x) \Rightarrow x \\
 & \Rightarrow\ \langle \mathsf{AB}{\Rightarrow} \rangle & (\neg\, x \Rightarrow x) \Rightarrow x
\end{array}$$

After this, all further theorems are relatively straightforward.

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

i. 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 $\neg\, 1 \Rightarrow 0$

1.2.1 Quick calculation rules and proof techniques; rules for derived operators

a. Quick calculation rules

- 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,

  $$\boxed{\vdash p[^v_0 \Rightarrow p[^v_1 \Rightarrow p \qquad \text{and, equivalently,} \qquad p[^v_0 \,\&\, p[^v_1 \vdash p}$$

  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.
- An implicative variant of a theorem 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 $\vdash p \Rightarrow v \Rightarrow p[^v_1$

b. Brief summary of derived operators and calculation rules

i. 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:

$$\text{Leibniz:} \quad (x \equiv y) \Rightarrow (p[^v_x \equiv p[^v_y) \qquad (\text{L}\equiv)$$

Some earlier theorems in equational form and some new ones

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 $\neg\, 1 \equiv 0$	

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$\Rightarrow/\equiv$

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

$$\boxed{\text{Axiom, propositional inequality:}\quad (x \not\equiv y) \equiv \neg\,(x \equiv y)}$$

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$\not\equiv/\equiv$

Formulas with (only) $\equiv$ and $\not\equiv$ depend only on even/odd number of occurrences.

iii. 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.,

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

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

1.2.2 Binary Algebra and formal calculation with conditional expressions

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

$$\boxed{a \Cup b \le c \equiv a \le c \wedge b \le c \quad \text{and} \quad c \le a \Cap b \equiv c \le a \wedge c \le b}$$

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

$$\boxed{c \le a \Cup b \equiv c \le a \vee c \le b \quad \text{and} \quad a \Cup b \le c \equiv a \le c \wedge b \le c}$$

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

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

b. 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
$\mathbb{B}$		⊽	$<$		$>$		$\not\equiv$	⊼	$\wedge$	$\equiv$	»	$\Rightarrow$	«	$\Leftarrow$	$\vee$	
$\mathbb{R}'$			$<$		$>$		$\neq$		⩚	$=$	»	$\leq$	«	$\geq$	⩛	

- 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

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

c. Relation to various other algebras and calculi (SKIP IF TIME IS SHORT)
Just some examples, also explaining our preference for $\mathbb{B} := \{0, 1\}$.

- Fuzzy logic, usually defined on the interval $[0, 1]$.
  Basic operators are restrictions of $\Cup$ and $\Cap$ 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.
- Basic arithmetic: $\{0, 1\}$ embedded in numbers without separate mapping. This proves especially useful in software specification and word problems.
- Combinatorics: 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.
  In our formalism, the choice $\{0, 1\}$ makes this unnecessary.
- Modulo 2 arithmetic (as a Boolean ring or Galois field), as in coding theory. Associativity of $\equiv$ is counterintuitive for "logical equivalence" with $\{\mathrm{F}, \mathrm{T}\}$.
  With $\{0, 1\}$ it is directly clear from the equality $(a \equiv b) = (a \oplus b) \oplus 1$, linking it to modulo-2 arithmetic, where associativity of $\oplus$ is intuitive.

d. Calculating with conditional expressions, formally

Design decisions making the approach described here possible.

- Defining tuples as *functions* taking natural numbers as arguments:

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

- Embedding proposition calculus in arithmetic: constants 0, 1 as numbers.
- 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: types ignored for the time being (types give rise to later variants).

Observe the analogy with the lambda combinators:

$$\mathbf{C} := \lambda f x y . f (x y) \quad \text{and} \quad \mathbf{T} := \lambda f x y . f y x$$

Conditionals as binary indexing: definition and calculation rules

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

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

- Deriving calculation rules using the distributivity laws for $—^{\mathsf{U}}$ 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:

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

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

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

- 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:

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

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

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

Proof:

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

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

1.3 Formal calculation with sets via proposition calculus

1.3.0 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 ($\in$) as the basic set operator

- Syntax: $\boxed{e \in X}$ with (normally)
  e any expression, X a set expression (introduced soon)
- Examples: $(p \wedge q) \in \mathbb{B}$ $\quad \pi/2 \in \mathbb{R}$ $\quad f +\!\!\!+\, g \in \times (F +\!\!\!+\, G)$
  Note: $(e \in X) \in \mathbb{B}$ for any e and X

Warning: *never* overload the relational operator $\in$ with binding. So,

$\boxed{\text{Poor syntax are } \ \forall\, x \in X \,.\, p \ \text{ and } \ \{x \in X \mid p\} \ \text{ and } \ \sum n \in \mathbb{N} \,.\, 1/n^2}$

Problem-free are $\boxed{\ \forall\, x : X \,.\, p \ \text{ and } \ \{x : X \mid p\} \ \text{ and } \ \sum n : \mathbb{N} \,.\, 1/n^2}$ (later).

1.3.1 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⇒): $(p \Rightarrow X = Y) \Rightarrow p \Rightarrow (x \in X \equiv x \in Y)$

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* $(x \in X \equiv x \in Y) \Rightarrow X = Y$.

1.3.2 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)

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

- 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: $\boxed{X \subseteq Y \ \equiv\ Y = X \cup Y}$

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

2 Functions and introduction to concrete generic functionals

2.0 Motivation

2.1 Functions as first-class mathematical objects

2.1.0 Rationale of the formulation

2.1.1 Equality for functions

2.1.2 Function expressions

2.2 A first introduction to concrete generic functionals

2.2.0 Principle

2.2.1 Functionals designed generically: first batch (those useful for predicate calculus)

2.2.2 Elastic extensions for generic operators

2.0 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.

2.1 Functions as first-class mathematical objects

2.1.0 Rationale of the formulation

a. Relation to common set-based axiomatizations: a function is *not* a set of pairs, which is just 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 $x \in \mathcal{D}\, f \equiv p \quad (f \notin \varphi\, p)$
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): $n \in \mathcal{D}\, halve \Rightarrow n = double\,(halve\, n)$

2.1.1 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

$$\boxed{f = g \;\Rightarrow\; \mathcal{D}\,f = \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \wedge x \in \mathcal{D}\,g \Rightarrow f\,x = g\,x) \qquad (1)}$$

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

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

$$\boxed{\text{(strict inf. rule)}\ \frac{p \Rightarrow \mathcal{D}\,f = \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x)}{p \Rightarrow f = g} \qquad (2)}$$

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 $\mathcal{D}\,f = \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \Rightarrow f = g$.

2.1.2 Function expressions

a. Four kinds of function expressions: functions as better-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 $v \notin \varphi\, X$)

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

v is a variable, X a set expression, p a proposition, e any expression.
The *filter* $\wedge p$ is optional, and $v : X \,.\, e$ stands for $v : X \wedge 1 \,.\, e$.
Axioms for abstraction (+ substitution rules as in lambda calculus):

Domain axiom:	$d \in \mathcal{D}\,(v : X \wedge p \,.\, e) \equiv d \in X \wedge p[^v_d$	(3)
Mapping axiom:	$d \in \mathcal{D}\,(v : X \wedge p \,.\, e) \Rightarrow (v : X \wedge p \,.\, e)\, d = e[^v_d$	(4)

b. Some examples regarding abstraction

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

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

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

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

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

$$\boxed{X^\bullet e = v:X \,.\, e, \ \text{ assuming } v \not\in \varphi\, e} \tag{5}$$

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

$$\boxed{\varepsilon := \emptyset^\bullet e \quad \text{and} \quad x \mapsto y = \iota\, x^\bullet y} \tag{6}$$

c. Remark Abstractions look unlike common mathematics.

Yet, we shall show their use in synthesizing traditional notations formally correct and more general, *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. Equality for abstractions

Instantiating function equality with $f := v : X \wedge\!\!\!\!\cdot\;\, p \,.\, d$ and $g := v : Y \wedge\!\!\!\!\cdot\;\, q \,.\, e$ yields:

THEOREM, EQUALITY FOR ABSTRACTIONS

By Leibniz: $(v : X \wedge\!\!\!\!\cdot\;\, p \,.\, d) = (v : Y \wedge\!\!\!\!\cdot\;\, q \,.\, e)$

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

By extensionality: (property conveniently separated in 2 parts)

domain part: $v \in X \wedge p \equiv v \in Y \wedge q \vdash (v : X \wedge\!\!\!\!\cdot\;\, p \,.\, e) = (v : Y \wedge\!\!\!\!\cdot\;\, q \,.\, e)$

mapping part: $v \in X \wedge p \Rightarrow d = e \vdash (v : X \wedge\!\!\!\!\cdot\;\, p \,.\, d) = (v : X \wedge\!\!\!\!\cdot\;\, p \,.\, e)$

2.2 A first introduction to concrete generic functionals

2.2.0 Principle

a. Motivation

- In a functional formalism, shared by many more mathematical objects.
- Support point-free formulations and conversion between formulations.
- Avoid restrictions of similar operators in traditional mathematics 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 $\mathcal{D}\, f^- = \mathcal{R}\, f$

b. 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: preserve properties, but for new functionals: exploit design freedom

2.2.1 Functionals designed generically: first batch

a. One function argument (function modifiers, domain modulators)

i. Filtering ($\downarrow$)

Function filtering generalizes η-conversion $f = x : \mathcal{D}\, f \,.\, f\, x$:
For any function f and predicate P,

$$\boxed{f \downarrow P \;=\; x : \mathcal{D}\, f \cap \mathcal{D}\, P \mathbin{\underset{\cdot}{\wedge}} P\, x \,.\, f\, x} \tag{7}$$

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 $\mathbb{R}_{\geq 0}$.

ii. Function restriction ($\rceil$): the usual domain restriction:

$$\boxed{f \rceil X \;=\; f \downarrow (X^{\bullet} 1).} \tag{8}$$

b. Two function arguments (function combiners)

i. Composition ($\circ$) generalizes traditional composition:
For *any* functions f and g (without restriction),

$$\boxed{\begin{array}{lcl} x \in \mathcal{D}\,(f \circ g) & \equiv & x \in \mathcal{D}\,g \wedge g\,x \in \mathcal{D}\,f \\ x \in \mathcal{D}\,(f \circ g) & \Rightarrow & (f \circ g)\,x = f\,(g\,x). \end{array}}$$

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

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

ii. Dispatching ($\&$) and parallel ($\parallel$) For any functions f and g,

$$\boxed{\begin{array}{ll} \mathcal{D}\,(f \,\&\, g) = \mathcal{D}\,f \cap \mathcal{D}\,g & x \in \mathcal{D}\,(f \,\&\, g) \Rightarrow (f \,\&\, g)\,x = f\,x, g\,x \\ \mathcal{D}\,(f \parallel g) = \mathcal{D}\,f \times \mathcal{D}\,g & x \in \mathcal{D}\,(f \parallel g) \Rightarrow (f \parallel g)\,(x, y) = f\,x, g\,y \end{array}} \qquad (9)$$

Equivalently (using abstraction):

$$\boxed{\begin{array}{lcl} f \,\&\, g & = & x : \mathcal{D}\,f \cap \mathcal{D}\,g \,.\, f\,x, g\,x \\ f \parallel g & = & x, y : \mathcal{D}\,f \times \mathcal{D}\,g \,.\, f\,x, g\,y \end{array}}$$

iii. Direct extension

- Duplex direct extension ($\widehat{\ }$) For any infix operator $\star$, functions f, g,

$$\boxed{\begin{array}{lcl} x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) & \equiv & x \in \mathcal{D}\,f \cap \mathcal{D}\,g \wedge (f\,x, g\,x) \in \mathcal{D}\,(\star) \\ x \in \mathcal{D}\,(f \mathbin{\widehat{\star}} g) & \Rightarrow & (f \mathbin{\widehat{\star}} g)\,x = f\,x \star g\,x. \end{array}} \qquad (10)$$

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

iv. Function override ($\otimes$ and $\oslash$) For funcs. f and g, $\boxed{g \otimes f = f \oslash g}$ and

$$\boxed{\begin{array}{rcl} \mathcal{D}\,(f \oslash g) & = & \mathcal{D}\,f \cup \mathcal{D}\,g \\ x \in \mathcal{D}\,(f \oslash g) & \Rightarrow & (f \oslash g)\,x = x \in \mathcal{D}\,f\,?\,f\,x \mid g\,x \end{array}}$$

Equivalently, $\boxed{f \oslash g = x : \mathcal{D}\,f \cup \mathcal{D}\,g \,.\, x \in \mathcal{D}\,f\,?\,f\,x \mid g\,x}$.

v. Function merge ($\Cup$) For any functions f and g,

$$\boxed{\begin{array}{rcl} x \in \mathcal{D}\,(f \Cup g) & \equiv & x \in \mathcal{D}\,f \cup \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \\ x \in \mathcal{D}\,(f \Cup g) & \Rightarrow & (f \Cup g)\,x = x \in \mathcal{D}\,f\,?\,f\,x \mid g\,x. \end{array}}$$

Equivalently,

$\boxed{f \Cup g = x : \mathcal{D}\,f \cup \mathcal{D}\,g \wedge (x \in \mathcal{D}\,f \cap \mathcal{D}\,g \Rightarrow f\,x = g\,x) \,.\, x \in \mathcal{D}\,f\,?\,f\,x \mid g\,x}$.

c. Relational functionals

i. Compatibility (©)

$$\boxed{f \mathbin{©} g \quad \equiv \quad f \rceil \mathcal{D}\, g = g \rceil \mathcal{D}\, f} \tag{11}$$

ii. Subfunction ($\subseteq$)

$$\boxed{f \subseteq g \quad \equiv \quad f = g \rceil \mathcal{D}\, f} \tag{12}$$

iii. Equality ($=$) (already covered; expressed below as a single formula)

Examples of typical algebraic properties

- $\boxed{f \subseteq g \equiv \mathcal{D}\, f \subseteq \mathcal{D}\, g \wedge f \mathbin{©} g}$ and $\boxed{f \mathbin{©} g \Rightarrow f \ogreaterthan g = f \mathbin{\dot\cup} g = f \olessthan g}$
- $\subseteq$ is a partial order (reflexive, antisymmetric, transitive)
- For equality:

$$\boxed{\begin{array}{lcl} f = g & \equiv & \mathcal{D}\, f = \mathcal{D}\, g \wedge f \mathbin{©} g \\ f = g & \equiv & f \subseteq g \wedge g \subseteq f \end{array}}$$

2.2.2 Elastic extensions for generic operators

a. Principle: elastic operators in genreral

- *Elastic operators* (together with function abstraction) replace the usual ad hoc abstractors like $\boxed{\forall\, x : X}$ and $\boxed{\sum_{i=m}^{n}}$ and $\boxed{\lim_{x \to a}}$.

  We shall introduce them (and also entirely new ones) as we proceed.

- An *elastic extension* of an infix operator $\star$ is an elastic operator F satisfying

$$\boxed{x, y \in \mathcal{D}\,(\star) \;\Rightarrow\; F\,(x, y) = x \star y}$$

- Remark: typically an elastic extension F of $\star$ is defined at least for tuples.

  Hence *variadic application*, of the form $x \star y \star z$ (any number of arguments) is *always* defined via an appropriate elastic extension:

$$\boxed{x \star y \star z \;=\; F\,(x, y, z)}$$

b. Elastic extensions for generic operators

i. Function transposition ($—^{\mathsf{T}}$) The image definition is $\boxed{f^{\mathsf{T}} y\, x = f\, x\, y}$.
Making $—^{\mathsf{T}}$ generic requires decision about $\mathcal{D}\, f^{\mathsf{T}}$ for *any* function family f.

- Intersecting variant ($—^{\mathsf{T}}$) Motivation: $\mathcal{D}\,(f \,\&\, g) = \mathcal{D}\, f \cap \mathcal{D}\, g$
  This suggests taking $\mathcal{D}\, f^{\mathsf{T}} = \bigcap x : \mathcal{D}\, f \,.\, \mathcal{D}\,(f\, x)$ or $\mathcal{D}\, f^{\mathsf{T}} = \bigcap\,(\mathcal{D} \circ f)$ and

  $$\boxed{f^{\mathsf{T}} = y : \bigcap\,(\mathcal{D} \circ f) \,.\, x : \mathcal{D}\, f \,.\, f\, x\, y} \qquad (13)$$

  Variadic application Observation: $(g \,\&\, h)\, x\, i = (g, h)\, i\, x$ for $i : \{0, 1\}$.
  Design decision:

  $$\boxed{f \,\&\, g \,\&\, h = (f, g, h)^{\mathsf{T}}}$$

- Uniting variant ($—^{\mathsf{U}}$) Motivation: maximizing domain: $\mathcal{D}\, f^{\mathsf{U}} = \bigcup\,(\mathcal{D} \circ f)$.

  $$\boxed{f^{\mathsf{U}} = y : \bigcup\,(\mathcal{D} \circ f) \,.\, x : \mathcal{D}\, f \wedge y \in \mathcal{D}\,(f\, x) \,.\, f\, x\, y} \qquad (14)$$

ii. Elastic parallel, merge, compatibility, equality (or *function constancy*) (p.m.)

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

3 Functional predicate calculus: calculating with quantifiers

3.0 Deriving basic calculation rules and metatheorems

3.0.0 Predicates and quantifiers: axioms and initial application examples (to functions)

3.0.1 Direct consequences, duality, distributivity and monotonicity rules

3.0.2 Case analysis, generalized Shannon expansion and more distributivity rules

3.0.3 Instantiation, generalization and their use in proving equational laws

3.1 Expanding the toolkit of calculation rules

3.1.0 Observation: trouble-free variants of common notations

3.1.1 Selected rules for $\forall$

3.1.2 Remarks on the one-point rule

3.1.3 Swapping quantifiers/dummies and function comprehension

3.0 Deriving basic calculation rules and metatheorems

3.0.0 Predicates and quantifiers: axioms and initial application examples (to functions)

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

The *quantifiers* $\forall$ and $\exists$ are predicates over predicates defined by

$$\boxed{\text{AXIOMS: } \forall\, P \;\equiv\; P = \mathcal{D}\, P^{\bullet} 1 \quad \text{and} \quad \exists\, P \;\equiv\; P \neq \mathcal{D}\, P^{\bullet} 0} \tag{15}$$

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

Remarks

- Simple definition, intuitively clear to engineers/applied mathematicians. Calculation rules equally obvious, but derived axiomatically soon.
- 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".
- Derivations for some initial rules requires separating "$\equiv$" in "$\Rightarrow$" and "$\Leftarrow$"
  Need for doing so will gradually vanish as the package of rules grows

b. First application example: function equality as a formula

Function equality is pivotal in the quantifier axioms (15).
Conversely, (15) can unite Leibniz (1) and extensionality (2) for functions.

THEOREM, FUNCTION EQUALITY: $f = g \equiv \mathcal{D}\, f = \mathcal{D}\, g \wedge \forall\, (f \mathbin{\widehat{=}} g)$ (16)

PROOF: we show ($\Rightarrow$); the second step ⟨Weakening⟩ is just for saving space.

$$
\begin{array}{lll}
f = g \Rightarrow & \langle \text{Leibniz (1)} \rangle & \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, f \cap \mathcal{D}\, g \Rightarrow f\, x = g\, x) \\
\equiv & \langle p \equiv p = 1 \rangle & \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, f \cap \mathcal{D}\, g \Rightarrow (f\, x = g\, x) = 1) \\
\equiv & \langle \text{Def. } \widehat{\ }\ (10) \rangle & \mathcal{D}\, f = \mathcal{D}\, g \wedge (x \in \mathcal{D}\, (f \mathbin{\widehat{=}} g) \Rightarrow (f \mathbin{\widehat{=}} g)\, x = 1) \\
\equiv & \langle \text{Def. } {}^\bullet\ (5) \rangle & \mathcal{D}\, f = \mathcal{D}\, g \wedge \\
 & & (x \in \mathcal{D}\, (f \mathbin{\widehat{=}} g) \Rightarrow (f \mathbin{\widehat{=}} g)\, x = (\mathcal{D}\, (f \mathbin{\widehat{=}} g)^\bullet\, 1)\, x) \\
\Rightarrow & \langle \text{Extns. (2)} \rangle & \mathcal{D}\, f = \mathcal{D}\, g \wedge (f \mathbin{\widehat{=}} g) = \mathcal{D}\, (f \mathbin{\widehat{=}} g)^\bullet\, 1 \\
\equiv & \langle \text{Def. } \forall\ (15) \rangle & \mathcal{D}\, f = \mathcal{D}\, g \wedge \forall\, (f \mathbin{\widehat{=}} g)
\end{array}
$$

Step ⟨Extens. (2)⟩ tacitly used $\mathcal{D}\, h = \mathcal{D}\, h \cap \mathcal{D}\, ((\mathcal{D}\, h)^\bullet\, 1)$.

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

c. Application: defining function types via quantification

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)

$\rightarrow$	*function arrow*	$f \in X \rightarrow Y \equiv \mathcal{D}\,f = X \wedge \forall\, x : \mathcal{D}\,f\,.\,f\,x \in Y$	(17)
$\not\rightarrow$	*partial arrow*	$f \in X \not\rightarrow Y \equiv \mathcal{D}\,f \subseteq X \wedge \forall\, x : \mathcal{D}\,f\,.\,f\,x \in Y$	(18)

Note: Functions of type $X \not\rightarrow Y$ 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.

- $X \not\rightarrow Y = \bigcup (S : \mathcal{P}\,X\,.\,S \rightarrow Y)$
- $|X \not\rightarrow Y| = \sum (k : 0\,..\,n\,.\,\binom{n}{k} \cdot m^k) = (m+1)^n = |X \rightarrow (Y \cup \iota\,\bot)|$
  for finite X and Y with $n := |X|$ and $m := |Y|$, since $|X \rightarrow Y| = m^n$.

Later we shall define a generic functional for more refined typing

3.0.1 Direct consequences, duality, distributivity and monotonicity rules

a. Direct consequences: elementary properties by "head calculation"

- For constant predicates: $\forall\,(X^{\bullet}1) \equiv 1$ and $\exists\,(X^{\bullet}0) \equiv 0$ (by (5, 15))
- For the empty predicate: $\forall\,\varepsilon \equiv 1$ and $\exists\,\varepsilon \equiv 0$ (since $\varepsilon = \emptyset^{\bullet}1 = \emptyset^{\bullet}0$)

b. $\boxed{\textsc{Theorem, Duality:}\quad \forall\,(\overline{\neg}\,P) \equiv (\overline{\neg}\,\exists)\,P}$ (19)

PROOF:

$$\begin{array}{lcll}
\forall\,(\overline{\neg}\,P) & \equiv & \langle\text{Def. } \forall \text{ (15), Lemma A (20)}\rangle & \overline{\neg}\,P = \mathcal{D}\,P^{\bullet}1 \\
& \equiv & \langle\text{Lemma B (21)}\rangle & P = \overline{\neg}\,(\mathcal{D}\,P^{\bullet}1) \\
& \equiv & \langle\text{Lemma C (22), } 1 \in \mathcal{D}\,\neg\rangle & P = \mathcal{D}\,P^{\bullet}(\neg\,1) \\
& \equiv & \langle\neg\,1 = 0\text{, definition } \exists \text{ (15)}\rangle & \neg\,(\exists\,P) \\
& \equiv & \langle\text{Defin. } \overline{} \text{ and } \exists\,P \in \mathcal{D}\,\neg\rangle & \overline{\neg}\,\exists\,P
\end{array}$$

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

LEMMA A $\quad \mathcal{D}\,(\overline{\neg}\,P) = \mathcal{D}\,P$ (20)

LEMMA B: $\quad \overline{\neg}\,P = Q \equiv P = \overline{\neg}\,Q$ (21)

LEMMA C: $\quad x \in \mathcal{D}\,g \Rightarrow \overline{g}\,(X^{\bullet}x) = g \circ (X^{\bullet}x) = X^{\bullet}(g\,x)$ (22)

c. Some distributivity and monotonicity rules

$$\boxed{\text{THEOREM, COLLECTING } \forall/\wedge: \quad \forall\, P \wedge \forall\, Q \Rightarrow \forall\,(P \mathbin{\widehat{\wedge}} Q)} \tag{23}$$

PROOF: $\forall\, P \wedge \forall\, Q$

$\equiv\langle$Defin. $\forall\rangle$ $\quad P = \mathcal{D}\,P^{\bullet}\,1 \wedge Q = \mathcal{D}\,Q^{\bullet}\,1$

$\Rightarrow\langle$Leibniz$\rangle$ $\quad \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\,(\mathcal{D}\,P^{\bullet}\,1 \mathbin{\widehat{\wedge}} \mathcal{D}\,Q^{\bullet}\,1)$

$\equiv\langle$Defin. $\widehat{\ }\rangle$ $\quad \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\, x : \mathcal{D}\,P \cap \mathcal{D}\,Q\,.\,(\mathcal{D}\,P^{\bullet}\,1)\,x \wedge (\mathcal{D}\,Q^{\bullet}\,1)\,x$

$\equiv\langle$Defin. $^{\bullet}$)$\rangle$ $\quad \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv \forall\, x : \mathcal{D}\,P \cap \mathcal{D}\,Q\,.\,1 \wedge 1$

$\equiv\langle\forall\,(X^{\bullet}\,1)\rangle$ $\quad \forall\,(P \mathbin{\widehat{\wedge}} Q) \equiv 1$

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 $\exists/\vee$:	$\mathcal{D}\,P = \mathcal{D}\,Q \Rightarrow (\exists\,(P \mathbin{\widehat{\vee}} Q) \equiv \exists\, P \vee \exists\, Q)$

d. Properties for equal predicates; monotonicity rules

$$\begin{array}{lrcl}
\textsc{Equal pred.}\ \backslash\forall: & \mathcal{D}\,P = \mathcal{D}\,Q & \Rightarrow & \forall\,(P \mathrel{\widehat{\equiv}} Q) \Rightarrow (\forall\,P \equiv \forall\,Q) & (24)\\
\textsc{Equal pred.}\ \backslash\exists: & \mathcal{D}\,P = \mathcal{D}\,Q & \Rightarrow & \forall\,(P \mathrel{\widehat{\equiv}} Q) \Rightarrow (\exists\,P \equiv \exists\,Q) & (25)\\
\textsc{Monotony}\ \forall/\Rightarrow: & \mathcal{D}\,Q \subseteq \mathcal{D}\,P & \Rightarrow & \forall\,(P \mathrel{\widehat{\Rightarrow}} Q) \Rightarrow (\forall\,P \Rightarrow \forall\,Q) & (26)\\
\textsc{Monotony}\ \exists/\Rightarrow: & \mathcal{D}\,P \subseteq \mathcal{D}\,Q & \Rightarrow & \forall\,(P \mathrel{\widehat{\Rightarrow}} Q) \Rightarrow (\exists\,P \Rightarrow \exists\,Q) & (27)
\end{array}$$

- Proof outlines (intended as exercises with hints):
  - (24) and (25): function equality (16), Leibniz, (T⇒), or monotony.
  - (26): shunting $\forall\,(P \mathrel{\widehat{\Rightarrow}} Q)$ and $\forall\,P$, expanding $\forall\,P$ by (15), Leibniz.
  - (27): from (26) via contrapositivity and duality.
- Importance: crucial in chaining proof steps: (assuming right inclusion)

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

e. 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 calculation yields the correct rules:

$$\boxed{\textsc{Theorem, Constant Predicate under }\forall \quad \forall\,(X^{\bullet}p) \equiv X = \emptyset \vee p} \qquad (28)$$

3.0.2 Case analysis, generalized Shannon expansion and more distributivity rules

a. Case analysis and generalized Shannon

Recall propositional theorems: (i) case analysis $p[^v_0 \wedge p[^v_1 \Rightarrow p$ (or $p[^v_0, p[^v_1 \vdash p$), (ii) Shannon: $p \equiv (v \Rightarrow p[^v_1) \wedge (\neg v \Rightarrow p[^v_0)$ and $p \equiv (v \wedge p[^v_1) \vee (\neg v \wedge p[^v_0)$. We want this power for predicate calculus.

A little technicality: for expression e and variable v, let $\mathsf{D}[^v_e$ denote the domain. Informal definition: largest X s.t. $d \in X \Rightarrow (e[^v_d$ only in-domain applications)

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

$$\boxed{\forall\, P[^v_0 \wedge \forall\, P[^v_1 \;\Rightarrow\; \forall\, P} \tag{29}$$

Theorem, Shannon expansion If $\mathbb{B} \subseteq \mathsf{D}[^v_P$ and $v \in \mathbb{B}$ then

$$\boxed{\begin{array}{lcl} \forall\, P & \equiv & (v \Rightarrow \forall\, P[^v_1) \wedge (\neg\, v \Rightarrow \forall\, P[^v_0) \\ \forall\, P & \equiv & (v \vee \forall\, P[^v_0) \wedge (\neg\, v \vee \forall\, P[^v_1) \\ \forall\, P & \equiv & (v \wedge \forall\, P[^v_1) \vee (\neg\, v \wedge \forall\, P[^v_0) \end{array}} \tag{30}$$

and other variants. Proofs by case analysis.

b. Application examples: deriving more distributivity rules

DISTRIBUTIVITY AND PSEUDODISTRIBUTIVITY THEOREMS (proofs: exercises)

LEFT DISTRIBUTIVITY $\Rightarrow/\forall$:	$p \Rightarrow \forall\, P \equiv \forall\,(p \overrightarrow{\Rightarrow} P)$	(31)
RIGHT DISTRIBUTIVITY $\Rightarrow/\exists$:	$\exists\, P \Rightarrow p \equiv \forall\,(P \overleftarrow{\Rightarrow} p)$	(32)
DISTRIBUTIVITY OF $\vee/\forall$:	$p \vee \forall\, P \equiv \forall\,(p \overrightarrow{\vee} P)$	(33)
PSEUDODISTRIBUTIVITY $\wedge/\forall$:	$(p \wedge \forall\, P) \vee \mathcal{D}\, P = \emptyset \equiv \forall\,(p \overrightarrow{\wedge} P)$	(34)

Note: distributivity rules generalize, e.g., $(r \vee q) \Rightarrow p \;\equiv\; (r \Rightarrow p) \wedge (q \Rightarrow p)$
Pseudodistributivity rules generalize, e.g., $(r \wedge q) \wedge p \;\equiv\; (r \wedge p) \wedge (q \wedge p)$
(in fact: idempotency, associativity and distributivity combined)

Clearly, theorems, (29) and (30) also hold if $\forall$ is replaced everywhere by $\exists$.
This yields a collection of similar laws, also obtainable by duality.
Examples:

LEFT PSEUDODISTR. $\Rightarrow/\exists$:	$(p \Rightarrow \exists\, P) \wedge \mathcal{D}\, P \neq \emptyset \equiv \exists\,(p \overrightarrow{\Rightarrow} P)$
RIGHT PSEUDODIST. $\Rightarrow/\forall$:	$(\forall\, P \Rightarrow p) \wedge \mathcal{D}\, P \neq \emptyset \equiv \exists\,(P \overleftarrow{\Rightarrow} p)$
PSEUDODISTRIBUT. $\vee/\exists$:	$(p \vee \exists\, P) \wedge \mathcal{D}\, P \neq \emptyset \equiv \exists\,(p \overrightarrow{\vee} P)$
DISTRIBUTIVITY OF $\wedge/\exists$:	$p \wedge \exists\, P \equiv \exists\,(p \overrightarrow{\wedge} P)$

3.0.3 Instantiation, generalization and their use in proving equational laws

a. THEOREM, INSTANTIATION AND GENERALIZATION (note: (36), assumes new v)

INSTANTIATION:	$\forall P \Rightarrow e \in \mathcal{D}\, P \Rightarrow P\, e$	(35)
GENERALIZATION:	$p \Rightarrow v \in \mathcal{D}\, P \Rightarrow P\, v \;\vdash\; p \Rightarrow \forall P$	(36)

Proofs: in (1) and (2), let $f := P$ and $g := \mathcal{D}\, P^{\bullet}\, 1$, then apply (5) and (15).

COROLLARY, $\forall$-INTRODUCTION/REMOVAL again assuming new v,

$$\boxed{p \Rightarrow \forall P \text{ is a theorem iff } p \Rightarrow v \in \mathcal{D}\, P \Rightarrow P\, v \text{ is a theorem.}} \qquad (37)$$

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 $P\, v$.

COROLLARY, WITNESS assuming new v

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

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.

b. Proof style: weaving generalization (36) into a calculation chain as follows.

CONVENTION, GENERALIZATION OF THE CONSEQUENT : assuming new v,

$$\boxed{\begin{array}{lll} p & \Rightarrow \langle\text{Calculation to } v \in \mathcal{D}\,P \Rightarrow P\,v\rangle & v \in \mathcal{D}\,P \Rightarrow P\,v \\ & \Rightarrow \langle\text{Generalizing the consequent}\rangle & \forall\,P \end{array}} \qquad (39)$$

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.

c. Application example: proving a very important theorem.

$$\boxed{\text{THEOREM, TRADING UNDER } \forall\text{:} \quad \forall\,P_Q \equiv \forall\,(Q \mathrel{\widehat{\Rightarrow}} P)} \qquad (40)$$

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

$$\boxed{\begin{array}{lll} \forall\,P_Q & \Rightarrow \langle\text{Instantiation (35)}\rangle & v \in \mathcal{D}\,(P_Q) \Rightarrow P_Q\,v \\ & \equiv \langle\text{Definition} \downarrow \text{(7)}\rangle & v \in \mathcal{D}\,P \cap \mathcal{D}\,Q \wedge Q\,v \Rightarrow P\,v \\ & \equiv \langle\text{Shunting } \wedge \text{ to } \Rightarrow\rangle & v \in \mathcal{D}\,P \cap \mathcal{D}\,Q \Rightarrow Q\,v \Rightarrow P\,v \\ & \equiv \langle\text{Axiom } \widehat{\ }\text{, remark}\rangle & v \in \mathcal{D}\,(Q \mathrel{\widehat{\Rightarrow}} P) \Rightarrow (Q \mathrel{\widehat{\Rightarrow}} P)\,v \\ & \Rightarrow \langle\text{Gen. conseq. (39)}\rangle & \forall\,(Q \mathrel{\widehat{\Rightarrow}} P) \end{array}}$$

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

3.1 Expanding the toolkit of calculation rules

3.1.0 Observation: trouble-free variants of common notations

a. Synthesizing or "repairing" common notations via abstraction. Example: let $R := v : X\,.\,r$ and $P := v : X\,.\,p$ in the trading theorem (40) and its dual, then

$$\boxed{\begin{array}{lcl} \forall\,(v : X \mathbin{\dot\wedge} r\,.\,p) & \equiv & \forall\,(v : X\,.\,r \Rightarrow p) \\ \exists\,(v : X \mathbin{\dot\wedge} r\,.\,p) & \equiv & \exists\,(v : X\,.\,r \wedge p). \end{array}} \tag{41}$$

For readers not yet fully comfortable with direct extensions: a *direct* proof for (41) instead of presenting it an instance of the general formulation (40).

$$\begin{array}{ll} \forall\,(v : X \mathbin{\dot\wedge} r\,.\,p) & \\ \Rightarrow\quad \langle \text{Instantiation (35)} \rangle & v \in \mathcal{D}\,(v : X \mathbin{\dot\wedge} r\,.\,p) \Rightarrow (v : X \mathbin{\dot\wedge} r\,.\,p)\,v \\ \equiv\quad \langle \text{Abstraction (3), (4)} \rangle & v \in X \wedge r \Rightarrow p \\ \equiv\quad \langle \text{Shunting } \wedge \text{ to } \Rightarrow \rangle & v \in X \Rightarrow r \Rightarrow p \\ \Rightarrow\quad \langle \text{Gen. conseq. (39)} \rangle & \forall\,(v : X\,.\,r \Rightarrow p) \end{array}$$

The converse follows from the reversion principle.

b. Additional examples: summary of some selected axioms and theorems

Table for $\forall$	General form	Form with $P := v : X . p$ and $v \notin \varphi\, q$
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 $v \notin \varphi\, q$
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.

3.1.1 Selected rules for $\forall$

a. A few more important rules in algebraic style

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:	$\forall S = \forall (\forall \circ S^{\mathsf{C}})$
ONE-POINT RULE:	$\forall P_{=e} \equiv e \in \mathcal{D}\, P \Rightarrow P\, e$

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 $f^{\mathsf{C}} = v : X \,.\, y : Y \,.\, f\,(v, y)$.

b. Similar rules using dummies

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:	$\forall ((x, y) : X \times Y \,.\, p) \equiv \forall (x : X \,.\, \forall y : Y \,.\, p)$
ONE-POINT RULE:	$\forall (x : X \mathbin{\dot\wedge} x = e \,.\, p) \equiv e \in X \Rightarrow p[^x_e$

3.1.2 Remarks om the one-point rule

a. Recall Point-free and pointwise forms for $\forall$:

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

Duals for $\exists$ (both styles):

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

b. 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

c. 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:

$$\forall\, (x : \mathcal{D}\, P \,.\, P\, x \Rightarrow x = e) \;\Rightarrow\; \exists\, P \Rightarrow P\, e \qquad (42)$$

3.1.3 Swapping quantifiers/dummies and function comprehension

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

$$\boxed{\begin{array}{l}\text{THEOREM, HOMOGENEOUS SWAPPING:}\\ \quad \forall\,(x\,{:}\,X\,.\,\forall\,y\,{:}\,Y\,.\,p) \;\equiv\; \forall\,(y\,{:}\,Y\,.\,\forall\,x\,{:}\,X\,.\,p)\\ \quad \exists\,(x\,{:}\,X\,.\,\exists\,y\,{:}\,Y\,.\,p) \;\equiv\; \exists\,(y\,{:}\,Y\,.\,\exists\,x\,{:}\,X\,.\,p)\end{array}} \tag{43}$$

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

$$\boxed{\begin{array}{l}\text{THEOREM, MOVING } \forall \text{ OUTSIDE } \exists\text{:}\\ \quad \exists\,(y\,{:}\,Y\,.\,\forall\,x\,{:}\,X\,.\,p) \;\Rightarrow\; \forall\,(x\,{:}\,X\,.\,\exists\,y\,{:}\,Y\,.\,p)\end{array}} \tag{44}$$

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

$$\boxed{\begin{array}{l}\text{AXIOM, FUNCTION COMPREHENSION:}\\ \quad \forall\,(x\,{:}\,X\,.\,\exists\,y\,{:}\,Y\,.\,p) \;\Rightarrow\; \exists\,f\,{:}\,X \to Y\,.\,\forall\,x\,{:}\,X\,.\,p[^y_{f\,x}.\end{array}} \tag{45}$$

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

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

4 Generic applications to functions, functionals and relations

4.0 Application to functions and functionals

4.0.0 The function range operator

4.0.1 Application of the function range operator to set comprehension

4.0.2 Defining and reasoning about generic functionals — examples

4.0.3 Designing a generic functional for specifying functions within a given tolerance

4.1 Calculating with relations

4.1.0 Characterizing properties of relations

4.1.1 Calculational reasoning about extremal elements — an example

4.2 Induction principles

4.2.0 Well-foundedness and supporting induction

4.2.1 Particular instances of well-founded induction

4.0 Application to functions and functionals

4.0.0 The function range operator

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

$$\boxed{\textsc{Axiom, Function Range:} \quad e \in \mathcal{R}\, f \;\equiv\; \exists\,(x : \mathcal{D}\, f\, .\, f\, x = e)} \qquad (46)$$

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

Examples (exercises):

- Assuming $\subseteq$ is defined by $Z \subseteq Y \equiv \forall\, z : Z\, .\, z \in Y$,
  - $\forall\, x : \mathcal{D}\, f\, .\, f\, x \in \mathcal{R}\, f$
  - $\mathcal{R}\, f \subseteq Y \equiv \forall\, x : \mathcal{D}\, f\, .\, f\, x \in Y$
- Proving for the function arrow: $\boxed{f \in X \to Y \equiv \mathcal{D}\, f = X \wedge \mathcal{R}\, f \subseteq Y}$

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 $\mathcal{D}\, P \subseteq \mathcal{R}\, f \Rightarrow \forall (P \circ f) \Rightarrow \forall P$	(47)
FOR $\exists$:	$\exists (P \circ f) \Rightarrow \exists P$ and $\mathcal{D}\, P \subseteq \mathcal{R}\, f \Rightarrow \exists P \Rightarrow \exists (P \circ f)$	(48)

Remarks on point-wise forms of the composition theorem (47)

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

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

Proof of the composition theorem: next page.

Observation: in applications, proofs are dominantly purely equational, i.e., no (inelegant!) separation of “$\equiv$” in “$\Rightarrow$” and “$\Leftarrow$”

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.

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

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

Proof for part (i)

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

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

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

4.0.1 Application of the range operator to set comprehension

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

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 (46).

- 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

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

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

$$\boxed{\textsc{Theorem, Set comprehension:}\quad e \in \{x : X \mid p\} \equiv e \in X \wedge p[^x_e \qquad (49)}$$

$$\begin{array}{lll} \textsc{Proof:}\quad e \in \{x : X \wedge p \,.\, x\} & \equiv \langle \text{Function range (46)} \rangle & \exists\, x : X \wedge p \,.\, x = e \\ & \equiv \langle \text{Trading, one-pt rule} \rangle & e \in X \wedge p[^x_e \end{array}$$

4.0.2 Defining and reasoning about generic functionals — examples

We define some generic functionals announced earlier (requiring quantification)

a. Generic inversion ($—^{-}$) For any function f,

$$\boxed{\mathcal{D}\, f^{-} = \mathsf{Bran}\, f \quad \text{and} \quad x \in \mathsf{Bdom}\, f \Rightarrow f^{-}\,(f\, x) = x.} \tag{50}$$

For Bdom (*bijectivity domain*) and Bran (*bijectivity range*):

$$\mathsf{Bdom}\, f = \{x : \mathcal{D}\, f \mid \forall\, x' : \mathcal{D}\, f\, .\, f\, x' = f\, x \Rightarrow x' = x\} \tag{51}$$

$$\mathsf{Bran}\, f = \{f\, x \mid x : \mathsf{Bdom}\, f\}. \tag{52}$$

Note that, if the traditional injectivity condition is satisfied, $\mathcal{D}\, f^{-} = \mathcal{R}\, f$.

b. Elastic compatibility For any function family f

$$\boxed{\copyright\, f \quad \equiv \quad \forall\,(x, y) : (\mathcal{D}\, f)^2\, .\, f\, x \copyright f\, y} \tag{53}$$

c. Elastic merge For any function family f,

$$\boxed{\begin{array}{c} y \in \mathcal{D}\,(\dot{\bigcup} f) \quad \equiv \\ y \in \bigcup\,(\mathcal{D} \circ f) \wedge \forall\,(x, x') : (\mathcal{D}\,f)^2 \,.\, y \in \mathcal{D}\,(f\,x) \cap \mathcal{D}\,(f\,x') \Rightarrow f\,x\,y = f\,x'\,y \\ y \in \mathcal{D}\,(\dot{\bigcup} f) \quad \Rightarrow \quad \forall\,x : \mathcal{D}\,f \,.\, y \in \mathcal{D}\,(f\,x) \Rightarrow \dot{\bigcup} f\,y = f\,x\,y \end{array}} \tag{54}$$

Some interesting properties whose calculational proofs are good practice:

- Construction and inversion by merging For any function f,

$$\boxed{f = \dot{\bigcup}\,x : \mathcal{D}\,f \,.\, x \mapsto f\,x \qquad \text{and} \qquad f^{-} = \dot{\bigcup}\,x : \mathcal{D}\,f \,.\, f\,x \mapsto x}$$

  (illustrates how generic design leads to fine operator intermeshing)

- Conditional associativity of merging In general, $\dot{\cup}$ is not associative, but

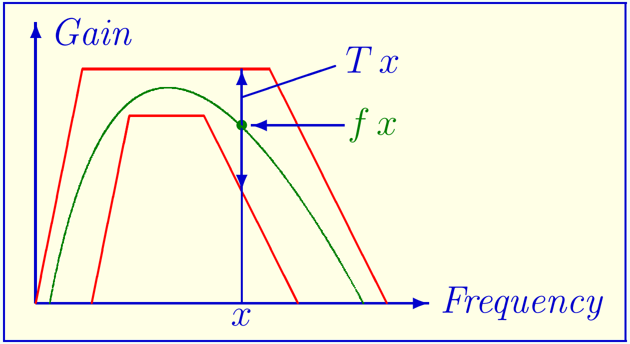
$$\boxed{\copyright\,(f, g, h) \quad \Rightarrow \quad (f \mathbin{\dot{\cup}} g) \mathbin{\dot{\cup}} h = f \mathbin{\dot{\cup}} (g \mathbin{\dot{\cup}} h)}$$

4.0.3 Designing a generic functional for specifying functions within a given tolerance

a. The function approximation paradigm

- Purpose: formalizing *tolerances* for *functions*
- Principle: *tolerance function* T specifies for every domain value x the set $T\,x$ of allowable values. Note: $\mathcal{D}\,T$ supplies the domain specification.

  Example: RF (radio frequency) filter characteristic

  Formalized: a function f *meets* tolerance T iff

  $$\boxed{\mathcal{D}\,f = \mathcal{D}\,T \;\wedge\; (x \in \mathcal{D}\,f \cap \mathcal{D}\,T \Rightarrow f\,x \in T\,x)}$$

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

$$\boxed{f \in \times T \;\equiv\; \mathcal{D}\, f = \mathcal{D}\, T \wedge \forall\, x : \mathcal{D}\, f \cap \mathcal{D}\, T \,.\, f\, x \in T\, x} \qquad (55)$$

  Immediate properties of (55):

  - 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"

$$\boxed{f = g \equiv f \in \times (\iota \circ g)}$$

  - (Semi-)pointfree form:

$$\boxed{\times T = \{f : \mathcal{D}\, T \to \bigcup T \mid \forall (f \mathrel{\widehat{\in}} T\}}$$

b. Commonly used concepts as particularizations

- Usual Cartesian product is defined by $(x, y) \in X \times Y \equiv x \in X \wedge y \in Y$
  Letting $T := X, Y$ in (55), calculation shows how this is captured by $\bigtimes$

  $$\boxed{\bigtimes(X, Y) = X \times Y}$$

  Variadic application of $\times$ defined by $X \times Y \times Z = \bigtimes(X, Y, Z)$
- The common function arrow: letting $T := X^{\bullet} Y$

  $$\boxed{\bigtimes(X^{\bullet} Y) = X \to Y}$$

- Dependent types: letting $T := x : X \,.\, Y_x$ in (55).
  Convenient shorthand: $X \ni x \to Y_x$ for $\bigtimes x : X \,.\, Y_x$

c. Inverse of $\bigtimes$ Interesting explicit formula: for nonempty S in $\mathcal{R}\,\bigtimes$

$$\boxed{\bigtimes^{-} S \;=\; x : \bigcap (f : S \,.\, \mathcal{D}\, f) \,.\, \{f\, x \mid f : S\}}$$

4.1 Calculating with relations

4.1.0 Characterizing properties of relations

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

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

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	$\forall\,(x,y,z) : X^3 \,.\, x\,R\,y \Rightarrow y\,R\,z \Rightarrow x\,R\,z$
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
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: $\boxed{\mathit{ismin}\_\ \text{with}\ x\ \mathit{ismin}_\prec\ S \ \equiv\ x \in S \wedge \forall\, y : X\,.\,y \prec x \Rightarrow y \notin S}$

- 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 (helps 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

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

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

4.1.1 Calculational reasoning about extremal elements — an example

DEFINITION, MONOTONICITY:

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

$$\forall (x, y) : X^2 . x \prec y \Rightarrow P\, x \Rightarrow P\, y \tag{56}$$

THEOREM, PROPERTIES OF EXTREMAL ELEMENTS:

For any $\text{—}\prec\text{—} : \mathsf{rel}_X$ 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

$$\mathsf{lst}_\prec P\, x \wedge \mathsf{lst}_\prec P\, y \Rightarrow x = y \text{ (uniqueness).} \tag{57}$$

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

Proof (samples). 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.

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

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

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

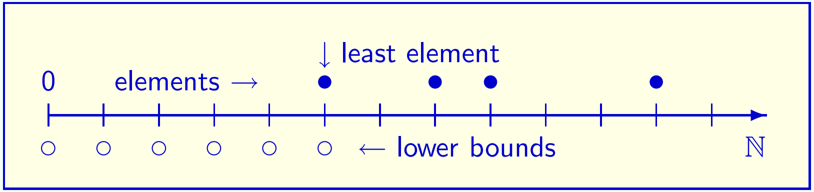
4.1.2 A case study: when is a greatest lower bound a least element? SKIP

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 $\bigwedge$, defined as the particularization of $\bigsqcap$ 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. (exercise)

4.2 Induction principles

4.2.0 Well-foundedness and supporting induction

a. DEFINITION, WELL-FOUNDEDNESS (58)

A relation $\prec : X^2 \to \mathbb{B}$ is *well-founded* iff every nonempty subset of X has a minimal element.

$$\boxed{\mathsf{WF}\,(\prec) \;\equiv\; \forall\, S : \mathcal{P}\, X \,.\, S \neq \emptyset \Rightarrow \exists\, x : X \,.\, x\; ismin_{\prec}\; S}$$

b. DEFINITION, SUPPORTING INDUCTION (59)

A relation $\prec\; : X^2 \to \mathbb{B}$ *supports induction* iff SI $(\prec)$, wwith definition

$$\boxed{\mathsf{SI}\;(\prec) \;\equiv\; \forall\, P : \mathsf{pred}_X \,.\, \forall\,(x : X \,.\, \forall\,(y : X_{\prec\, x} \,.\, P\, y) \Rightarrow P\, x) \Rightarrow \forall\, x : X \,.\, P\, x}$$

c. $\boxed{\text{THEOREM, EQUIVALENCE OF WF AND SI:}\quad \mathsf{WF}\,(\prec) \equiv \mathsf{SI}\;(\prec)}$ (60)

PROOF: next slide

$$
\begin{array}{ll}
& \mathsf{WF}\,(\prec) \\
\equiv & \langle \text{Definition WF (58) and } S \neq \emptyset \equiv \exists\, x : S \,.\, 1 \rangle \\
& \forall\, S : \mathcal{P}\, X \,.\, \exists\,(x : S \,.\, 1) \Rightarrow \exists\,(x : X \,.\, x \mathrel{ismin_{\prec}} S) \\
\equiv & \langle S = X \cap S,\ \text{trading} \rangle \\
& \forall\, S : \mathcal{P}\, X \,.\, \exists\,(x : X \,.\, x \in S) \Rightarrow \exists\,(x : X \,.\, x \mathrel{ismin_{\prec}} S) \\
\equiv & \langle \text{Definition } ismin \rangle \\
& \forall\, S : \mathcal{P}\, X \,.\, \exists\,(x : X \,.\, x \in S) \Rightarrow \exists\,(x : X \,.\, x \in S \wedge \forall\, y : X \,.\, y \prec x \Rightarrow y \notin S) \\
\equiv & \langle p \Rightarrow q \equiv \neg\, q \Rightarrow \neg\, p \rangle \\
& \forall\, S : \mathcal{P}\, X \,.\, \neg\,(\exists\, x : X \,.\, x \in S \wedge \forall\, y : X \,.\, y \prec x \Rightarrow y \notin S) \Rightarrow \neg\,(\exists\, x : X \,.\, x \in S) \\
\equiv & \langle \text{Duality } \forall/\exists,\ \text{De Morgan} \rangle \\
& \forall\, S : \mathcal{P}\, X \,.\, \forall\,(x : X \,.\, x \notin S \vee \neg\,(\forall\, y : X \,.\, y \prec x \Rightarrow y \notin S)) \Rightarrow \forall\, x : X \,.\, x \notin S \\
\equiv & \langle \vee \text{ to } \Rightarrow,\ \text{i.e., } a \vee \neg\, b \equiv b \Rightarrow a \rangle \\
& \forall\, S : \mathcal{P}\, X \,.\, \forall\,(x : X \,.\, \forall\,(y : X \,.\, y \prec x \Rightarrow y \notin S) \Rightarrow x \notin S) \Rightarrow \forall\, x : X \,.\, x \notin S \\
\equiv & \langle \text{Change of variables: } S = \{x : X \mid \neg\,(P\, x)\} \text{ and } P\, x \equiv x \notin S \rangle \\
& \forall\, P : X \to B \,.\, \forall\,(x : X \,.\, \forall\,(y : X \,.\, y \prec x \Rightarrow P\, y) \Rightarrow P\, x) \Rightarrow \forall\, x : X \,.\, P\, x \\
\equiv & \langle \text{Trading, def. SI (59)} \rangle \\
& \mathsf{SI}\,(\prec)
\end{array}
$$

4.2.1 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 (60) 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 (59) 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 $P : A^* \to \mathbb{B}$,

$$\forall\,(x : A^* \,.\, P\,x) \;\equiv\; P\,\varepsilon \wedge \forall\,(x : A^* \,.\, P\,x \Rightarrow \forall\,a : A \,.\, P\,(a \succ\!\!- x)) \qquad (61)$$

Suffices for proving most properties about functional programs with lists.

- Other example: structural induction over *expressions*

Assuming a lambda-term like syntax with conventions as in lecture 1,

THEOREM, STRUCTURAL INDUCTION OVER EXPRESSIONS

For any predicate $P : E \to \mathbb{B}$ on expressions

$$\begin{aligned}
&\forall\,(e : E \,.\, P\,e) \;\equiv\\
&\quad \forall\,(c : C \,.\, P\,c) \;\wedge\; \forall\,(v : V \,.\, P\,v) \wedge\\
&\quad \forall\,(v : V \,.\, \forall\,e : E \,.\, P\,[\![(\lambda v.e)]\!]) \wedge\\
&\quad \forall\,(e, e') : E^2 \,.\, P\,e \wedge P\,e' \Rightarrow P\,[\![(e\,e')]\!] \wedge\\
&\qquad\qquad\qquad\qquad \forall\,(\star : C'' \,.\, P\,[\![(e \star e')]\!])
\end{aligned}$$

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

5 Applications of generic functionals in computing science

Principle: generic functionals are inherited by all objects defined as functions

This will be explained for various topics:

5.0 Application to sequences

5.1 Application to overloading and polymorphism

5.2 Application to aggregate types

5.3 Application: relational databases in functional style

5.4 Application to hardware — examples and analogies

5.0 Application to sequences

5.0.0 Sequences as functions

- 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, e.g.,
    - recursive definition: $[\,]$ is a list and, if x is a list, so is $\mathit{cons}\,a\,x$
    - index function separate: $\mathit{ind}\,(\mathit{cons}\,a\,x)\,(n+1) = \mathit{ind}\,x\,n$ e.g., Haskell: `ind [a:x] 0 = a  and  ind [a:x] (n + 1) = x n`
  - in the few exceptions, functional properties left unexploited.

- Examples of (usually underexploited) functional properties of sequences
  - Function inverse: $\boxed{(a,b,c,d)^{-}\,c = 2}$ (provided $c \not\in \{a,b,d\}$)
  - Composition: $\boxed{(0,3,5,7)\circ(2,3,1) = 5,7,3 \text{ and } f\circ(x,y) = f\,x, f\,y}$
  - Transposition: $\boxed{(f,g)^{\mathsf{T}}\,x = f\,x, g\,x}$

5.0.1 Typing operators for sequences

- Domain specification: "block" $\square$: for $n : \mathbb{N}'$ where $\mathbb{N}' := \mathbb{N} \cup \iota\,\infty$,

$$\boxed{\square\, n = \{k : \mathbb{N} \mid k < n\}}$$

- Array types: for set X and $n : \mathbb{N} \cup \iota\infty$, define

$$\boxed{X \uparrow n = \square\, n \to X}$$

  Shorthand: X^n. This is the n-fold Cartesian product: $\boxed{X \uparrow n = \times(\square\, n \bullet X)}$

- List types (finite length by definition)

$$\boxed{X^* = \bigcup n : \mathbb{N} \,.\, X^n}$$

- Including infinite sequences: $\boxed{X^\omega = X^* \cup X^\infty}$ recalling that $X^\infty = \mathbb{N} \to X$.
- More general sequence types are covered by our "workhorse" $\times$.

5.0.2 Function(al)s for sequences ("user library")

- 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)

$$\boxed{\forall\,(x : A^* \,.\, P\,x) \;\equiv\; P\,\varepsilon \,\wedge\, \forall\,(x : A^* \,.\, P\,x \Rightarrow \forall\,a : A \,.\, P\,(a >\!\!- x))}$$

(well-foundedness is provable)

5.1 Application to overloading and polymorphism

5.1.0 Basic concepts

a. 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.

b. 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.

5.1.1 Options: overloading with/without parametrization

a. Overloading by explicit parametrization (Church style) Trivial with $\times$.
   Example: *binary addition* function adding two binary words of equal length.

$$\boxed{\textbf{def}\ binadd\_ : \times n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}\ \textbf{with}\ binadd_n\,(x, y) = \ldots}$$

   Only the type is relevant. Note: $\boxed{binadd_n \in (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}}$ for any $n : \mathbb{N}$.

b. Option: overloading without auxiliary parameter (Curry style)

   Requirement: operator $\otimes$ with properties $\boxed{\text{exemplified}}$ for $binadd$ by

$$\boxed{\textbf{def}\ binadd : \otimes n : \mathbb{N} \,.\, (\mathbb{B}^n)^2 \to \mathbb{B}^{n+1}\ \textbf{with}\ binadd\,(x, y) = \ldots}$$

   Design: (after considerable work) *function type merge* operator ($\otimes$)

$$\boxed{\textbf{def}\ \otimes : \mathsf{fam}\,(\mathcal{P}\,\mathcal{F}) \to \mathcal{P}\,\mathcal{F}\ \textbf{with}\ \otimes F = \{\dot{\bigcup}\, f \mid f : (\times F)_{©}\}}$$

   Clarification: for sets G and H of functions: $G \otimes H = \otimes\,(G, H)$
   or, elaborating, $G \otimes H = \{g \mathbin{\dot\cup} h \mid g, h : G \times H \wedge g \mathbin{©} h\}$

5.2 Applications to aggregate types

a. Pascal-like records (ubiquitous in programs) How making them functional?

- Well-known approach: selector functions matching the field labels.
  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. Other structures are also defined as functions (e.g., trees).

5.3 Application: 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
...	...	...	

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.

b. Relational database recast from traditional view into functional view

- Traditional view: tables as relations, rows as 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 $F = \times(\bigcup F)$
  Advantage: embed in general framework, inherit properties, operators.
  Relational databases as sets of functions: use record $F = \times(\bigcup F)$

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

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

c. 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: $\boxed{GCI \downarrow (r : CID\,.\,r\,\mathsf{code} = \mathsf{CS300})}$ 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: $\boxed{\pi\,(GCI, \{\mathsf{code}, \mathsf{inst}\})}$ selects the columns for question (a) and $\boxed{\pi\,(GCI \downarrow (r : CID\,.\,r\,\mathsf{code} = \mathsf{CS300}), \iota\,\mathsf{inst})}$ reflects the entire question.

- $\boxed{\text{Join } (\bowtie)}$ combines tables S, T by uniting the field name sets, rejecting records whose contents for common field names disagree.

  Solution: $\boxed{S \bowtie T = \{s \mathbin{\dot\cup} t \mid (s,t) : S \times T \wedge s \mathbin{©} t\}}$ (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
...	...	...	...	...

  Algebraic remarks Note that $\boxed{S \bowtie T = S \otimes T}$ (function type merge).

  One can show $©\,(f,g,h) \Rightarrow (f \mathbin{\dot\cup} g) \mathbin{\dot\cup} h = f \mathbin{\dot\cup} (g \mathbin{\dot\cup} h)$

  Hence, although $\dot\cup$ is *not* associative, $\otimes$ (and hence $\bowtie$) is associative.

5.4 Application to hardware — examples and analogies

5.4.0 Motivation: a practical need for point-free formulations

- 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, made generic afterwards for general use.

  Extra feature: LabVIEW (a graphical language) taken as an opportunity for

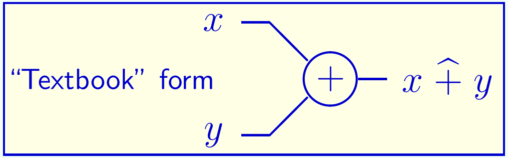
  - Presenting a language with uncommon yet interesting semantics
  - Using it as one of the application examples of our approach (functional description of the semantics using generic functionals)
- Time is not structural

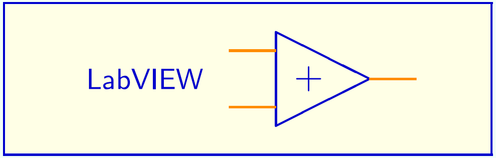
  Hence transformational design = elimination of the time variable

  This was the example area from which our entire formalism emerged.

5.4.1 Basic building blocks for the example

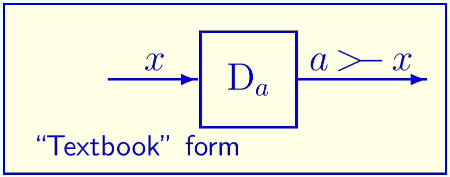
- Memoryless devices realizing arithmetic operations
  - Sum (product, ...) of signals x and y modelled as $\boxed{(x \mathbin{\widehat{+}} y)\,t = x\,t + y\,t}$
  - Explicit *direct extension* operator $\widehat{\ }$ (in engineering often left implicit)

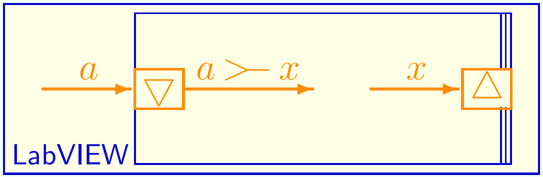




- Memory devices: latches (discrete case), integrators (continuous case)

$$\boxed{\mathrm{D}_a\,x\,n = (n = 0)\,?\,a \mid x\,(n-1) \text{ or, without time variable, } \mathrm{D}_a\,x = a \rightarrowtail x}$$





5.4.2 A transformational design example

a. From specification to realization

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

$$\boxed{\textbf{def}\ f : \mathbb{N} \to A\ \textbf{with}\ f\,n = (n = 0)\,?\,a \nmid g\,(f\,(n-1)) \qquad (62)}$$

- Calculational transformation into the $\boxed{\text{fixpoint equation } f = (\mathrm{D}_a \circ \overline{g}\,)\,f}$

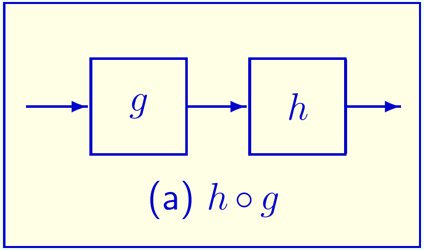
$$\begin{array}{rll} f\,n = & \langle \text{Def. } f \rangle & (n = 0)\,?\,a \nmid g\,(f\,(n-1)) \\ = & \langle \text{Def. } \circ \rangle & (n = 0)\,?\,a \nmid (g \circ f)\,(n-1) \\ = & \langle \text{Def. D} \rangle & \mathrm{D}_a\,(g \circ f)\,n \\ = & \langle \text{Def. } \overline{} \rangle & \mathrm{D}_a\,(\overline{g}\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

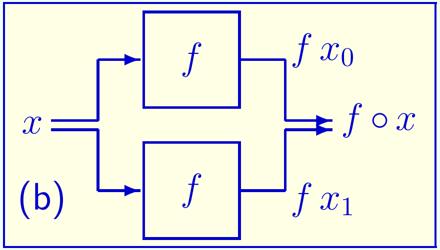
b. Functionals introduced (types omitted; designed during the generification)

- 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

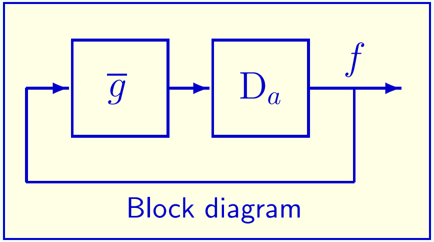
- Structural interpretations of composition: (a) cascading; (b) replication

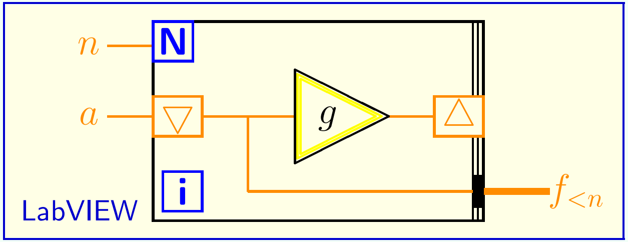




Example property: $\overline{h \circ g} = \overline{h} \circ \overline{g}$ (proof: exercise)

- 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.

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

6 Applications of formal calculation in programming theories

Topic I: Description styles in formal semantics

6.0 Formal semantics: from conventional languages to LabVIEW

Topic II: Calculational derivation of programming theories

6.1 Motivation and principle: a mechanical analogon

6.2 Calculating the “axioms” for assignment from equations

6.3 Generalization to program semantics

6.4 Application to assignment, sequencing, choice and iteration

6.5 Practical rules for iteration

6.0 Formal semantics: from conventional languages to LabVIEW

6.0.0 Expressing abstract syntax

a. For *aggregate constructs* and *list productions*: functional record and *. This is $\times$ actually: record $F = \times(\biguplus F)$ and $A^* = \bigcup n : \mathbb{N} \,.\, \times(\square\, n \,^\bullet\, A)$.

For *choice productions* needing disjoint union: generic elastic $|$-operator

For any family F of types,

$$\boxed{\left| F = \bigcup x : \mathcal{D}\, F \,.\, \{x \mapsto y \mid y : F\, x\}\right.} \tag{63}$$

Idea: analogy with $\bigcup F = \bigcup (x : \mathcal{D}\, F \,.\, F\, x) = \bigcup x : \mathcal{D}\, F \,.\, \{y \mid y : F\, x\}$.

Remarks

- Variadic: $\boxed{A \mid B = \left|(A, B) = \{0 \mapsto a \mid a : A\} \cup \{1 \mapsto b \mid b : B\}\right.}$
- 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".

b. Typical examples:

- Abstract syntax of programs. Eith field labels from an enumeration type,

$$\boxed{\begin{array}{l}\mathbf{def}\ Program := \textsf{record}\,(declarations \mapsto Dlist, body \mapsto Instruction)\\ \mathbf{def}\ Dlist := D^*\\ \mathbf{def}\ D := \textsf{record}\,(v \mapsto Variable, t \mapsto Type)\\ \mathbf{def}\ Instruction := Skip \cup Assignment \cup Compound \cup \textsf{ etc.}\end{array}}$$

  A few items are left undefined here (easily inferred).

- If disjoint union wanted:

$$\boxed{Skip \mid Assignment \mid Compound \mid \textsf{etc.}}$$

- Instances of programs, declarations, etc. can be defined as

$$\boxed{\mathbf{def}\ p : Program\ \mathbf{with}\ p = declarations \mapsto dl \mathbin{\dot\cup} body \mapsto instr}$$

6.0.1 Static semantics example

Validity of declaration lists (no double declarations) and the variable inventory

$$\boxed{\begin{array}{l}\textbf{def}\ Vdcl : Dlist \to \mathbb{B}\ \textbf{with}\ Vdcl\, dl = \mathsf{inj}\,(dl^T v)\\ \textbf{def}\ Var : Dlist \to \mathcal{P}\ Variable\ \textbf{with}\ Var\, dl = \mathcal{R}\,(dl^T\, v)\end{array}}$$

The *type map* of a valid declaration list (mapping variables to their types) is

$$\boxed{\begin{array}{l}\textbf{def}\ typmap : Dlist_{Vdcl} \ni dl \to Var\, dl \to Tval\ \ \textbf{with}\\ \qquad typmap\, dl = tval \circ (dl^T\, t) \circ (dl^T\, v)^-\end{array}}$$

Equivalently, $\boxed{typmap\, dl = \bigcup\!\!\!\cdot\ d : \mathcal{R}\, dl\, .\, d\, v \mapsto tval\,(d\, t)}$.

A type map can be used as a context parameter for expressing validity of expressions and instructions, shown next.

Semantics (continuation) How function *merge* ($\mathbin{\cup\!\!\!\cdot}$) obviates case expressions

Example: type (Texp) and type correctness (Vexp) of expressions. Assume

$$\begin{array}{l}
\textbf{def}\ \mathit{Expr} := \mathit{Constant} \cup \mathit{Variable} \cup \mathit{Applic} \\
\textbf{def}\ \mathit{Constant} := \mathit{IntCons} \cup \mathit{BoolCons} \\
\textbf{def}\ \mathit{Applic} := \mathsf{record}\,(\mathit{op} \mapsto \mathit{Oper}, \mathit{term} \mapsto \mathit{Expr}, \mathit{term}' \mapsto \mathit{Expr})
\end{array}$$

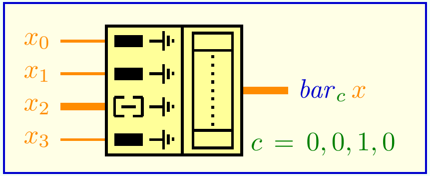
Letting $\mathit{Tmap} := \bigcup\, \mathit{dl} : \mathit{Dlist}_{V\,dcl}\,.\,\mathit{typmap}\,\mathit{dl}$ and $\mathit{Tval} := \{\mathit{it}, \mathit{bt}, \mathit{ut}\}$, define

$$\begin{array}{l}
\textbf{def}\ \mathit{Texp} : \mathit{Tmap} \to \mathit{Expr} \to \mathit{Tval}\ \textbf{with} \\
\qquad \begin{array}{rcl}
\mathit{Texp}\,\mathit{tm} & = & (c : \mathit{IntCons}\,.\,\mathit{it}) \mathbin{\cup\!\!\!\cdot} (c : \mathit{BoolCons}\,.\,\mathit{bt}) \\
 & \mathbin{\cup\!\!\!\cdot} & (v : \mathit{Variable}\,.\,\mathit{ut}) \lessdot \mathit{tm} \\
 & \mathbin{\cup\!\!\!\cdot} & (a : \mathit{Applic}\,.\,(a\,\mathit{op} \in \mathit{Arith_op})\,?\,\mathit{it} \nmid \mathit{bt})
\end{array}
\end{array}$$

$$\begin{array}{l}
\textbf{def}\ \mathit{Vexp} : \mathit{Tmap} \to \mathit{Expr} \to \mathbb{B}\ \textbf{with} \\
\qquad \begin{array}{rcl}
\mathit{Vexp}\,\mathit{tm} & = & (c : \mathit{Constant}\,.\,1) \mathbin{\cup\!\!\!\cdot} (v : \mathit{Variable}\,.\,v \in \mathcal{D}\,\mathit{tm}) \\
 & \mathbin{\cup\!\!\!\cdot} & (a : \mathit{Applic}\,.\,\mathit{Vexp}\,\mathit{tm}\,(a\,\mathit{term}) \wedge \mathit{Vexp}\,\mathit{tm}\,(a\,\mathit{term}') \wedge \\
 & & \mathit{Texp}\,\mathit{tm}\,(a\,\mathit{term}) = \mathit{Texp}\,\mathit{tm}\,(a\,\mathit{term}') \\
 & & = (a\,\mathit{op} \in \mathit{Bool_op})\,?\,\mathit{bt} \nmid \mathit{it}))
\end{array}
\end{array}$$

6.0.2 Semantics of data flow languages

Some time ago done for Silage (textual), now for 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: $\boxed{bar_c\, x = +\!\!+\, i : \mathcal{D}\,c\,.\,(\tau\,(x\,i), x\,i)\,(c\,i)}$. Point-free form:

$$\boxed{\begin{array}{l}\textbf{def}\ bar\_ : \mathbb{B}^+ \ni c \to \bigotimes V : \mathcal{T}\,.\,\times((V, V^*) \circ c) \to V^*\ \textbf{with}\\ \qquad bar_c = +\!\!+ \circ \|\,((\tau, id) \circ c).\end{array}}$$

Topic II: Calculational derivation of programming theories

6.1 Motivation and principle: a mechanical analogon

6.1.0 General

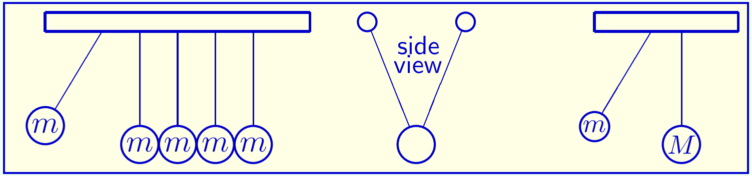
- Educational: axiomatic semantics (Hoare, Dijkstra) nonintuitive, "opaque"
- Research: further unification of mathematical methods
  for continuous and discrete systems (ongoing work)

6.1.1 Specific

- Justification of axiomatic semantics usually detours via denotational semantics
  e.g. Mike Gordon, Bertrand Meyer, Glynn Winskel
- Confusing terminology: what looks like *propositions* is often called *predicates*
- Correct semantics for assignment seems "backwards" (as observed by Gordon)
  Certain "forward" semantics is also correct (reinforces the "mystery"),
  e.g., $\{v = d\}\ v := e\ \{v = e[^v_d]\}$ provided $v \notin \varphi\, d$

6.1.2 Principle: a mechanical analogon

An analogy: colliding balls ("Newton's Cradle")



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

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

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

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

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

6.2 Calculating the "axioms" for assignment from equations

6.2.0 Principle

a. Basic ideas

- If pre- and postcondition *look* like propositions, *treat* them as such
- *Derive* axiomatic semantics from basic *program equations*. Convention: for program variable v, new vars: $`v$ before command, v' after command.
  - Antecondition a becomes $a[^v_{`v}$ and postcondition p becomes $p[^v_{v'}$
  - Substitution and change of variables are familiar in engineering math
  - Consider axiomatic semantics just as "economy in variable use"
- Advantages of the approach:
  - Expressivity: direct formalization of intuitive program behaviour
  - Calculationally: all becomes predicate calculus (no "special" logics)

b. Convention

- Mnemonic symbols, even for bound variables (as in physics, applied math)
- We prefer "ante" over "pre" (better preposition, leads to distinct letters)

c. Expressing Floyd-Hoare semantics in terms of a program equation

- Side issue: assume v to be of type V (as specified by the declarations)
- Intuitive understanding of behaviour of assignment: equation $\boxed{v' = e[^{v}_{`v}}$
- Use in formalizing intuitive understanding of Floyd-Hoare semantics:
  - About $`v$ and $v'$ we know $a[^{v}_{`v}$ and $v' = e[^{v}_{`v}$ (no less, no more).
  - Hence any assertion about v' must be implied by it, in particular $p[^{v}_{v'}$.
    Formally: $a[^{v}_{`v} \wedge v' = e[^{v}_{`v} \Rightarrow p[^{v}_{v'}$ (implicitly quantified over $`v$ and v').

$$\boxed{\{a\}\ v := e\ \{p\} \quad \equiv \quad \forall\, `v : V \,.\, \forall\, v' : V \,.\, a[^{v}_{`v} \wedge v' = e[^{v}_{`v} \Rightarrow p[^{v}_{v'} \qquad (64)}$$

No detour via denotational semantics; assertions remain propositions.

- Example: assume $\mathtt{x}$ declared as integer. Then, by the preceding definition,

$$\boxed{\begin{array}{c} \{\mathtt{x} > 27\}\ \mathtt{x\ \ :=\ \ x+3}\ \{\mathtt{x} > 30\} \\ \equiv \\ \forall\, `\mathtt{x} : \mathbb{Z} \,.\, \forall\, \mathtt{x}' : \mathbb{Z} \,.\, `\mathtt{x} > 27 \wedge \mathtt{x}' = `\mathtt{x} + 3 \Rightarrow \mathtt{x}' > 30 \end{array}}$$

The latter expression evaluates to 1 (or T if one prefers) by calculation.

6.2.1 Calculating the weakest antecondition

- Calculation (assuming type correctness checked, viz., $\forall \lq v : V \,.\, e[^{v}_{\lq v} \in V$)

$$
\begin{array}{lcll}
\{a\}\ v := e\ \{p\} & \equiv & \langle \text{Definit. (64)} \rangle & \forall \lq v : V \,.\, \forall v' : V \,.\, a[^{v}_{\lq v} \wedge v' = e[^{v}_{\lq v} \Rightarrow p[^{v}_{v'} \\
 & \equiv & \langle \text{Shunting} \rangle & \forall \lq v : V \,.\, \forall v' : V \,.\, a[^{v}_{\lq v} \Rightarrow v' = e[^{v}_{\lq v} \Rightarrow p[^{v}_{v'} \\
 & \equiv & \langle \text{Ldist} \Rightarrow / \forall \rangle & \forall \lq v : V \,.\, a[^{v}_{\lq v} \Rightarrow \forall v' : V \,.\, v' = e[^{v}_{\lq v} \Rightarrow p[^{v}_{v'} \\
 & \equiv & \langle \text{One-pt. rule} \rangle & \forall \lq v : V \,.\, a[^{v}_{\lq v} \Rightarrow e[^{v}_{\lq v} \in V \Rightarrow p[^{v}_{e[^{v}_{\lq v}} \\
 & \equiv & \langle \text{Assumption} \rangle & \forall \lq v : V \,.\, a[^{v}_{\lq v} \Rightarrow p[^{v}_{e[^{v}_{\lq v}} \\
 & \equiv & \langle \text{Change vars.} \rangle & \forall v : V \,.\, a \Rightarrow p[^{v}_{e}
\end{array}
$$

- This proves the $\boxed{\text{Theorem: } \{a\}\ v := e\ \{p\} \equiv \forall v : V \,.\, a \Rightarrow p[^{v}_{e}}$. Hence
  - $p[^{v}_{e}$ is at most as strong as any antecondition a.
  - $p[^{v}_{e}$ is itself an antecondition since $\{p[^{v}_{e}\}\ v := e\ \{p\} \equiv \forall v : V \,.\, p[^{v}_{e} \Rightarrow p[^{v}_{e}$

- Therefore we define $\boxed{\text{wa}\ [\![v := e]\!]\ p \ \equiv\ p[^{v}_{e} \qquad (65)}$

6.2.2 Calculating the strongest postcondition

- Calculation

$$
\boxed{
\begin{array}{rcll}
\{a\}\ v := e\ \{p\} & \equiv & \langle\text{Definit. (64)}\rangle & \forall\,\text{`}v : V\,.\,\forall\,v' : V\,.\,a[^{v}_{\text{`}v} \wedge v' = e[^{v}_{\text{`}v} \Rightarrow p[^{v}_{v'} \\
& \equiv & \langle\text{Swap } \forall/\forall\rangle & \forall\,v' : V\,.\,\forall\,\text{`}v : V\,.\,a[^{v}_{\text{`}v} \wedge v' = e[^{v}_{\text{`}v} \Rightarrow p[^{v}_{v'} \\
& \equiv & \langle\text{Rdist } \Rightarrow/\forall\rangle & \forall\,v' : V\,.\,\exists\,(\text{`}v : V\,.\,a[^{v}_{\text{`}v} \wedge v' = e[^{v}_{\text{`}v}) \Rightarrow p[^{v}_{v'} \\
& \equiv & \langle\text{Change var}\rangle & \forall\,v : V\,.\,\exists\,(\text{`}v : V\,.\,a[^{v}_{\text{`}v} \wedge v = e[^{v}_{\text{`}v}) \Rightarrow p
\end{array}
}
$$

- Hence $\boxed{\text{Theorem: } \{a\}\ v := e\ \{p\} \equiv \forall\,v : V\,.\,\exists\,(u : V\,.\,a[^{v}_{u} \wedge v = e^{v}_{u}) \Rightarrow p}$, so
  - $\exists\,(u : V\,.\,a[^{v}_{u} \wedge v = e^{v}_{u})$ is at least as strong as any postcondition p.
  - $\exists\,(u : V\,.\,a[^{v}_{u} \wedge v = e^{v}_{u})$ is itself a postcondition.
- Therefore we define $\boxed{\mathsf{sp}\,[\![v := e]\!]\,a \ \equiv\ \exists\,(u : V\,.\,a[^{v}_{u} \wedge v = e^{v}_{u}) \qquad (66)}$

6.2.3 A few interesting excursions / illustrations

a. Justifying the "forward" rule $\boxed{\{v = d\}\; v := e\; \{v = e[^v_d\} \quad \text{provided} \quad v \notin \varphi\, d}$

$$
\boxed{
\begin{array}{lll}
\multicolumn{3}{l}{\{v = d\}\; v := e\; \{v = e[^v_d\}} \\
\equiv & \langle \text{Definit. (64)} \rangle & \forall\, \grave{}v : V \,.\, \forall\, v' : V \,.\, \grave{}v = d[^v_{\grave{}v} \wedge v' = e[^v_{\grave{}v} \Rightarrow v' = e[^v_{d[^v_{v'}}} \\
\equiv & \langle v \notin \varphi\, d \rangle & \forall\, \grave{}v : V \,.\, \forall\, v' : V \,.\, \grave{}v = d \wedge v' = e[^v_{\grave{}v} \Rightarrow v' = e[^v_d \\
\equiv & \langle \text{Leibniz, bis} \rangle & \forall\, \grave{}v : V \,.\, \forall\, v' : V \,.\, \grave{}v = d \wedge v' = e[^v_{\grave{}v} \Rightarrow e[^v_{\grave{}v} = e[^v_{\grave{}v} \\
\equiv & \langle \text{Reflex.} = \rangle & 1 \qquad \text{(or } \mathrm{T} \text{ if one prefers)}
\end{array}
}
$$

b. Bouncing ante- and postconditions Letting $c := [\![v := e]\!]$, calculation yields

$$
\boxed{\mathsf{sp}\, c\, (\mathsf{wa}\, c\, p) \equiv p \wedge \exists\, u : V \,.\, v = e[^v_u \qquad \mathsf{wa}\, c\, (\mathsf{sp}\, c\, a) \equiv \exists\, u : V \,.\, a[^v_u \wedge e = e^v_u}
$$

E.g., assume the declaration $\mathtt{y} : \mathtt{integer}$ and let $c :=$ '$\mathtt{y} := \mathtt{y}^2 + 7$';
furthermore, let $p :=$ '$\mathtt{y} > 11$' and $q :=$ '$\mathtt{y} < 7$' and $a :=$ '$\mathtt{y} > 2$'

- $\mathsf{sp}\, c\, (\mathsf{wa}\, c\, p) \equiv \mathtt{y} > 11 \wedge \exists\, x : \mathbb{Z} \,.\, \mathtt{y} = x^2 + 7$ (stronger than p !)
- $\mathsf{sp}\, c\, (\mathsf{wa}\, c\, q) \equiv \mathtt{y} < 7 \wedge \exists\, x : \mathbb{Z} \,.\, \mathtt{y} = x^2 + 7$ (simplifies to 0 or F)
- $\mathsf{wa}\, c\, (\mathsf{sp}\, c\, a) \equiv \exists\, x : \mathbb{Z} \,.\, x > 2 \wedge \mathtt{y}^2 + 7 = x^2 + 7$ (yields $\mathtt{y} > 2 \vee \mathtt{y} < -2$)

$$\boxed{\begin{array}{rcll} \text{Recall: } \mathsf{sp}\,c\,(\mathsf{wa}\,c\,p) & \equiv & p \wedge \exists\, u : V \,.\, v = e[^v_u & \text{(a)} \\ \mathsf{wa}\,c\,(\mathsf{sp}\,c\,a) & \equiv & \exists\, u : V \,.\, a[^v_u \wedge e = e^v_u & \text{(b)} \end{array}} \qquad (67)$$

$$\boxed{\begin{array}{rccl}
\mathsf{sp}\,c\,(\mathsf{wa}\,c\,q) & \equiv & \langle \text{Bounce (67a)} \rangle & q \wedge \exists\, \grave{v} : V \,.\, v = e[^v_{\grave{v}} \\
 & \equiv & \langle \text{Def. } q,\, V,\, e \rangle & \mathrm{y} < 7 \wedge \exists\, x : \mathbb{Z} \,.\, \mathrm{y} = x^2 + 7 \\
 & \equiv & \langle \text{Dist. } \wedge/\exists \rangle & \exists\, x : \mathbb{Z} \,.\, \mathrm{y} < 7 \wedge \mathrm{y} = x^2 + 7 \\
 & \equiv & \langle \text{Leibniz} \rangle & \exists\, x : \mathbb{Z} \,.\, x^2 + 7 < 7 \wedge \mathrm{y} = x^2 + 7 \\
 & \equiv & \langle \text{Arithmetic} \rangle & \exists\, x : \mathbb{Z} \,.\, x^2 < 0 \wedge \mathrm{y} = x^2 + 7 \\
 & \equiv & \langle \forall\, x : \mathbb{Z} \,.\, x^2 \geq 0 \rangle & \exists\, x : \mathbb{Z} \,.\, 0 \wedge \mathrm{y} = x^2 + 7 \\
 & \equiv & \langle 0 \wedge p \equiv 0 \rangle & \exists\, x : \mathbb{Z} \,.\, 0 \\
 & \equiv & \langle \exists\, (S^{\bullet}\, 0) \equiv 0 \rangle & 0 \ \text{(strongest of all propositions)} \\
\mathsf{wa}\,c\,(\mathsf{sp}\,c\,a) & \equiv & \langle \text{Bounce (67b)} \rangle & \exists\, \grave{v} : V \,.\, a[^v_{\grave{v}} \wedge e = e^v_{\grave{v}} \\
 & \equiv & \langle \text{Def. } a,\, V,\, e \rangle & \exists\, x : \mathbb{Z} \,.\, x > 2 \wedge \mathrm{y}^2 + 7 = x^2 + 7 \\
 & \equiv & \langle \text{Arithmetic} \rangle & \exists\, x : \mathbb{Z} \,.\, x > 2 \wedge (x = \mathrm{y} \vee x = -\mathrm{y}) \\
 & \equiv & \langle \text{Distr. } \wedge/\vee \rangle & \exists\, x : \mathbb{Z} \,.\, (x = \mathrm{y} \wedge x > 2) \vee (x = -\mathrm{y} \wedge x > 2) \\
 & \equiv & \langle \text{Distr. } \exists/\vee \rangle & \exists\, (x : \mathbb{Z} \,.\, x = \mathrm{y} \wedge x > 2) \vee \exists\, (x : \mathbb{Z} \,.\, x = -\mathrm{y} \wedge x > 2) \\
 & \equiv & \langle \text{One-pt. rule} \rangle & (\mathrm{y} \in \mathbb{Z} \wedge \mathrm{y} > 2) \vee (-\mathrm{y} \in \mathbb{Z} \wedge -\mathrm{y} > 2) \\
 & \equiv & \langle \text{Arithmetic} \rangle & (\mathrm{y} \in \mathbb{Z} \wedge \mathrm{y} > 2) \vee (\mathrm{y} \in \mathbb{Z} \wedge \mathrm{y} < -2) \\
 & \equiv & \langle \text{Distr. } \exists/\vee \rangle & \mathrm{y} \in \mathbb{Z} \wedge (\mathrm{y} > 2 \vee \mathrm{y} < -2) \ \text{(compare with } a\text{)}
\end{array}}$$

6.3 Generalization to program semantics

6.3.0 Conventions

a. Preliminary remark   A familiar "problem": Given a variable x, we have

$x \in \mathit{Variable}$ (at metalevel) and, for instance, $x \in \mathbb{Z}$ (in the language).

Not resolvable without considerable nomenclature, yet clear from the context.
Conclusion: let us exploit it rather than lose time over it (at this stage).
Remark: similar (more urgent) problem in calculus: how to "save Leibniz" by formalizing if $y = x^2$, then $\mathrm{d}\, y = 2 \cdot x \cdot \mathrm{d}\, x$ (partial solution 11 years ago)

b. State space

- Not the denotational semantics view where state $s : \mathit{Variable} \to \mathit{Value}$
- State space $\mathbf{S}$ is Cartesian product determined by variable declarations. Henceforth, s is shorthand for the tuple of all program variables. Auxiliary variables ("ghost" or "rigid" variables) appended if necessary.
- Example: `var x : int, b : bool` yields $\mathbf{S} = \mathbb{Z} \times \mathbb{B}$ and $s = \mathtt{x}, \mathtt{b}$.
- In what follows, v is a tuple of variables, type S_v, in particular $\mathbf{S} = S_s$.

6.3.1 Expressing Floyd-Hoare semantics in terms of program equations

a. Program equations formalizing the intuitive behaviour of command c

- $R\,c\,(\grave{}s, s')$ expressing the state change (suitable $R : C \to \mathbf{S}^2 \to \mathbb{B}$)
- $T\,c\,s$ expressing termination of c started in state s (suitable $T : C \to \mathbf{S} \to \mathbb{B}$)
  In further treatment: only guaranteed, not just possible termination
  E.g., *not* as in the example in Gordon's *Specification and Verification 1*, which would amount (with our conventions) to $T\,c\,s \equiv \exists\, s' : \mathbf{S} \,.\, R\,c\,(s, s')$.

b. Formalizing intuitive Floyd-Hoare semantics for weak correctness

- About $\grave{}s$ and s' we know $a[^s_{\grave{}s}$ and $R\,c\,(\grave{}s, s')$, no less, no more.
- Therefore: $a[^s_{\grave{}s} \wedge R\,c\,(\grave{}s, s') \Rightarrow p[^s_{s'}$ (implicitly quantified over $\grave{}s$ and s').
- Hence $\boxed{\{a\}\; c\; \{p\} \;\equiv\; \forall\, \grave{}s : \mathbf{S} \,.\, \forall\, s' : \mathbf{S} \,.\, a[^s_{\grave{}s} \wedge R\,c\,(\grave{}s, s') \Rightarrow p[^s_{s'} \qquad (68)}$

c. Strong correctness: defining Term by $\boxed{\mathit{Term}\,c\,a \equiv \forall\, s : \mathbf{S} \,.\, a \Rightarrow T\,c\,s \qquad (69)}$

$$\begin{array}{lcll} [a]\; c\; [p] & \equiv & \{a\}\; c\; \{p\} \wedge \mathit{Term}\,c\,a & \text{or, blending in (68):} \\ [a]\; c\; [p] & \equiv & \forall\, \grave{}s : \mathbf{S} \,.\, \forall\, s' : \mathbf{S} \,.\, a[^s_{\grave{}s} \Rightarrow T\,c\,\grave{}s \wedge (R\,c\,(\grave{}s, s') \Rightarrow p[^s_{s'}) & (70) \end{array}$$

6.3.2 Calculating the weakest antecondition

- Calculation

$$\begin{array}{lll}
[a]\ c\ [p] & & \\
\equiv & \langle\text{Definit. (70)}\rangle & \forall\,`s : \mathbf{S} \,.\, \forall\, s' : \mathbf{S} \,.\, a[^s_{`s} \Rightarrow T\,c\,`s \wedge (R\,c\,(`s, s') \Rightarrow p[^s_{s'}) \\
\equiv & \langle\text{Ldist} \Rightarrow/\forall\rangle & \forall\,`s : \mathbf{S} \,.\, a[^s_{`s} \Rightarrow \forall\, s' : \mathbf{S} \,.\, T\,c\,`s \wedge (R\,c\,(`s, s') \Rightarrow p[^s_{s'}) \\
\equiv & \langle\text{Pdist } \forall/\wedge\rangle & \forall\,`s : \mathbf{S} \,.\, a[^s_{`s} \Rightarrow T\,c\,`s \wedge \forall\, s' : \mathbf{S} \,.\, R\,c\,(`s, s') \Rightarrow p[^s_{s'} \\
\equiv & \langle\text{Change vars.}\rangle & \forall\, s : \mathbf{S} \,.\, a \Rightarrow T\,c\,s \wedge \forall\, s' : \mathbf{S} \,.\, R\,c\,(s, s') \Rightarrow p[^s_{s'}
\end{array}$$

- So we proved $\boxed{[a]\ c\ [p] \ \equiv\ \forall\, s : \mathbf{S} \,.\, a \Rightarrow T\,c\,s \wedge \forall\, s' : \mathbf{S} \,.\, R\,c\,(s, s') \Rightarrow p[^s_{s'}}$
  - Observe, as before, that $T\,c\,s \wedge \forall\, s' : \mathbf{S} \,.\, R\,c\,(s, s') \Rightarrow p[^s_{s'}$ is at most as strong as any antecondition a and is itself an antecondition
  - Hence $\boxed{\text{wa}\, c\, p \ \equiv\ T\,c\,s \wedge \forall\, s' : \mathbf{S} \,.\, R\,c\,(s, s') \Rightarrow p[^s_{s'} \qquad (71)}$
- Liberal variant: $\boxed{\text{wla}\, c\, p \ \equiv\ \forall\, s' : \mathbf{S} \,.\, R\,c\,(s, s') \Rightarrow p[^s_{s'} \qquad (72)}$
  (shortcut: obtained by substituting $T\,c\,s \equiv 1$)

6.3.3 Calculating the strongest postcondition

- Calculation

$$
\boxed{
\begin{array}{lcl}
[a]\ c\ [p] & & \\
\equiv & \langle \text{Def. (68–70)} \rangle & Term\, c\, a \wedge \forall\, {\backprime}s : \mathbf{S} \,.\, \forall\, s' : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s') \Rightarrow p[^{s}_{s'} \\
\equiv & \langle \text{Swap } \forall \rangle & Term\, c\, a \wedge \forall\, s' : \mathbf{S} \,.\, \forall\, {\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s') \Rightarrow p[^{s}_{s'} \\
\equiv & \langle \text{Rdist } \Rightarrow/\forall \rangle & Term\, c\, a \wedge \forall\, s' : \mathbf{S} \,.\, \exists\, ({\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s')) \Rightarrow p[^{s}_{s'} \\
\equiv & \langle \text{Change var} \rangle & Term\, c\, a \wedge \forall\, s : \mathbf{S} \,.\, \exists\, ({\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s)) \Rightarrow p
\end{array}
}
$$

- So we proved $\boxed{[a]\ c\ [p] \ \equiv\ Term\, c\, a \wedge \forall\, s : \mathbf{S} \,.\, \exists\, ({\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s)) \Rightarrow p}$
  - Assuming $Term\, c\, a$, observe, as before, that $\exists\, {\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s)$ is at least as strong as any postcondition p and is itself a postcondition
  - Hence $\boxed{\mathsf{sp}\, c\, p \ \equiv\ \exists\, {\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s) \text{ provided } Term\, c\, a \qquad (73)}$
- Liberal variant: $\boxed{\mathsf{slp}\, c\, p \ \equiv\ \exists\, {\backprime}s : \mathbf{S} \,.\, a[^{s}_{{\backprime}s} \wedge R\, c\, ({\backprime}s, s) \qquad (74)}$

6.4 Application to assignment, sequencing, choice and iteration

6.4.0 Assignment revisited (embedded in the general case)

a. We consider (possibly) multiple assignment   Let $c := [\![v := e]\!]$

- Here v may be a tuple of variables and e a matching tuple of expressions.
- Convenient in calculations: (W.L.O.G.) $s = v +\!\!+\, w$ (w rest of variables); similarly $`s = `v +\!\!+\, `w$ and $s' = v' +\!\!+\, w'$

b. Formalizing intuitive understanding (note simplest choice of bound variables)

$$\boxed{\begin{array}{rcll} R\,c\,(s,s') & \equiv & s' = s[^v_e & \quad (75) \\ T\,c\,s & \equiv & 1 & \end{array}}$$

E.g., $R\,[\![\mathtt{y,j\ :=\ y+j,j+1}]\!]\,((\mathtt{y},\mathtt{j},\mathtt{k}),(\mathtt{y}',\mathtt{j}',\mathtt{k}')) \equiv \mathtt{y}',\mathtt{j}',\mathtt{k}' = \mathtt{y}+\mathtt{j},\mathtt{j}+1,\mathtt{k}$

c. Weakest ante- and strongest postconditions   From (71) and (73) with (75),

$$\boxed{\begin{array}{rcll} \mathsf{wa}\,[\![v := e]\!]\,p & \equiv & p[^v_e & \quad (76) \\ \mathsf{sp}\,[\![v := e]\!]\,a & \equiv & \exists\,`v : S_v\,.\,a[^v_{`v} \wedge v = e[^v_{`v} & \quad (77) \end{array}}$$

6.4.1 Sequencing

a. Formalization of intuitive understanding of behaviour

$$\boxed{\begin{array}{rcl} R\,[\![c' ; c'']\!]\,(\grave{}s, s') & \equiv & \exists\, t : \mathbf{S}\,.\, R\,c'\,(\grave{}s, t) \wedge R\,c''\,(t, s') \\ T\,[\![c' ; c'']\!]\,s & \equiv & T\,c'\,s \wedge \forall\, t : \mathbf{S}\,.\, R\,c'\,(s, t) \Rightarrow T\,c''\,t \qquad (78) \end{array}}$$

b. Weakest antecondition (strongest postcondition similar) Let $c := [\![c' ; c'']\!]$ in

$$\begin{array}{lll} & \mathsf{wa}\,c\,p & \\ \equiv & \langle \text{Eqn. wa (71)} \rangle & T\,c\,s \wedge \forall\, s' : \mathbf{S}\,.\, R\,c\,(s, s') \Rightarrow p[^s_{s'} \\ \equiv & \langle \text{Def. } R \text{ (78)} \rangle & T\,c\,s \wedge \forall\, s' : \mathbf{S}\,.\, \exists\,(t : \mathbf{S}\,.\, R\,c'\,(s, t) \wedge R\,c''\,(t, s')) \Rightarrow p[^s_{s'} \\ \equiv & \langle \text{Rdist. } \Rightarrow/\forall \rangle & T\,c\,s \wedge \forall\, s' : \mathbf{S}\,.\, \forall\, t : \mathbf{S}\,.\, R\,c'\,(s, t) \wedge R\,c''\,(t, s') \Rightarrow p[^s_{s'} \\ \equiv & \langle \text{Rearrange} \rangle & T\,c\,s \wedge \forall\, t : \mathbf{S}\,.\, R\,c'\,(s, t) \Rightarrow \forall\, s' : \mathbf{S}\,.\, R\,c''\,(t, s') \Rightarrow p[^s_{s'} \\ \equiv & \langle \text{Blend } T \text{ (78)} \rangle & T\,c'\,s \wedge \forall\, t : \mathbf{S}\,.\, R\,c'\,(s, t) \Rightarrow T\,c''\,t \wedge \forall\, s' : \mathbf{S}\,.\, R\,c''\,(t, s') \Rightarrow p[^s_{s'} \\ \equiv & \langle \text{Eqn. wa (71)} \rangle & T\,c'\,s \wedge \forall\, t : \mathbf{S}\,.\, R\,c'\,(s, t) \Rightarrow (\mathsf{wa}\,c''\,p)[^s_t \\ \equiv & \langle \text{Eqn. wa (71)} \rangle & \mathsf{wa}\,c'\,(\mathsf{wa}\,c''\,p) \end{array}$$

Remark: Gordon observes that this could not be obtained by $T\,c\,s \equiv \exists\, t : \mathbf{S}\,.\, R\,c\,(s, t)$

6.4.2 Choice (nondeterministic; deterministic as particular case)

a. Formalizing intuitive understanding Let $ch := [\![\texttt{if}\ \ [\!]\ i : I \,.\, b_i\ \texttt{->}\ c_i\ \ \texttt{fi}]\!]$ in

$$\boxed{\begin{array}{rcl} R\,ch\,(s, s') & \equiv & \exists\, i : I \,.\, b_i \wedge R\,c_i\,(s, s') \\ T\,ch\,s & \equiv & \forall\, i : I \,.\, b_i \Rightarrow T\,c_i\,s \end{array} \qquad (79)}$$

Remark: I is just a (finite) indexing set, say, $0 \mathinner{..} n - 1$ for n alternatives.

b. Weakest ante-, strongest postcondition Let $ch := [\![\texttt{if}\ \ [\!]\ i : I \,.\, b_i\ \texttt{->}\ c_i\ \ \texttt{fi}]\!]$ in

$$\boxed{\begin{array}{rcll} \mathsf{wa}\,ch\,p & \equiv & \langle \text{Eqn. wa (71)} \rangle & T\,ch\,s \wedge \forall\, s' : \mathbf{S} \,.\, R\,ch\,(s, s') \Rightarrow p[^s_{s'} \\ & \equiv & \langle \text{Def. } R \text{ (79)} \rangle & T\,ch\,s \wedge \forall\, s' : \mathbf{S} \,.\, \exists\,(i : I \,.\, b_i \wedge R\,c_i\,(s, s')) \Rightarrow p[^s_{s'} \\ & \equiv & \langle \text{Rdist} \Rightarrow / \exists \rangle & T\,ch\,s \wedge \forall\, s' : \mathbf{S} \,.\, \forall\, i : I \,.\, b_i \wedge R\,c_i\,(s, s') \Rightarrow p[^s_{s'} \\ & \equiv & \langle \text{Shunt, dist.} \rangle & T\,ch\,s \wedge \forall\, i : I \,.\, b_i \Rightarrow \forall\, s' : \mathbf{S} \,.\, R\,c_i\,(s, s') \Rightarrow p[^s_{s'} \\ & \equiv & \langle \text{Blend } T \text{ (79)} \rangle & \forall\, i : I \,.\, b_i \Rightarrow T\,c_i\,s \wedge \forall\, s' : \mathbf{S} \,.\, R\,c_i\,(s, s') \Rightarrow p[^s_{s'} \\ & \equiv & \langle \text{Eqn. wa (71)} \rangle & \forall\, i : I \,.\, b_i \Rightarrow \mathsf{wa}\,c_i\,p \\ \mathsf{sp}\,ch\,a & \equiv & \langle \text{Similar calc.} \rangle & \exists\, i : I \,.\, \mathsf{sp}\,c_i\,(a \wedge b_i), \text{ provided } \mathit{Term}\,c\,a \end{array}}$$

c. Particular case: defining $[\![\texttt{if}\ b\ \texttt{then}\ c'\ \texttt{else}\ c''\ \texttt{fi}]\!] = [\![\texttt{if}\ b\ \texttt{->}\ c'\ [\!]\ \neg\, b\ \texttt{->}\ c''\ \texttt{fi}]\!]$

yields $\boxed{\mathsf{wa}\,[\![\texttt{if}\ b\ \texttt{then}\ c'\ \texttt{else}\ c''\ \texttt{fi}]\!]\,p \equiv (b \Rightarrow \mathsf{wa}\,c'\,p) \wedge (\neg\, b \Rightarrow \mathsf{wa}\,c''\,p)}$

6.4.3 Iteration

a. Formalizing intuitive understanding Let $l := [\![\texttt{do}\ b\ \texttt{->}\ c\ \texttt{od}]\!]$ in what follows.

Then $\boxed{l\ =\ [\![\texttt{if}\ \neg\, b\ \texttt{->}\ \texttt{skip}\ [\!]\ b\ \texttt{->}\ c\,;\ l\ \texttt{fi}]\!]}$ formalizes intuition about behaviour.

b. Calculating $R\,l$, $T\,l$ and wa l Using the earlier results, (head) calculation yields:

$$\boxed{\begin{array}{rcll} R\,l\,(s,s') & \equiv & (\neg\, b \Rightarrow s = s') \wedge (b \Rightarrow \exists\, t : \mathbf{S} \,.\, R\,c\,(s,t) \wedge R\,l\,(t,s')) & (80) \\ T\,l\,s & \equiv & (\neg\, b \Rightarrow 1) \wedge (b \Rightarrow T\,c\,s \wedge \forall\, t : \mathbf{S} \,.\, R\,c\,(s,t) \Rightarrow T\,l\,t) & (81) \\ \mathsf{wa}\,l\,p & \equiv & (\neg\, b \Rightarrow p) \wedge (b \Rightarrow \mathsf{wa}\,c\,(\mathsf{wa}\,l\,p)) & (82) \end{array}}$$

Equivalently: $\mathsf{wa}\,l\,p \equiv (\neg\, b \wedge p) \vee (b \wedge \mathsf{wa}\,c\,(\mathsf{wa}\,l\,p))$. Unfolding suggests defining

$$\boxed{\begin{array}{rcl} w_{n+1}\,l\,p & \equiv & (\neg\, b \wedge p) \vee (b \wedge \mathsf{wa}\,c\,(w_n\,l\,p)) \\ w_0\,l\,p & \equiv & \neg\, b \wedge p \end{array}}$$

By induction, one can prove $\forall\, n : \mathbb{N} \,.\, w_n\,l\,p \Rightarrow \mathsf{wa}\,l\,p$ so $\boxed{\exists\,(n : \mathbb{N} \,.\, w_n\,l\,p) \Rightarrow \mathsf{wa}\,l\,p}$

c. Bounded nondeterminism: extra requirement $T\,l\,s \Rightarrow \exists\,(n : \mathbb{N} \,.\, d_n\,l\,s)$ where $d_0\,l\,s \equiv \neg\, b$ and $d_{n+1}\,l\,s \equiv b \Rightarrow T\,c\,s \wedge \forall\, t : \mathbf{S} \,.\, R\,c\,(s,t) \Rightarrow d_n\,l\,t$ ($\#$ steps $\leq n$).
Then $\boxed{\mathsf{wa}\,l\,p \equiv \exists\,(n : \mathbb{N} \,.\, w_n\,l\,p)}$ (as in Dijkstra's *A Discipline of Programming*).

6.5 Practical rules for iteration

Let D be a set with order $<$ and $W : \mathcal{P}\, D$ a well-founded subset under $<$. Then an expression e of type D is a *bound expression* for c iff (i) $\forall\, s \bullet b \Rightarrow e \in W$; (ii) $\forall\, w : W \,.\, [b \wedge w = e]\ c'\ [e < w]$. Combining:

> DEFINITION: $i : \mathbb{B}$ and $e : D$ are an *invariant/bound pair* for c iff
> (i) $\forall\, s \bullet i \wedge b \Rightarrow e \in W$ and (ii) $\forall\, w : W \,.\, [i \wedge b \wedge w = e]\ c'\ [i \wedge e < w]$

We can prove the following theorem.

> THEOREM: If i, e is an invariant/bound pair for c then $[i]\ c\ [i \wedge \neg\, b]$ (83)

Using (83) in practice is best done via a *checklist*, as suggested by Gries: to show $[a]$ do b -> c' od $[p]$, find suitable i, e and prove

(i) i satisfies $[i \wedge b]\ c'\ [i]$ or, equivalently, $i \wedge b \Rightarrow \mathsf{wa}\, c'\, i$.
(ii) i satisfies $a \Rightarrow i$.
(iii) i satisfies $i \wedge \neg\, b \Rightarrow p$.
(iv) e satisfies $i \wedge b \Rightarrow e \in W$.
(v) e satisfies $i \wedge b \wedge w = e \Rightarrow \mathsf{wa}\, c'\, (e < w)$ for any $w : W$.

Next topic

13:30–13:40	0. Introduction: purpose and approach
	Lecture A: Mathematical preliminaries and generic functionals
13:40–14:10	1. Preliminaries: formal calculation with equality, propositions, sets
14:10–14:30	2. Functions and introduction to concrete generic functionals
14:30–15:00	Half hour break
	Lecture B: Functional predicate calculus and general applications
15:00–15:30	3. Functional predicate calculus: calculating with quantifiers
15:30–15:55	4. General applications to functions, functionals, relations, induction
15:55–16:05	Ten-minute break
	Lecture C: Applications in computer and software engineering
16:05–16:40	5. Applications of generic functionals in computing science
16:40–17:00	6. Applications of formal calculation in programming theories
(given time)	7. Formal calculation as unification with classical engineering

Note: depending on the definitive program for tutorials, times indicated may shift.

7 Formal calculation as unification with classical engineering

7.0 Applications in analysis: calculation replacing syncopation

7.1 Applications in continuous and general systems theory

7.2 Applications in discrete systems theory

7.3 Closing remarks: a discipline of Electrical and Computer Engineering

7.0 Analysis: calculation replacing syncopation — an example

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

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

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

7.1 Applications in continuous and general systems theory

7.1.0 Transform methods

a. Emphasis: formally correct use of functionals; clear and unambiguous bindings

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

$$\boxed{\mathcal{F}\,f\,\omega \;=\; \int_{-\infty}^{+\infty} e^{-j\cdot\omega\cdot t}\cdot f\,t\cdot \mathrm{d}\,t \qquad\qquad \mathcal{F}'g\,t \;=\; \frac{1}{2\cdot\pi}\cdot\int_{-\infty}^{+\infty} e^{j\cdot\omega\cdot t}\cdot g\,\omega\cdot \mathrm{d}\,\omega}$$

b. Example: formalizing Laplace transforms via Fourier transforms.

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

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

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

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

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

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

c. Calculation example: the inverse Laplace transform

Specification of $\mathcal{L}'$: $\boxed{\mathcal{L}'\,(\mathcal{L}\,f)\,t = f\,t \text{ for all } t \geq 0}$

(weakened where $\ell_\sigma \widehat{\cdot} f$ is discontinous).

Calculation of an explicit expression: For t as specified,

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

7.1.1 Characterization of properties of systems

a. Definitions and conventions

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

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

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

b. Characteristics Let $s : \mathcal{S}_A \rightarrow \mathcal{S}_B$. Then:

- System s is

  memoryless iff $\;\exists\, f_{\_} : \mathbb{T} \rightarrow A \rightarrow B \,.\, \forall\, x : \mathcal{S}_A \,.\, \forall\, t : \mathbb{T} \,.\, s\, x\, t = f_t\, (x\, t)$

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

  time-invariant iff $\;\forall\, \tau : \mathbb{T} \,.\, s \circ \sigma_\tau = \sigma_\tau \circ s$

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

  linear iff $\;\forall\, z : \mathcal{S}_\mathbb{C} \,.\, \forall\, c : \mathbb{C} \,.\, s\, (c \overrightarrow{\cdot}\, z) = c \overrightarrow{\cdot}\, s\, z$

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

c. Response of LTI systems

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

Then we have:

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

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

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

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

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

Note that this proof uses only the essential hypotheses.

7.2 Applications in discrete systems theory

7.2.0 Motivation and chosen topic

Automata: classical common ground between computing and systems theory.
Even here formalization yields unification and new insights.
Topic: sequentiality and the derivation of properties by predicate calculus.

7.2.1 Sequential discrete systems

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

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

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

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

Now we can prove:

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

(proof: next)

Proof: we start from the sequentiality side.

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

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

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

7.2.2 Derivatives and primitives

The preceding framework leads to the following.

a. Observation: An rb function is unique (exercise).

b. We define the *derivation* operator D on sequential systems by

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

With the rb function r of s, $\mathsf{D}\,s\,(x \prec a) = r\,(x, \tau\,a)$.

c. *Primitivation* I is defined for any $g : A^* \to B^*$ by

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

d. Properties (a striking analogy from analysis is shown in the second row)

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

In the second row, D is derivation as in analysis, and $\mathsf{I}\,g\,x = \int_0^x g\,y \cdot \mathrm{d}\,y$.

e. The $\boxed{\textit{state space}}$ is $\{(y : A^* \,.\, r\,(x, y)) \mid x : A^*\}$.

If we replace r by its Curried version, the state space is $\boxed{\{r\,x \mid x : A^*\}}$

7.3 Closing remarks: a discipline of ECE

a. What we have shown

- A formalism with a very simple language and powerful formal rules
- Notational and methodological unification
  of CS and other engineering theories
- Unification also encompassing a large part of mathematics.

b. Ramifications

- Scientific: obvious (deepens insight, consolidates knowledge, reduces fragmentation, expands intuition, suggests solutions, exploits analogies etc.)
- Educational: unified basis for ECE (Electrical and Computer Engineering)
  Possible curriculum structure:
  - Formal calculation at early stage
  - Other engineering math courses rely on it and provide consolidation