

# Tutorial — FM 2008

## Formal Methods and Signal Processing

Raymond Boute, INTEC — Ghent University

Tuesday, 2008-05-27

09:00–09:20	0. Motivation: synergy FM-SP; value of defect-free formalisms
	<b>Part I — Unifying the mathematics of SP and FM</b>
09:20–09:40	1. Review of the basics and outline of the formalism used
09:40–10:10	2. Inspiration from SP to FM: functions and functionals
10:10–10:30	3. Calculation with predicates and quantifiers for engineers
10:30–11:00	(Half-hour break)
	<b>Part II — Application examples to modeling in SP and FM</b>
11:00–11:40	4. Predicate calculus and generic functionals applied to classical SP
11:40–11:45	5. Brief intermezzo about endosemantic functions
11:45–12:05	6. Modeling programs by program equations
12:05–12:25	7. Reasoning about temporal behavior by temporal operators
12:25–12:30	8. Final considerations

## 0 Motivation: synergy FM-SP; defect-free formalisms

### 0.0 Context: bridging the rift between classical engineering and CS

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

(David L. Parnas, "Predicate Logic for Software Engineering")

- Observation: rift in the use of mathematical modeling in everyday practice
  - In classical engineering (electrical, ...): mathematical modeling regular
  - In software "engineering": mathematical modeling rarely used (occasionally in critical systems under the name "Formal Methods")  
C. Michael Holloway: "software designers aspire to be(come) engineers"
- Differences reflected in design methods and support tools
  - Electronics engineers readily use, e.g., Matlab, Simulink (*textbook math*)
  - Software designers use acronym-ridden "soft" tools (with mathphobic notation), rarely provers or model checkers (problem: no common math)

## 0.1 Motivation: exploiting the many synergies between SP and FM

- Observations about SP

- + Traditionally mathematics-intensive; mathematical modeling is regular engineering practice — a rôle model for [wannabe] software engineers!
  - Still many defective formulations (examples shown soon)
- + Broadened from analog models/circuits: now also multidimensional information and algorithms implemented in software
  - Models for signals and systems are well-known, for algorithms and software largely unexploited by the SP community

- Observations about FM

- Growing importance of SP  $\Rightarrow$  more involvement of software engineers  
Consequence: future SWE requires signals and systems background (already observed by Parnas nearly two decades ago)  
Establishing this background via FM provides significant synergy.

## 0.2 Necessity and value of defect-free formalisms

Consider the degree of systematic symbolic calculation in engineering mathematics

- Well-developed in long-standing areas of mathematics (algebra, analysis, ...)

From: R. Bracewell / transforms

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

From: R. Blahut / data compacting

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

- Poorly developed in logical parts. This causes a serious style breach.

“The notation of elementary school arithmetic, which nowadays everyone takes for granted, took centuries to develop. There was an intermediate stage called *syncopation*, using abbreviations for the words for addition, square, root, *etc.* For example Rafael Bombelli (c. 1560) would write

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

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

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

in spite of the efforts of [Frege, Peano, Russell] [...]. Even now, mathematics students are expected to learn complicated ( $\epsilon$ - $\delta$ )-proofs in analysis with no help in understanding the logical structure of the arguments. Examiners fully deserve the garbage that they get in return.”

(Paul Taylor, “Practical Foundations of Mathematics”)

- Increasingly worse as we get closer to the necessities in computing/software (calculating with logic expressions, set expressions, etc.)

## Examples A: defects in often-used conventions in common mathematics

- Ellipsis, i.e., “omission dots” (...) 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*:

$$\sum_{i=1}^n \sum_{j=i}^m 1 = \frac{n \cdot (2 \cdot m - n + 1)}{2} \quad (1)$$

Letting  $n := 3$  and  $m := 1$  yields  $1 = 0$

In Maple: `sum(sum(1,j=i..m),i=1..n);` (observe the *awful* syntax!)

returns  $m(n+1) + \frac{3}{2}n + \frac{1}{2} - \frac{1}{2}(n+1)^2 - m$ , which is the same as (1).

Correct formula (obtained by formal calculation in Funmath):

$$\sum_{i=1}^n \sum_{j=i}^m 1 = (k \geq 1) \cdot \frac{k \cdot (2 \cdot m - k + 1)}{2} \text{ where } k := m \wedge n \quad (2)$$

## Examples B (closer to FM); 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  $\text{Even} := \{2 \cdot m \mid m \in \mathbb{Z}\}$  (set of even numbers) in

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

*Each example* matches *both patterns*, thereby illustrating the ambiguity.

Worse: notational defects *prohibit even the formulation of formal calculation rules!*  
Symptom: formal calculation with set expressions rare/nonexistent in the literature.

Observation: ambiguity and loss of opportunities inherited by many FM and tools!

**Examples C** (typical in SP) Serious cleanup also needed in “continuous” math.  
Defects in common conventions as pointed out (around 2000) in

E. Lee, P. Varaiya, *Introducing signals and systems: the Berkeley approach*  
<http://ptolemy.eecs.berkeley.edu/publications/papers/00/spe1/spe1.pdf>

- Using the function argument (in attempting to) characterize the domain  
Example:  $x(n) = x(nT)$  to express sampling
- Expressions, functions and function applications “systematically” confused  
Examples:  $y(t) = T(x(t))$  for systems behavior  
 $y(t) = x(t) * h(t)$  for convolution
- Consequences: no formal calculation rules, erroneous instantiation  
Example:  $y(t - \tau) = x(t - \tau) * h(t - \tau)$  (wrong from any viewpoint)

Note: using Funmath, all those defects were already eliminated before 1990.

### 0.3 Approach: exploiting the advantages of formalization

- “Formal approach” means: not just “using math”, but doing it *formally*
  - “formal” = manipulating expressions on the basis of their *form*
  - “informal” = manipulating expressions on the basis of their *meaning*
- Dispelling poor reputation of formal mathematics
  - Idea “difficult, tedious” deserved only where badly done (traditional logic)
  - Formality tacitly much appreciated where successful (algebra, calculus)
  - Practical application in critical HW/SW systems (well-known issue)
  - Even more important: **UT FACIANT OPUS SIGNA**  
(Maxim of the conferences on *Mathematics of Program Construction*)

Provides help in *thinking*: deriving guidance from the *shape* of formulas  
→ additional kind of / added dimension to intuition, tool for discovery!

- “All that remains” is showing how it is done

Making things simple required considerable thinking and effort!

## Next topic

09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms

### **Part I — Unifying the mathematics of SP and FM**

09:20–09:40 1. Review of the basics and outline of the formalism used

09:40–10:10 2. Inspiration from SP to FM: functions and functionals

10:10–10:30 3. Calculation with predicates and quantifiers for engineers

10:30–11:00 (Half-hour break)

### **Part II — Application examples to modeling in SP and FM**

11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP

11:40–11:45 5. Brief intermezzo about endosemantic functions

11:45–12:05 6. Modeling programs by program equations

12:05–12:25 7. Reasoning about temporal behavior by temporal operators

12:25–12:30 8. Final considerations

# 1 Review of the basics and outline of the formalism used

## 1.0 Basics reviewed: equality, proposition logic, sets, conditionals

### a. Rules for equational and calculational reasoning (e.g. [Gries & Schneider])

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

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

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

- *General inference rule*: For any theorem  $p$  (a propositional expression),

INSTANTIATION: from  $p$ , infer  $p[e^v]$ .

$d[e^v]$  or  $d[v := e]$  is substitution of expression(s)  $e$  for variable(s)  $v$  in  $d$

e.g.,  $(x + y = y + x)[x, y := y, z + x]$  is  $y + (z + x) = (z + x) + y$ .

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

LEIBNIZ'S PRINCIPLE: from  $e = e'$ , infer  $d[e^v] = d[e'^v]$

b. **Proposition calculus** Usual propositional operators  $\neg, \equiv, \Rightarrow, \wedge, \vee$ . Notes:

- Boolean equality ( $\equiv$ ) is associative:  $(x \equiv (y \equiv z)) \equiv ((x \equiv y) \equiv z)$ .
- Implication ( $\Rightarrow$ ) is *not* associative; we read  $p \Rightarrow q \Rightarrow r$  as  $p \Rightarrow (q \Rightarrow r)$ .
- For practical use, an *extensive* set of rules is needed [Gries, Boute] e.g.,  
*shunting*:  $p \Rightarrow q \Rightarrow r \equiv q \Rightarrow p \Rightarrow r$  and  $p \Rightarrow (q \Rightarrow r) \equiv (p \wedge q) \Rightarrow r$
- Leibniz's principle can be rewritten  $e = e' \Rightarrow d[e^v] = d[e'^v]$ .

c. **Calculating with sets** The basic operator is  $\in$ .

- The rules are derived ones (set calculus from proposition calculus), e.g.,

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

- *Set equality* is defined via

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

#### d. Binary algebra and calculating with conditionals

- *Principle:* binary algebra as a restriction of *minimax algebra*, i.e., *least upper bound* ( $\vee$ ) and *greatest lower bound* ( $\wedge$ ) over  $\mathbb{R}' := \mathbb{R} \cup \{-\infty, +\infty\}$ .

Definition:  $a \vee b \leq c \equiv a \leq c \wedge b \leq c$  and  $c \leq a \wedge b \equiv c \leq a \wedge c \leq b$

Restriction to  $\mathbb{B}$  illustrated by listing  $f_i(x, y)$  for  $i: 1..15$  (functions  $\mathbb{B}^2 \rightarrow \mathbb{B}$ ).

$x, y$	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
$\mathbb{B}$		$\forall$	$<$		$>$		$\neq$	$\wedge$	$\wedge$	$\equiv$	$\gg$	$\Rightarrow$	$\ll$	$\Leftarrow$	$\vee$	
$\mathbb{R}'$			$<$		$>$		$\neq$		$\wedge$	$=$	$\gg$	$\leq$	$\ll$	$\geq$	$\vee$	

- *Conditionals*  $c?e' \dagger e = (e, e')c$  (“if  $c$  then  $e'$  else  $e$ ”) Properties:

- $(c?f \dagger g)x = c?fx \dagger gx$  and  $f(c?x \dagger y) = c?fx \dagger fy$
- $c?b \dagger b' \equiv (c \Rightarrow b) \wedge (\neg c \Rightarrow b')$  for boolean  $b, b'$
- $z = (c?x \dagger y) \equiv (c \Rightarrow z = x) \wedge (\neg c \Rightarrow z = y)$

## 1.1 Formalism design: Functional Mathematics (Funmath)

- Unifying formalism for continuous and discrete mathematics
  - *Formalism* = *language* (notation) + *formal rules*
  - Unifying concept: *function* (= *domain* + *mapping*)  
Functions as first-class objects and basis for unification
- The language (characteristics)
  - No “ad hoc” patching of defects, but restart from systematic basis.
  - Simple structure: 4 constructs: identifier, application, abstraction, tupling
    - Synthesizing common notations, without their defects
    - Synthesizing new useful forms of expression, in particular: “point-free”,  
e.g.:  $square = times \circ duplicate$  versus  $square\ x = x\ times\ x$
    - If point-free power not used ( $pity^\infty!$ ), the notation looks “standard”
- Formal rules (main characteristic): *calculational*

*Warning:* here come a few language technicalities — but they eliminate all defects!

## The four constructs

Overview: 0. identifier, 1. application, 2. abstraction, 3. tupling

0. **Identifier**: any symbol or string except a (very) few keywords.

- Established symbols (e.g.,  $\mathbb{B}$ ,  $\Rightarrow$ ,  $\mathbb{R}$ ,  $+$ ) taken as predefined constants.
- Identifiers are *introduced* (or *declared*) by *bindings*

General form:  $i : X \wedge p$ , read “ $i$  in  $X$  satisfying  $p$ ” ( $\wedge p$  optional)

Here  $i$  is the (tuple of) identifier(s),  $X$  a set and  $p$  a proposition.

Example:  $n : \mathbb{N}$  is interchangeable with  $n : \mathbb{Z} \wedge n \geq 0$

- Identifiers come in two flavors.
  - *Constants*: declared by
    - a *specification* of the form  $\text{spec binding}$  (no obligations)
    - a *definition* of the form  $\text{def binding}$  (proof obligation:  $\exists!$ )
  - *Variables*: in an *abstraction*  $\text{binding} . \text{expression}$  (soon!)

## 1. Function application:

- Default form:  $\boxed{f x}$  for function  $f$  and argument  $e$   
Other affix conventions: by dashes in the binding, e.g.,  $— \star —$  for infix.
- Role of parentheses: *never* used as operators.  
Only for parsing (overruling/emphasizing affix conventions/precedence).  
If  $f$  is a function-valued function,  $\boxed{f x y}$  stands for  $(f x) y$

- Special application forms for any infix operator  $\star$ 
  - *Partial application* is of the form  $a \star$  or  $\star b$ , and is defined by

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

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

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

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

Examples:  $a + b + c = \sum(a, b, c)$  and  $a \neq b \neq c \equiv \text{inj}(a, b, c)$ .

## 2. Abstraction:

- General form:  $\boxed{b.e}$  where  $b$  is a binding and  $e$  an expression

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

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

– Mapping: maps value  $v$  to value  $e$ . Formally:  $(v : X \wedge p . e) d = e \Big|_d^v$ .

- Examples

(i) The function  $n : \mathbb{Z} . 2 \cdot n$  doubles every integer.

(ii) If  $v$  not free in  $e$  (trivial case), we define  $\bullet$  by  $\boxed{X \bullet e = v : X . e}$

Illustration:  $(\mathbb{Z} \bullet 3) 7 = 3$

- Syntactic sugar (not new constructs!):

$\boxed{e \mid b}$  stands for  $b . e$     and     $\boxed{v : X \mid p}$  stands for  $v : X \wedge p . v$ .

Note: abstractions help synthesizing familiar expressions

Examples:  $\boxed{\sum i : 0 .. n . q^i}$      $\boxed{\{m \cdot n \mid m : \mathbb{Z}\}}$      $\boxed{\{m : \mathbb{Z} \mid m < n\}}$

### 3. Tupling:

- General form:  $\boxed{e, e', e''}$  (any length) for 1 dimension  
Intuitive meaning:  $e, e', e''$  denotes a *function*
  - Domain:  $\mathcal{D}(e, e', e'') = \{0, 1, 2\}$
  - Mapping:  $(e, e', e'') 0 = e$  and  $(e, e', e'') 1 = e'$  and  $(e, e', e'') 2 = e''$ .
- Parentheses are *not* part of tupling:  
they are as optional in  $(m, n)$  as they are optional in  $(m + n)$ .
- The empty tuple is  $\varepsilon$  and for singleton tuples we define  $\tau$  with  $\tau e = 0 \mapsto e$ .  
Legend: here we used two particular forms of  $\bullet$ :
  - we define  $\varepsilon$  by  $\varepsilon := \emptyset \bullet e$  (any  $e$ ) for the *empty function*;
  - we define  $\mapsto$  by  $d \mapsto e = \iota d \bullet e$  for *one-point functions*.
- Matrices are 2-dimensional tuples.

*Relax!* This concludes the language technicalities.

## Next topic

- 09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms
- Part I — Unifying the mathematics of SP and FM**
- 09:20–09:40 1. Review of the basics and outline of the formalism used
- 09:40–10:10 2. Inspiration from SP to FM: functions and functionals
- 10:10–10:30 3. Calculation with predicates and quantifiers for engineers
- 10:30–11:00 (Half-hour break)
- Part II — Application examples to modeling in SP and FM**
- 11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP
- 11:40–11:45 5. Brief intermezzo about endosemantic functions
- 11:45–12:05 6. Modeling programs by program equations
- 12:05–12:25 7. Reasoning about temporal behavior by temporal operators
- 12:25–12:30 8. Final considerations

## 2 Inspiration from SP to FM: functions and functionals

### 2.0 Proper use of functions and functionals in SP: opportunities

- a. Starting example: criticism of Lee & Varaiya about the common yet nonsensical

$$z(t) = x(t) * y(t) \quad \text{and instantiation} \quad z(t - \tau) = x(t - \tau) * y(t - \tau)$$

How to do it right? Approach by L&V: see web. In the Funmath framework:

- **Conventions** For value space  $A$  and time domain  $\mathbb{T}$ , define  $\mathcal{S}_A = \mathbb{T} \rightarrow A$ .  
*Signal*: function of type  $\mathcal{S}_A$ . *System*: function of type  $\mathcal{S}_A \rightarrow \mathcal{S}_B$ .  
Response of  $s: \mathcal{S}_A \rightarrow \mathcal{S}_B$  to signal  $x: \mathcal{S}_A$  at time  $t: \mathbb{T}$  is  $sxt$ , read  $(sx)t$ .
- **Convolution** Proper definition: for signals  $x$  and  $y$

$$(x * y)t = \int_{-\infty}^{+\infty} x\tau \cdot y(t - \tau) \cdot d\tau$$

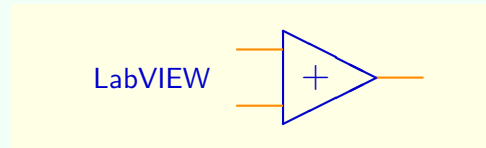
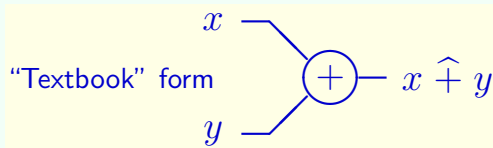
Using the *shift* operator **def**  $\sigma_-: \mathbb{T} \rightarrow \mathcal{S}_A \rightarrow \mathcal{S}_A$  **with**  $\sigma_\tau xt = x(t + \tau)$   
one can now express convolution of shifted signals properly as

$$\sigma_\tau x * \sigma_\nu y = \sigma_{\tau+\nu}(x * y) \quad \text{and hence} \quad \sigma_{-\tau} x * \sigma_{-\tau} y = \sigma_{-2\tau}(x * y)$$

b. Functionals for structuring signal and system models: typical needs

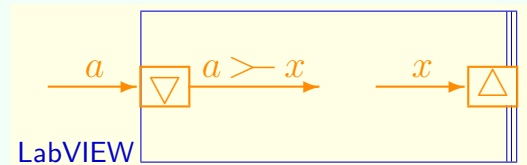
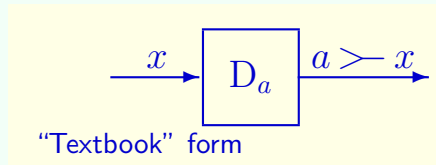
- Proper formal modeling of memoryless devices for pointwise operations

- E.g., sum of signals  $x$  and  $y$  modelled as  $(x \hat{+} y) t = x t + y t$
- Explicit *direct extension* operator  $\hat{\phantom{x}}$  (in engineering often left implicit)



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

$$D_a x n = (n = 0) ? a \dagger x(n - 1) \text{ or, in point-free form, } D_a x = a \succ x$$

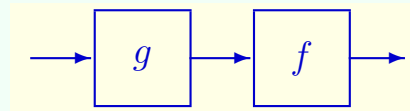


- Combining systems and interconnections

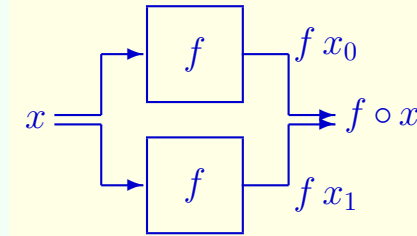
- The manifold role of function composition ( $\circ$ )

- \* Image definition:  $(f \circ g) x = f (g x)$

- \* Origin: cascade connection (classical)



- \* Other opportunity (in functional framework): system replication



- \* Yet another opportunity: direct extension of 1-argument operators: extension operator  $\overline{\quad}$  with mapping  $\overline{g} x = g \circ x$

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

- \* Yet another: subsumes `map` from functional programming.

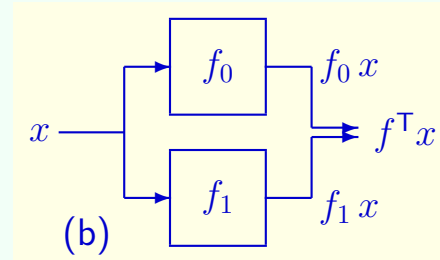
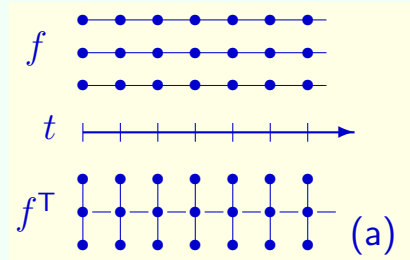
– A third operator: transposition (abeside composition, extension)

\* Image definition:  $f^\top y x = f x y$  (swapping arguments)

Name obviously borrowed from matrix theory

\* (a) Origin: from a family of signals to a tuple-valued signal

\* (b) Other opportunity: signal replication (fanout)



\* Other opportunity: subsumes `zip` from functional programming

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

(taking lists as functions, and up to currying)

## 2.1 Formalization: calculating with functions and generic functionals

### a. General rules for functions

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

$$\text{Leibniz: } f = g \Rightarrow \mathcal{D} f = \mathcal{D} g \wedge (x \in \mathcal{D} f \cap \mathcal{D} g \Rightarrow f x = g x)$$

$$\text{Extensionality: } \frac{p \Rightarrow \mathcal{D} f = \mathcal{D} g \wedge (x \in \mathcal{D} f \cap \mathcal{D} g \Rightarrow f x = g x)}{p \Rightarrow f = g}$$

- Abstraction encapsulates substitution. Formal axioms:

$$\text{Domain axiom: } d \in \mathcal{D}(v : X \wedge p . e) \equiv d \in X \wedge p[d^v]$$

$$\text{Mapping axiom: } d \in \mathcal{D}(v : X \wedge p . e) \Rightarrow (v : X \wedge p . e) d = e[d^v]$$

Equality is characterized via function equality (exercise).

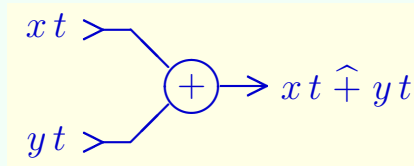
## b. Making functionals generic

- Goals:

(a) Removing restrictions in common functionals from mathematics.

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

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



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

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

Made explicit:  $(x \hat{+} y) t = x t + y t$  (“direct extension”  $\hat{+}$ )

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

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

- Design illustrating goal (a): *composition* ( $\circ$ )

For any functions  $f, g$ ,

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

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

- Design illustrating goal (b): *Duplex direct extension* ( $\hat{\ }$ )

For any functions  $\star$  (infix),  $f, g$ ,

$$f \hat{\ } g = x : \mathcal{D}f \cap \mathcal{D}g \wedge (fx, gx) \in \mathcal{D}(\star) . fx \star gx$$

Example: given  $f : \mathbb{N} \rightarrow \mathbb{R}$  and  $g : \mathbb{Z} \rightarrow \mathbb{C}$  we get  $\mathcal{D}(f \hat{\ } g) = \mathbb{N}$ .

Often we need *half direct extension*: for function  $f$ , any  $e$ ,

$$f \overleftarrow{\ } e = f \hat{\ } (\mathcal{D}f \bullet e) \quad \text{and} \quad e \overrightarrow{\ } f = (\mathcal{D}f \bullet e) \hat{\ } f$$

Typical algebraic property:  $x \overrightarrow{\ } f = (x \star) \circ f$

*Simplex direct extension* ( $\overline{\ }$ ) is defined by

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

c. Some other important generic functionals

- *Function merge* ( $\cup$ ) is defined in 2 parts to fit the line:

$$\begin{aligned} x \in \mathcal{D}(f \cup g) &\equiv x \in \mathcal{D}f \cup \mathcal{D}g \wedge (x \in \mathcal{D}f \cap \mathcal{D}g \Rightarrow f x = g x) \\ x \in \mathcal{D}(f \cup g) &\Rightarrow (f \cup g) x = (x \in \mathcal{D}f) ? f x \uparrow g x \end{aligned}$$

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

$$f \downarrow P = x : \mathcal{D}f \cap \mathcal{D}P \wedge P x . f x$$

A particularization is the familiar *restriction* ( $\upharpoonright$ ):  $f \upharpoonright X = f \downarrow (X \bullet 1)$ .

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

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

- *Function compatibility* ( $\odot$ ) is a relation on functions:

$$f \odot g \equiv f \upharpoonright \mathcal{D}g = g \upharpoonright \mathcal{D}f$$

Algebraic property:  $f = g \equiv \mathcal{D}f = \mathcal{D}g \wedge f \odot g$ .

## 2.2 Extended SP example: functional characterization of system properties

### a. Reminder: conventions

Define  $\mathcal{S}_A = \mathbb{T} \rightarrow A$  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  $\mathcal{S}_A \rightarrow \mathcal{S}_B$ .

Note: the response of  $s: \mathcal{S}_A \rightarrow \mathcal{S}_B$  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

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

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

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

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

$$\text{linear} \quad \text{iff} \quad \forall z : \mathcal{S}_{\mathbb{C}} . \forall c : \mathbb{C} . s(c \vec{\cdot} z) = c \vec{\cdot} s z$$

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

### c. Response of LTI systems

Define the parametrized exponential  $E_c : \mathbb{C} \rightarrow \mathbb{T} \rightarrow \mathbb{C}$  with  $E_c t = e^{c \cdot t}$

Then we have:

**THEOREM:** if  $s$  is LTI then  $s E_c = s E_c 0 \cdot \vec{\cdot} E_c$

Proof: we calculate  $s E_c(t + \tau)$  to exploit all properties.

$$\begin{aligned}
 s E_c(t + \tau) &= \langle \text{Definition } \sigma \rangle \sigma_\tau (s E_c) t \\
 &= \langle \text{Time inv. } s \rangle s (\sigma_\tau E_c) t \\
 &= \langle \text{Property } E_c \rangle s (E_c \tau \cdot \vec{\cdot} E_c) t \\
 &= \langle \text{Linearity } s \rangle (E_c \tau \cdot \vec{\cdot} s E_c) t \\
 &= \langle \text{Defintion } \vec{\cdot} \rangle E_c \tau \cdot s E_c t
 \end{aligned}$$

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

The  $\langle \text{Property } E_c \rangle$  is  $\sigma_\tau E_c = E_c \tau \cdot \vec{\cdot} E_c$  (easy to prove).

Note that this proof uses only the essential hypotheses.

## Next topic

09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms

### **Part I — Unifying the mathematics of SP and FM**

09:20–09:40 1. Review of the basics and outline of the formalism used

09:40–10:10 2. Inspiration from SP to FM: functions and functionals

10:10–10:30 3. Calculation with predicates and quantifiers for engineers

10:30–11:00 (Half-hour break)

### **Part II — Application examples to modeling in SP and FM**

11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP

11:40–11:45 5. Brief intermezzo about endosemantic functions

11:45–12:05 6. Modeling programs by program equations

12:05–12:25 7. Reasoning about temporal behavior by temporal operators

12:25–12:30 8. Final considerations

### 3 **Calculation with predicates and quantifiers for engineers**

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

#### 3.0 **Axioms and forms of expression**

- a. **Basic axioms** *quantifiers* ( $\forall, \exists$ ) are predicates on predicates defined by

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

- b. **Forms of expression**

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

### 3.1 Derived rules

- a. **Duality** relating  $\forall/\exists$  (or *generalized De Morgan's law*)

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

- b. **Distributivity rules** (each has a dual, not stated here):

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

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

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

Distributivity $\vee/\forall$ generalizes	$q \vee (r \wedge s) \equiv (q \vee r) \wedge (q \vee s)$
L(eft)-distrib. $\Rightarrow/\forall$ generalizes	$q \Rightarrow (r \wedge s) \equiv (q \Rightarrow r) \wedge (q \Rightarrow s)$
R(ight)-distr. $\Rightarrow/\exists$ generalizes	$(r \vee s) \Rightarrow q \equiv (r \Rightarrow q) \wedge (s \Rightarrow q)$
P(seudo)-dist. $\wedge/\forall$ generalizes	$q \wedge (r \wedge s) \equiv (q \wedge r) \wedge (q \wedge s)$

### c. Some additional laws

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

Note:  $\forall/\wedge$  assumes  $\mathcal{D}P = \mathcal{D}Q$ . Otherwise,  $\forall P \wedge \forall Q \Rightarrow \forall(P \widehat{\wedge} Q)$ .

Just one derivation example:

$\forall P \wedge \forall Q$	
$\equiv$	$\langle \text{Def. } \forall \rangle \quad P = \mathcal{D}P \bullet 1 \wedge Q = \mathcal{D}Q \bullet 1$
$\Rightarrow$	$\langle \text{Leibniz} \rangle \quad \forall(P \widehat{\wedge} Q) \equiv \forall(\mathcal{D}P \bullet 1 \widehat{\wedge} \mathcal{D}Q \bullet 1)$
$\equiv$	$\langle \text{Def. } \widehat{\wedge} \rangle \quad \forall(P \widehat{\wedge} Q) \equiv \forall x : \mathcal{D}P \cap \mathcal{D}Q . (\mathcal{D}P \bullet 1) x \wedge (\mathcal{D}Q \bullet 1) x$
$\equiv$	$\langle \text{Def. } \bullet \rangle \quad \forall(P \widehat{\wedge} Q) \equiv \forall x : \mathcal{D}P \cap \mathcal{D}Q . 1 \wedge 1$
$\equiv$	$\langle \forall(X \bullet 1) \rangle \quad \forall(P \widehat{\wedge} Q)$

d. Function range: definition and derived rules for quantifiers

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

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

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

Pointwise form:  $\forall (y : \mathcal{R} f . p) \equiv \forall (x : \mathcal{D} f . p[\frac{y}{f x}])$  (“dummy change”).

e. Function range: making set comprehension calculational

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

Consequence: defect-free set notation:

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

## 3.2 Unifying induction principles via predicate calculus (SKIP)

- a. Some definitions: for any relation  $\prec : X^2 \rightarrow \mathbb{B}$ , any  $S : \mathcal{P} X$  and any  $x : X$ ,

Minimal element:  $x \text{ ismin}_{\prec} S \equiv x \in S \wedge \forall y : X . y \prec x \Rightarrow y \notin S$

Least element:  $x \text{ isleast}_{\prec} S \equiv x \in S \wedge \forall y : S . x \prec y$

- b. Well-foundedness and supporting induction

- Defining *Well-Foundedness*: every nonempty subset has a minimal element

$$\text{WF}(\prec) \equiv \forall S : \mathcal{P} X . S \neq \emptyset \Rightarrow \exists x : X . x \text{ ismin}_{\prec} S$$

- Definition, *Supporting Induction*:

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

- Equivalence theorem [Gries, Dijkstra, ...]

$$\text{THEOREM, EQUIVALENCE OF WF AND SI: } \text{WF}(\prec) \equiv \text{SI}(\prec)$$

PROOF: next image (in functional predicate calculus)

$WF(\prec)$   
 $\equiv \langle \text{Definition WF) and } S \neq \emptyset \equiv \exists x: S . 1 \rangle$   
 $\forall S: \mathcal{P} X . \exists (x: S . 1) \Rightarrow \exists (x: X . x \text{ ismin}_{\prec} S)$   
 $\equiv \langle S = X \cap S, \text{ trading} \rangle$   
 $\forall S: \mathcal{P} X . \exists (x: X . x \in S) \Rightarrow \exists (x: X . x \text{ ismin}_{\prec} S)$   
 $\equiv \langle \text{Definition } \textit{ismin} \rangle$   
 $\forall S: \mathcal{P} X . \exists (x: X . x \in S) \Rightarrow \exists (x: X . x \in S \wedge \forall y: X . y \prec x \Rightarrow y \notin S)$   
 $\equiv \langle p \Rightarrow q \equiv \neg q \Rightarrow \neg p \rangle$   
 $\forall S: \mathcal{P} X . \neg (\exists x: X . x \in S \wedge \forall y: X . y \prec x \Rightarrow y \notin S) \Rightarrow \neg (\exists x: X . x \in S)$   
 $\equiv \langle \text{Duality } \forall/\exists, \text{ De Morgan} \rangle$   
 $\forall S: \mathcal{P} X . \forall (x: X . x \notin S \vee \neg (\forall y: X . y \prec x \Rightarrow y \notin S)) \Rightarrow \forall x: X . x \notin S$   
 $\equiv \langle \forall \text{ to } \Rightarrow, \text{ i.e., } a \vee \neg b \equiv b \Rightarrow a \rangle$   
 $\forall S: \mathcal{P} X . \forall (x: X . \forall (y: X . y \prec x \Rightarrow y \notin S) \Rightarrow x \notin S) \Rightarrow \forall x: X . x \notin S$   
 $\equiv \langle \text{Dummy change using } f: \text{pred}_X \rightarrow \mathcal{P} X \text{ with } x \in f x \equiv \neg (P x) \rangle$   
 $\forall P: X \rightarrow B . \forall (x: X . \forall (y: X . y \prec x \Rightarrow P y) \Rightarrow P x) \Rightarrow \forall x: X . P x$   
 $\equiv \langle \text{Trading, definition SI} \rangle$   
 $SI (\prec)$

c. Important particular instances of well-founded induction

- Induction over  $\mathbb{N}$  (predicates  $P : \mathbb{N} \rightarrow \mathbb{B}$ ) An axiom for natural numbers:

Every nonempty subset of  $\mathbb{N}$  has a *minimal* element under  $<$ .

Equivalently, every nonempty subset of  $\mathbb{N}$  has a *least* element under  $\leq$ .

Strong induction over  $\mathbb{N}$ : define  $\prec$  in SI by  $m \prec n \equiv m < n$

$$\forall (n : \mathbb{N}. P n) \equiv \forall (n : \mathbb{N}. \forall (m : \mathbb{N}. m < n \Rightarrow P m) \Rightarrow P n)$$

Weak induction over  $\mathbb{N}$ : define  $\prec$  in SI by  $m \prec n \equiv m + 1 = n$

$$\forall (n : \mathbb{N}. P n) \equiv P 0 \wedge \forall (n : \mathbb{N}. P n \Rightarrow P (n + 1)).$$

- Structural induction over lists in  $A^*$  (predicates  $P : A^* \rightarrow \mathbb{B}$ )

List prefix is well-founded and yields

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

Suffices for proving most properties about functional programs with lists.

d. Illustration: proving a property of the Fibonacci numbers

Given  $\boxed{\text{def } F_{-} : \mathbb{N} \rightarrow \mathbb{N} \text{ with } F_0 = 0 \wedge F_1 = 1 \wedge F_{n+2} = F_{n+1} + F_n}$

To prove  $\boxed{\forall m : \mathbb{N}. \forall n : \mathbb{N}. F_{m+n+1} = F_{m+1} \cdot F_{n+1} + F_m \cdot F_n}$  we define

$\boxed{P : \mathbb{N} \rightarrow \mathbb{B} \text{ with } P n \equiv \forall m : \mathbb{N}. F_{m+n+1} = F_{m+1} \cdot F_{n+1} + F_m \cdot F_n}$

and prove  $\forall P$  by induction, i.e.,  $P 0 \wedge \forall (n : \mathbb{N}. P n \Rightarrow P (n + 1))$ .

(0) Proving  $P 0$ , i.e.,  $\forall m : \mathbb{N}. F_{m+1} = F_{m+1} \cdot F_1 + F_m \cdot F_0$  is trivial.

(1) Proving  $\forall (n : \mathbb{N}. P n \Rightarrow P (n + 1))$ : for given  $n$ , we assume  $P n$  (IH) and

prove  $\boxed{P (n + 1)}$ , i.e.,  $\forall m : \mathbb{N}. F_{m+(n+1)+1} = F_{m+1} \cdot F_{(n+1)+1} + F_m \cdot F_{n+1}$

as follows: for arbitrary  $m : \mathbb{N}$ , we calculate  $F_{m+(n+1)+1}$ .

$$\begin{aligned} F_{m+(n+1)+1} &= \langle \text{Assoc. } + \rangle F_{(m+1)+n+1} \\ &= \langle \text{Instant. IH} \rangle F_{m+2} \cdot F_{n+1} + F_{m+1} \cdot F_n \\ &= \langle \text{Def. F} \rangle (F_{m+1} + F_m) \cdot F_{n+1} + F_{m+1} \cdot F_n \\ &= \langle \text{Arithmetic} \rangle F_{m+1} \cdot (F_{n+1} + F_n) + F_m \cdot F_{n+1} \\ &= \langle \text{Def. F} \rangle F_{m+1} \cdot F_{n+2} + F_m \cdot F_{n+1} \end{aligned}$$

## Next topic

09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms

### **Part I — Unifying the mathematics of SP and FM**

09:20–09:40 1. Review of the basics and outline of the formalism used

09:40–10:10 2. Inspiration from SP to FM: functions and functionals

10:10–10:30 3. Calculation with predicates and quantifiers for engineers

10:30–11:00 (Half-hour break)

### **Part II — Application examples to modeling in SP and FM**

11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP

11:40–11:45 5. Brief intermezzo about endosemantic functions

11:45–12:05 6. Modeling programs by program equations

12:05–12:25 7. Reasoning about temporal behavior by temporal operators

12:25–12:30 8. Final considerations

## 4 Using predicate calculus and generic functionals in SP

### 4.0 Basic level: analysis — Calculation, not syncopation; example: limits

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

Example: proving the *closure property*  $\text{closed } P \equiv \text{ad } P = P$

$\text{closed } P$

$\equiv$   $\langle \text{Definit. closed} \rangle \text{ open } (\neg P)$   
 $\equiv$   $\langle \text{Definit. open} \rangle \forall v : \mathbb{R}_{\neg P} . \exists \epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . |x - v| < \epsilon \Rightarrow \neg P x$   
 $\equiv$   $\langle \text{Trading sub } \forall \rangle \forall v : \mathbb{R} . \neg P v \Rightarrow \exists \epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . |x - v| < \epsilon \Rightarrow \neg P x$   
 $\equiv$   $\langle \text{Contraposit.} \rangle \forall v : \mathbb{R} . \neg \exists (\epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . P x \Rightarrow \neg (|x - v| < \epsilon)) \Rightarrow P v$   
 $\equiv$   $\langle \text{Duality, twice} \rangle \forall v : \mathbb{R} . \forall (\epsilon : \mathbb{R}_{>0} . \exists x : \mathbb{R} . P x \wedge |x - v| < \epsilon) \Rightarrow P v$   
 $\equiv$   $\langle \text{Definition ad} \rangle \forall v : \mathbb{R} . \text{ad } P v \Rightarrow P v$   
 $\equiv$   $\langle P v \Rightarrow \text{ad } P v \rangle \forall v : \mathbb{R} . \text{ad } P v \equiv P v$  (proving  $P v \Rightarrow \text{ad } P v$  is simple)

## Calculational reasoning about limits (definition taken from S. Lang)

Define  $L \text{ islim}_f a \equiv \forall \epsilon : \mathbb{R}_{>0}. \exists \delta : \mathbb{R}_{>0}. \forall x : \mathcal{D} f. |x - a| < \delta \Rightarrow |f x - L| < \epsilon$

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

- (i)  $\exists (L : \mathbb{R}. L \text{ islim}_f a) \Rightarrow \exists (L : \mathbb{R}. L \text{ islim}_{f \upharpoonright_S} a)$ ,
- (ii)  $\forall L : \mathbb{R}. \forall M : \mathbb{R}. L \text{ islim}_f a \wedge M \text{ islim}_{f \upharpoonright_S} a \Rightarrow L = M$

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

$L \text{ islim}_f a \wedge M \text{ islim}_{f \upharpoonright_S} a$

$\Rightarrow \langle \text{Hint in prf. (i)} \rangle L \text{ islim}_{f \upharpoonright_S} a \wedge M \text{ islim}_{f \upharpoonright_S} a$

$\equiv \langle \text{Def. islim, hyp.} \rangle \forall (\epsilon : \mathbb{R}_{>0}. \exists \delta : \mathbb{R}_{>0}. L R \delta) \wedge \forall (\epsilon : \mathbb{R}_{>0}. \exists \delta : \mathbb{R}_{>0}. M R \delta)$

$\equiv \langle \text{Distribut. } \forall / \wedge \rangle \forall \epsilon : \mathbb{R}_{>0}. \exists (\delta : \mathbb{R}_{>0}. L R \delta) \wedge \exists (\delta : \mathbb{R}_{>0}. M R \delta)$

$\equiv \langle \text{Distribut. } \wedge / \exists \rangle \forall \epsilon : \mathbb{R}_{>0}. \exists \delta : \mathbb{R}_{>0}. \exists \delta' : \mathbb{R}_{>0}. L R \delta \wedge M R \delta'$

$\Rightarrow \langle \text{Closeness lem.} \rangle \forall \epsilon : \mathbb{R}_{>0}. \exists \delta : \mathbb{R}_{>0}. \exists \delta' : \mathbb{R}_{>0}. a \in \mathbf{Ad} S \Rightarrow |L - M| < 2 \cdot \epsilon$

$\equiv \langle \text{Hyp. } a \in \mathbf{Ad} S \rangle \forall \epsilon : \mathbb{R}_{>0}. \exists \delta : \mathbb{R}_{>0}. \exists \delta' : \mathbb{R}_{>0}. |L - M| < 2 \cdot \epsilon$

$\equiv \langle \text{Const. pred. } \exists \rangle \forall \epsilon : \mathbb{R}_{>0}. |L - M| < 2 \cdot \epsilon$

$\equiv \langle \text{Vanishing lem.} \rangle L - M = 0$

$\equiv \langle \text{Leibniz, inv. } + \rangle L = M$

#### 4.1 At the functional level in SP: transforms as functionals

- a. The Fourier transform as a functional Note: *not*  $\mathcal{F}\{f(t)\}$  but  $\mathcal{F}f\omega$

$$\mathcal{F}f\omega = \int_{-\infty}^{+\infty} e^{-j\omega t} \cdot f(t) \cdot dt \quad \text{and} \quad \mathcal{F}'g(t) = \frac{1}{2 \cdot \pi} \cdot \int_{-\infty}^{+\infty} e^{j\omega t} \cdot g(\omega) \cdot d\omega$$

Clear and unambiguous bindings allow formal calculation.

- b. Functional Laplace transform via Fourier transform.

Conditioning function:  $\ell_{\sigma} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$  with  $\ell_{\sigma} t = (t < 0) ? 0 : e^{-\sigma t}$

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

$$\mathcal{L}f(\sigma + j \cdot \omega) = \mathcal{F}(\ell_{\sigma} \hat{\cdot} f)\omega$$

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

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

$$\mathcal{L}f s = \int_0^{+\infty} f(t) \cdot e^{-s \cdot t} \cdot dt$$

c. Calculation example: the inverse Laplace transform

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

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

Calculation of an explicit expression: For  $t$  as specified,

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

## 4.2 Formal reasoning about discrete system models in SP

We consider systems  $s : A^* \rightarrow B^*$  (Note:  $X^* = \bigcup n : \mathbb{N}. X^n$ )

- a. **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 **sequential** iff  $x \leq y \Rightarrow s x \leq s y$

$r : (A^*)^2 \rightarrow B^*$  is a **residual behavior** of  $s$  iff  $s(x ++ y) = s x ++ r(x, y)$

**THEOREM:**  $s$  is sequential iff it has a residual behavior function.

Proof: we start from the sequentiality side.

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

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

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

b. **Derivatives and primitives** The preceding framework leads to the following.

- Observation: An rb function is unique (exercise).
- We define the *derivation* operator  $D$  on sequential systems by

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

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

- *Primitivation*  $I$  is defined for any  $g: A^* \rightarrow B^*$  by

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

- Properties (note a striking analogy from analysis)

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

In the second row,  $D$  is derivation as in analysis, and  $I g x = \int_0^x g y \cdot dy$ .

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

### 4.3 Intermezzo: getting things right in discrete mathematics

- a. Errors in mathematical software (more in “discrete” than in “continuous”)

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

Taking  $n := 3$  and  $m := 1$  yields  $1 = 0$ .

Cause of errors:  $\sum$  *not* well-understood, formal rules rare ( $\rightarrow$  syncopation)

- b. Getting things right by proper formalization and calculation

- Proper formal definition (Funmath): for any  $a$ , any number  $c$  and any number-valued functions  $f$  and  $g$  with finite nonintersecting domains:

$$\sum \varepsilon = 0 \quad \sum (a \mapsto c) = c \quad \sum (f \cup g) = \sum f + \sum g$$

Extension to infinite (but ordered) domains by the usual limit construction.

Classical notation  $\sum_{i=m}^n f_i$  defined as shorthand for  $\sum i : m .. n . f_i$ .

- Formal calculation (with *trading*  $\sum f_P = \sum (P \hat{=} f)$  as the star) yields

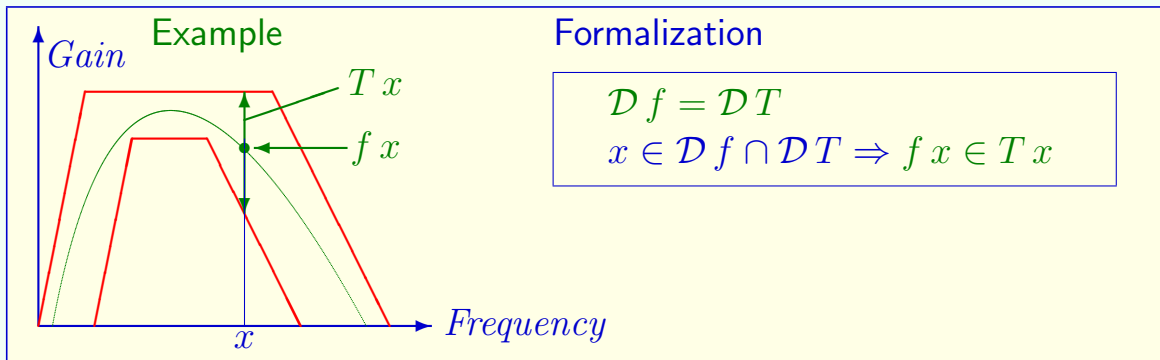
$$\sum_{i=1}^n \sum_{j=i}^m 1 = (k \geq 1) ? \frac{k \cdot (2 \cdot m - k + 1)}{2} + 0 \textbf{ where } k := m \wedge n$$

## 4.4 Tolerance on functions and the FunCart product

### a. Motivation from SP

- Tolerances for scalars: used routinely for all classical engineering artefacts
- Tolerances for functions: formalizing a convention in communications:  
A *tolerance function*  $T$  specifies for every domain value  $x$  the set  $T x$  of allowed function values. Note:  $\mathcal{D} T$  also taken as the domain specification.

Example: SP filter characteristic and its formalization



## b. The FunCart operator and ramifications

- *Defining the FunCart operator*  $\times$ : for *any* family  $T$  of sets,

Definition:  $f \in \times T \equiv \mathcal{D}f = \mathcal{D}T \wedge \forall x: \mathcal{D}f \cap \mathcal{D}T. fx \in Tx$   
 equivalently,  $\times T = \{f: \mathcal{D}T \rightarrow \bigcup T \mid \forall x: \mathcal{D}f \cap \mathcal{D}T. fx \in Tx\}$

- *Some properties* illustrating why  $\times$  is our “workhorse” for types

The familiar $\times$	$A \times B = \times(A, B)$ (for any sets $A$ and $B$ )
Function type	$A \rightarrow B = \times(A \bullet B)$ (idem)
Point-free form	$\times T = \{f: \mathcal{D}T \rightarrow \bigcup T \mid \forall (f \widehat{\in} T)\}$
Explicit inverse	$\times - S = x: \bigcup (f: S. \mathcal{D}f). \{fx \mid f: S\}$
Equal functions	$f = g \equiv f \in \times(\iota \circ g)$ (exact, “zero tolerance”)
Dependent type	$\times(a: A. B_a) = \{f: A \rightarrow \bigcup (a: A. B_a) \mid \forall a: A. fa \in B_a\}$

Useful shorthand:  $A \ni a \rightarrow B_a$  for  $\times a: A. B_a$ , as in:  $A \ni a \rightarrow B_a \ni b \rightarrow C_{a,b}$

As expected: more to follow later!

## 4.5 Generic functionals for overloading and polymorphism (SKIP)

### a. Terminology (0) and main issues (1)

- (0) Overloading: same identifier designating “different” objects (functions).  
Polymorphism: different argument types, formally same image definition.
- (1) Disambiguation: via argument type. Means: compatibility (©)  
Refined typing: link argument/result type. Means: proper operator

### b. Kinds of overloading/polymorphism

- By explicit parametrization Trivial with  $\times$ .  
Example: *binary addition* function adding two binary words of equal length.

```
def binadd_:  $\times n : \mathbb{N} . (\mathbb{B}^n)^2 \rightarrow \mathbb{B}^{n+1}$  with  $binadd_n(x, y) = \dots$ 
```

- Without auxiliary parameter (to be designed next)  
Requirement: operator  $\otimes$  with properties exemplified for *binadd* by

```
def binadd:  $\otimes n : \mathbb{N} . (\mathbb{B}^n)^2 \rightarrow \mathbb{B}^{n+1}$  with  $binadd(x, y) = \dots$ 
```

c. An interesting design satisfying the requirement

- Principle (explained via the example)

`binadd` as a *merge* of functions of type  $(\mathbb{B}^n)^2 \rightarrow \mathbb{B}^{n+1}$  (various  $n$ ).

Family of functions merged taken from  $\times n : \mathbb{N} . (\mathbb{B}^n)^2 \rightarrow \mathbb{B}^{n+1}$

Requirement: *compatibility* of merged functions (merging selectively)

- Generic functional: for 2 arbitrary function types (function sets)  $F, G$ :

$$F \otimes G = \{f \cup g \mid f, g : F \times G \wedge f \odot g\}$$

- Elastic extension to arbitrary function type families:

$$\mathbf{def} \otimes : \mathbf{fam} (\mathcal{P} \mathcal{F}) \rightarrow \mathcal{P} \mathcal{F} \mathbf{ with } \otimes T = \{\cup f \mid f : (\times T) \odot\}$$

- Applications for other purposes than polymorphism shown later.

## 4.6 How generic functionals motivated by SP simplify theories in CS

a. The funcart operator  $\times$  as the “workhorse” for function typing

- Recall  $A \rightarrow B = \times(A \bullet B)$  and  $A \times B = \times(A, B)$
- Array types: for set  $A$  and  $n: \mathbb{N} \cup \iota \infty$ , define

$$A \uparrow n = \square n \rightarrow A$$

General shorthand:  $a^b$  for  $a \uparrow b$ .

Note:  $A^n$  is the  $n$ -fold Cartesian product, since  $A \uparrow n = \times(\square n \bullet A)$

- Stream types for infinite sequences: simply  $A^\infty$  Note:  $A^\infty = \mathbb{N} \rightarrow A$ .
- List types for finite sequences

$$A^* = \bigcup_{n: \mathbb{N}} A^n$$

- Sequence types for any sequences:  $A^\omega = A^* \cup A^\infty$

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

- Well-known approach: selector function for each field label.  
(e.g., Haskell)

Problem: records themselves are arguments, not functions.

- Preferred alternative: generalized functional cartesian product  $\times$ : records as *functions*, domain: set of field labels from an *enumeration type*. E.g.,

$$Person := \times (name \mapsto \mathbb{A}^* \cup age \mapsto \mathbb{N}),$$

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

- Syntactic sugar:

$$\text{record} : \text{fam} (\text{fam } T) \rightarrow \mathcal{P} \mathcal{F} \quad \text{with} \quad \text{record } F = \times (\cup F)$$

Now we can write

$$Person := \text{record} (name \mapsto \mathbb{A}^*, age \mapsto \mathbb{N})$$

### c. Relational databases in functional style

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
...	...	...	

Relational database presents the tables as relations.

- **Traditional view:** rows as tuples (and tuples not seen as functions).  
Problem: access only by separate indexing function using numbers.  
Patch: “grafting” *attribute names* for column headings.  
Disadvantages: model not purely relational, operators on tables ad hoc.
- **Functional view;** the table rows as *records* using record  $F = \times (\bigcup F)$   
Advantage: embedding in general framework, inheriting algebraic properties and generic operators, record field labels supply table headings.

Relational databases as sets of functions using record  $F = \times (\cup F)$

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  $GCI : \mathcal{P} CID$ , a set of *Course Information Descriptors* with

```
def CID := record (code ↦ Code, name ↦  $\mathbb{A}^*$ , inst ↦ Staff, prrq ↦ Code*)
```

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.

## 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  $R$ .

- **Selection ( $\sigma$ )** selects in any table  $S : \mathcal{P} R$  those records satisfying  $P : R \rightarrow \mathbb{B}$ .

Solution: set filtering  $\sigma(S, P) = S \downarrow P$ .

Example:  $GCI \downarrow (r : CID . r \text{ code} = \text{CS300})$

selects the row for question (a), “Who is the instructor for CS300?”.

- **Projection ( $\pi$ )** yields in any  $S : \mathcal{P} R$  columns with field names in a set  $F$ .

Solution: restriction  $\pi(S, F) = \{r \upharpoonright F \mid r : S\}$ .

Example:  $\pi(GCI, \{\text{code}, \text{inst}\})$  selects the columns for question (a)

whereas  $\pi(GCI \downarrow (r : CID . r \text{ code} = \text{CS300}), \text{inst})$  reflects all of (a).

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

Solution:  $S \bowtie T = \{s \cup t \mid (s, t) : S \times T \wedge s \odot t\}$  (function type merge!)

Example:  $GCI \bowtie CS$  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 remark    Note that  $S \bowtie T = S \otimes T$     One can show

$$\odot (f, g, h) \Rightarrow (f \cup g) \cup h = f \cup (g \cup h)$$

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

Importance: this is all about generic functionals, no ad hoc database theories!

## Next topic

- 09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms
- Part I — Unifying the mathematics of SP and FM**
- 09:20–09:40 1. Review of the basics and outline of the formalism used
- 09:40–10:10 2. Inspiration from SP to FM: functions and functionals
- 10:10–10:30 3. Calculation with predicates and quantifiers for engineers
- 10:30–11:00 (Half-hour break)
- Part II — Application examples to modeling in SP and FM**
- 11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP
- 11:40–11:45 5. Brief intermezzo about endosemantic functions
- 11:45–12:05 6. Modeling programs by program equations
- 12:05–12:25 7. Reasoning about temporal behavior by temporal operators
- 12:25–12:30 8. Final considerations

## 5 Brief intermezzo about endosemantic functions

Purpose: salvaging certain “ad hoc” conventions by providing formal justification.  
Of course: desirable only if the convention has potential merit (hidden structure)

- **SP example** Recall: “*not*  $\mathcal{F}\{f(t)\}$  but  $\mathcal{F}f\omega$ ”. Reason: functionally clean

$$\mathcal{F}f\omega = \int_{-\infty}^{+\infty} e^{-j\omega t} \cdot f(t) \cdot dt \quad \text{and} \quad \mathcal{F}'g\omega = \frac{1}{2\pi} \cdot \int_{-\infty}^{+\infty} e^{j\omega t} \cdot g(t) \cdot dt$$

Leftmost formula:  $\omega$  free both sides,  $t$  bound. Rightmost formula: reverse

By contrast, the common formulation is functionally “dirty” (nonsensical)

$$\mathcal{F}\{f(t)\} = \int_{-\infty}^{+\infty} e^{-j\omega t} \cdot f(t) \cdot dt \quad \mathcal{F}'\{F(\omega)\} = \frac{1}{2\pi} \cdot \int_{-\infty}^{+\infty} e^{j\omega t} \cdot F(\omega) \cdot d\omega$$

- Endosemantic functions: mapping syntactic objects into the target language

Idea:  $f e = e'$  takes  $e$  as a *syntactic* argument and maps it to  $e'$

Image taken mathematically or syntactically depending on context.

Substitution  $[v = d]$ , as in  $e[v = d]$  is already the most basic example.

We can now justify writing things like  $\mathcal{F}\{e^{-a \cdot t} \cdot u(t)\} = 1/(j \cdot \omega + a)$

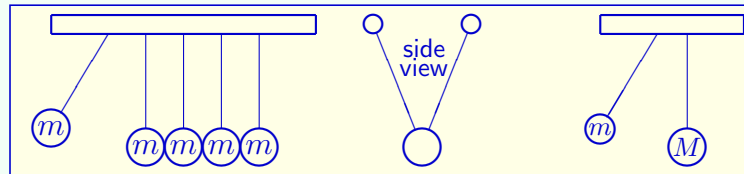
Tacit view in most program semantics, e.g.,  $\text{wa}(x := x + 1)(x > 3) \equiv x > 2$ .

## Next topic

- 09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms
- Part I — Unifying the mathematics of SP and FM**
- 09:20–09:40 1. Review of the basics and outline of the formalism used
- 09:40–10:10 2. Inspiration from SP to FM: functions and functionals
- 10:10–10:30 3. Calculation with predicates and quantifiers for engineers
- 10:30–11:00 (Half-hour break)
- Part II — Application examples to modeling in SP and FM**
- 11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP
- 11:40–11:45 5. Brief intermezzo about endosemantic functions
- 11:45–12:05 6. Modeling programs by program equations
- 12:05–12:25 7. Reasoning about temporal behavior by temporal operators
- 12:25–12:30 8. Final considerations

## 6 Modeling programs by program equations

### 6.0 An analogy from mechanics: colliding balls ("Newton's Cradle")



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

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

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

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

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

## 6.1 Program equations

Given here for a simple yet complete language (Dijkstra's guarded commands)  
 Interesting aspect: nondeterminism.

Let  $\mathbb{S}$  be the program state space (values of the variables).

State changes are expressed by  $R : C \rightarrow \mathbb{S}^2 \rightarrow \mathbb{B}$ , termination by  $T : C \rightarrow \mathbb{S} \rightarrow \mathbb{B}$ .

Syntax Command $c$	Behavior (program equations or equivalent program)	
	State change $R_c(s, s')$	Termination $T_c s$
$v := e$	$s' = s[e^v]$	1
skip	$s' = s$	1
abort	0	0
$c'; c''$	$\exists t. R_{c'}(s, t) \wedge R_{c''}(t, s')$	$T_{c'} s \wedge \forall t. R_{c'}(s, t) \Rightarrow T_{c''} t$
if $\square i: I. b_i \rightarrow c'_i$ fi	$\exists i: I. b_i \wedge R_{c'_i}(s, s')$	$\exists b \wedge \forall i: I. b_i \Rightarrow T_{c'_i} s$
do $b \rightarrow c'$ od	if $\neg b \rightarrow \text{skip} \square b \rightarrow (c'; c)$ fi	

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

## 6.2 Calculational semantics unifying theories of programming

**Program theories** *by predicate calculus only, no special logics!*

- General/unifying: see “Calculational semantics” paper
- Examples here: ante/post semantics, Hoare style and Dijkstra style

### A style issue

(0) Functional style: with predicates in  $\mathbb{S} \rightarrow \mathbb{B}$  (functionally clean,  $\mathcal{F} f \omega$  style)

$$\begin{array}{ll} \{A\} c \{P\} & \equiv \quad \forall (s, s') : (\mathbb{S}^2 \downarrow R_c) . A s \Rightarrow P s' & \text{“partial correctness”} \\ [A] c [P] & \equiv \quad \{A\} c \{P\} \wedge \text{Term}_c A & \text{“total correctness”} \\ \text{Term}_c A & \equiv \quad \forall s . A s \Rightarrow T_c s & \text{“termination”} \end{array}$$

(1) Expression style: with propositions (& endosemantic functions,  $\mathcal{F} \{f(t)\}$  style)

$$\begin{array}{ll} \{a\} c \{p\} & \equiv \quad \forall (s, s') : (\mathbb{S}^2 \downarrow R_c) . a s \Rightarrow p' & \text{“partial correctness”} \\ [a] c [p] & \equiv \quad \{a\} c \{p\} \wedge \text{Term}_c a & \text{“total correctness”} \\ \text{Term}_c a & \equiv \quad \forall s . a s \Rightarrow T_c s & \text{“termination”} \end{array}$$

Here we use (0) for illustration; (1) is elaborated in detail in the cited paper.

### 6.3 Calculating all properties of interest

Example: weakest antecondition semantics (Dijkstra style). Definitions:

- *Weakest liberal antecondition*: weakest  $A$  satisfying  $\{A\} c \{P\}$
- *Weakest antecondition*: weakest  $A$  satisfying  $[A] c [P]$

Calculational derivation of an expression for such antecondx: push  $A$  out

$$\begin{aligned}
 \{A\} c \{P\} &\equiv \langle \text{Def. } \{-\} - \{-\} \rangle \quad \forall (s, s') : (\mathbb{S}^2 \downarrow R_c) . A s \Rightarrow P s' \\
 &\equiv \langle \text{Def. } \downarrow, \text{ trading} \rangle \quad \forall (s, s') : \mathbb{S}^2 . R_c(s, s') \Rightarrow A s \Rightarrow P s' \\
 &\equiv \langle \text{Nest, shunt} \rangle \quad \forall s . \forall s' . A s \Rightarrow R_c(s, s') \Rightarrow P s' \\
 &\equiv \langle \text{Ldist. } \Rightarrow / \forall \rangle \quad \forall s . A s \Rightarrow \forall s' . R_c(s, s') \Rightarrow P s'
 \end{aligned}$$

This justifies the first definition in

$$\begin{aligned}
 \text{def } \text{wla} : C \rightarrow \text{pred}_{\mathbb{S}} \rightarrow \text{pred}_{\mathbb{S}} \text{ with } \text{wla } c P s &\equiv \forall s' . R_c(s, s') \Rightarrow P s' \\
 \text{def } \text{wa} : C \rightarrow \text{pred}_{\mathbb{S}} \rightarrow \text{pred}_{\mathbb{S}} \text{ with } \text{wa } c P s &\equiv \text{wla } c P s \wedge T_c s
 \end{aligned}$$

The second definition is justified by  $[A] c [P] \equiv \{A\} c \{P\} \wedge \text{Term}_c A$  in which

$$\text{Term}_c A \equiv \forall s . A s \Rightarrow T_c s$$

Compare the rules and the style of reasoning to earlier calculations, for instance,

closed $P$	
$\equiv$	$\langle \text{Definit. closed} \rangle$ open $(\neg P)$
$\equiv$	$\langle \text{Definit. open} \rangle$ $\forall v : \mathbb{R}_{\neg P} . \exists \epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} .  x - v  < \epsilon \Rightarrow \neg P x$
$\equiv$	$\langle \text{Trading sub } \forall \rangle$ $\forall v : \mathbb{R} . \neg P v \Rightarrow \exists \epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} .  x - v  < \epsilon \Rightarrow \neg P x$
$\equiv$	$\langle \text{Contraposit.} \rangle$ $\forall v : \mathbb{R} . \neg \exists (\epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . P x \Rightarrow \neg ( x - v  < \epsilon)) \Rightarrow P v$
$\equiv$	$\langle \text{Duality, twice} \rangle$ $\forall v : \mathbb{R} . \forall (\epsilon : \mathbb{R}_{>0} . \exists x : \mathbb{R} . P x \wedge  x - v  < \epsilon) \Rightarrow P v$
$\equiv$	$\langle \text{Definition ad} \rangle$ $\forall v : \mathbb{R} . \text{ad } P v \Rightarrow P v$
$\equiv$	$\langle P v \Rightarrow \text{ad } P v \rangle$ $\forall v : \mathbb{R} . \text{ad } P v \equiv P v$

$\forall (x, y) : (A^*)^2 . x \leq y \Rightarrow s x \leq s y$	
$\equiv$	$\langle \text{Definit. } \leq \rangle$ $\forall (x, y) : (A^*)^2 . \exists (z : A^* . y = x ++ z) \Rightarrow \exists (u : B^* . s y = s x ++ u)$
$\equiv$	$\langle \text{Rdst } \Rightarrow / \exists \rangle$ $\forall (x, y) : (A^*)^2 . \forall (z : A^* . y = x ++ z) \Rightarrow \exists u : B^* . s y = s x ++ u$
$\equiv$	$\langle \text{Nest, swp} \rangle$ $\forall x : A^* . \forall z : A^* . \forall (y : A^* . y = x ++ z) \Rightarrow \exists u : B^* . s y = s x ++ u$
$\equiv$	$\langle \text{1-pt, nest} \rangle$ $\forall (x, z) : (A^*)^2 . \exists u : B^* . s (x ++ z) = s x ++ u$
$\equiv$	$\langle \text{Compreh.} \rangle$ $\exists r : (A^*)^2 \rightarrow B^* . \forall (x, z) : (A^*)^2 . s (x ++ z) = s x ++ r (x, z)$

Rules and style are IDENTICAL, even for totally different subjects!

Conclusion: (functional) predicate calculus is as important for future SP and FM engineers as differential and integral calculus is for “classical” engineers.

## 6.4 Results and more analogies

- From the preceding, we obtain by functional predicate calculus:

$$\begin{aligned}
 \text{wa } \llbracket v := e \rrbracket P s &\equiv P (s[e^v]) \\
 \text{wa } \llbracket c' ; c'' \rrbracket &\equiv \text{wa } c' \circ \text{wa } c'' \\
 \text{wa } \llbracket \text{if } \prod i : I . b_i \rightarrow c'_i \text{ fi} \rrbracket P s &\equiv \exists b \wedge \forall i : I . b_i \Rightarrow \text{wa } c'_i P s \\
 \text{wa } \llbracket \text{do } b \rightarrow c' \text{ od} \rrbracket P s &\equiv \exists n : \mathbb{N} . w^n (\neg b \wedge P s) \text{ defining } w \text{ by} \\
 w q &\equiv (\neg b \wedge P s) \vee (b \wedge \text{wa } c' (s \bullet q) s)
 \end{aligned}$$

Syntactic shortcut used:  $s = \text{tuple of all program variables}$ .

- Remark: practical rules for loops (invariants, bound functions) similarly
- Analogies: Green functions (for linear device  $d$ ), Fourier transforms

$$\begin{aligned}
 \text{wla } c P s &\equiv \forall s' : \mathbb{S} . \mathbb{R}_c (s, s') \Rightarrow P s' \\
 \text{Rsp } d f x &= \int x' : \mathbb{R} . \mathbb{G} d (x, x') \cdot f x' \quad (\text{linear } d) \\
 \text{Rsp } d f t &= \int t' : \mathbb{R} . \mathbb{h} d (t - t') \cdot f t' \quad (\text{for LTI } d) \\
 \mathcal{F} f \omega &= \int t : \mathbb{R} . \exp(-j \cdot \omega \cdot t) \cdot f t
 \end{aligned}$$

## Next topic

- 09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms
- Part I — Unifying the mathematics of SP and FM**
- 09:20–09:40 1. Review of the basics and outline of the formalism used
- 09:40–10:10 2. Inspiration from SP to FM: functions and functionals
- 10:10–10:30 3. Calculation with predicates and quantifiers for engineers
- 10:30–11:00 (Half-hour break)
- Part II — Application examples to modeling in SP and FM**
- 11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP
- 11:40–11:45 5. Brief intermezzo about endosemantic functions
- 11:45–12:05 6. Modeling programs by program equations
- 12:05–12:25 7. Reasoning about temporal behavior by temporal operators
- 12:25–12:30 8. Final considerations

## 7 Reasoning about temporal behavior by temporal operators

### 7.0 Defining temporal operators — Functional style

#### a. Conventions for the Functional Temporal Calculus

State space  $\mathbb{S}$  (instantaneous values).

*Behaviors*: infinitestate sequences; functions of type  $\mathbb{N} \rightarrow \mathbb{S}$ , also written  $\mathbb{S}^\infty$ .

The predicates of interest are of type  $\text{BP} := \mathbb{S}^\infty \rightarrow \mathbb{B}$ .

#### b. Operators

Logical operators of FTC are implicit direct extensions of the propositional operators: for any infix operator  $\star$  (say,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\equiv$ ) and any  $\beta$  in  $\mathbb{S}^\infty$ ,

$$(P \star Q) \beta \equiv P \beta \star Q \beta .$$

Temporal operators of FTC are predicate transformers of type  $\text{BP} \rightarrow \text{BP}$ , e.g.,

- $\circ$  (“next”) defined by  $\circ P \beta \equiv P(\sigma \beta)$
- $\square$  (“henceforth”) defined by  $\square P \beta \equiv \forall n : \mathbb{N} . P(\sigma^n \beta)$
- $\diamond$  (“eventually”) defined by  $\diamond P \beta \equiv \exists n : \mathbb{N} . P(\sigma^n \beta)$

## 7.1 Illustration: deriving point-free theorems in FTC

Convention: for predicates  $P$  in BP, we define the *validity operator*  $\vdash$  by

$$\vdash P \equiv \forall P$$

Equivalently, in pointwise form:  $\vdash P \equiv \forall \beta : \mathbb{S}^\infty . P \beta$ .

Here are some example derivations of theorems useful for calculational temporal reasoning.

a. **Example A** Showing  $\vdash (\Box P) \equiv \vdash P$

$$\begin{aligned} \vdash (\Box P) &\equiv \langle \text{Def. } \vdash, \Box \rangle \forall \beta : \mathbb{S}^\infty . \forall n : \mathbb{N} . P (\sigma^n \beta) \\ &\equiv \langle \text{Nesting} \rangle \forall \beta, n : \mathbb{S}^\infty \times \mathbb{N} . P (\sigma^n \beta) \\ &\equiv \langle \text{Dom. ch.} \rangle \forall \beta : \mathbb{S}^\infty . P \beta \\ &\equiv \langle \text{Defin. } \vdash \rangle \vdash P \end{aligned}$$

Note: the “domain change” uses the fact that the function

$$f : \mathbb{S}^\infty \times \mathbb{N} \rightarrow \mathbb{S}^\infty \text{ with } f(\beta, n) = \sigma^n \beta$$

satisfies  $\mathcal{R} f = \mathbb{S}^\infty$  (trivial since  $f(\beta, 0) = \beta$ ).

b. **Example B** “temporal instantiation”:  $\boxed{\vdash (\Box P \Rightarrow P)}$

$$\begin{aligned}
 \vdash (\Box P \Rightarrow P) &\equiv \langle \text{Definition } \vdash \rangle \quad \forall \beta : \mathbb{S}^\infty . (\Box P \Rightarrow P) \beta \\
 &\equiv \langle \text{Implicit ext.} \rangle \quad \forall \beta : \mathbb{S}^\infty . \Box P \beta \Rightarrow P \beta \\
 &\equiv \langle \text{Definition } \Box \rangle \quad \forall \beta : \mathbb{S}^\infty . \forall (n : \mathbb{N} . P (\sigma^n \beta)) \Rightarrow P \beta \\
 &\equiv \langle f^0 x = x \rangle \quad \forall \beta : \mathbb{S}^\infty . \forall (n : \mathbb{N} . P (\sigma^n \beta)) \Rightarrow P (\sigma^0 \beta) \\
 &\equiv \langle \text{Instant., } n := 0 \rangle \quad \forall \beta : \mathbb{S}^\infty . 1 \\
 &\equiv \langle \text{Const. pred.} \rangle \quad 1 .
 \end{aligned}$$

Remarks

- Converse ( $P \Rightarrow \Box P$ ) is not valid (try  $\beta n = n$  and  $P \beta \equiv \beta 0 = 0$ ).
- Proof compacting: proving  $\vdash P$  is proving  $P \beta$  for arbitrary  $\beta$ , e.g.,

$$\begin{aligned}
 \Box P \beta &\equiv \langle \text{Definition } \Box \rangle \quad \forall n : \mathbb{N} . P (\sigma^n \beta) \\
 &\Rightarrow \langle \text{Inst. } n := 0 \rangle \quad P (\sigma^0) \\
 &\equiv \langle \text{Definition } f^n \rangle \quad P \beta ,
 \end{aligned}$$

hence  $\Box P \beta \Rightarrow P \beta$  and, by direct extension,  $(\Box P \Rightarrow P) \beta$ .

c. Example C: temporal induction  $\boxed{\vdash (\Box(P \Rightarrow \bigcirc P) \equiv \Box(P \Rightarrow \Box P))}$  (STI)

Interest: proof uses a nice generalization of *weak induction* over  $\mathbb{N}$  (WIN), called *strengthened weak induction* (SWIN). For comparison:

$$\forall (n:\mathbb{N}. Q n \Rightarrow Q (n+1)) \Rightarrow Q 0 \Rightarrow \forall m:\mathbb{N}. Q m \quad (\text{WIN})$$

$$\forall (n:\mathbb{N}. Q n \Rightarrow Q (n+1)) \equiv \forall n:\mathbb{N}. Q n \Rightarrow \forall m:\mathbb{N}. Q (n+m) \quad (\text{SWIN})$$

Deriving (STI) from (SWIN) is a straightforward exercise.

d. Example D, “infinitely often”  $\boxed{\Box(\Diamond P)\beta \equiv \exists_{\infty} n:\mathbb{N}. P(\sigma^n\beta)}$

- Use: in specifications,  $\Box\Diamond\varphi$  is typically used to express that formula  $\varphi$  is satisfied “infinitely often”. The functional counterpart is  $\Box(\Diamond P)$ .
- Intuitive interpretation is plausible, but we still must relate it to the usual mathematical characterization of infinity. Three steps:

- Finiteness: to express that a (general) predicate  $Q$  is satisfied for finitely many argument values,

DEFINITION:  $\text{Fin } Q \equiv \exists n:\mathbb{N}.\exists f:\mathbb{N}_{<n}\rightarrow\mathcal{D}Q.(\mathcal{D}Q)_Q\subseteq\mathcal{R}f$  .

- Infiniteness (negation of finiteness): define, for any predicate  $Q$ ,

DEFINITION:  $\exists_{\infty} Q \equiv \neg(\text{Fin } Q)$  .

- Specializing to predicates on natural numbers ( $\mathcal{D}Q = \mathbb{N}$ )

THEOREM:  $\exists_{\infty} Q \equiv \forall n:\mathbb{N}.\exists m:\mathbb{N}. Q(m+n)$

The proofs are instructive exercises.

e. Example E, distributivity(-like) properties

An important batch is

Dist. $\Box/\wedge$ :	$\Box(P \wedge Q) \equiv \Box P \wedge \Box Q$	Dual:	$\Diamond(P \vee Q) \equiv \Diamond P \vee \Diamond Q$
Disp. $\Diamond/\wedge$ :	$\Diamond(P \wedge Q) \Rightarrow \Diamond P \wedge \Diamond Q$	Dual:	$\Box(P \vee Q) \Leftarrow \Box P \vee \Box Q$
Equipred.:	$\Box(P \equiv Q) \Rightarrow (\Box P \equiv \Box Q)$	Also:	$\Box(P \equiv Q) \Rightarrow (\Diamond P \equiv \Diamond Q)$
Wkrpred.:	$\Box(P \Rightarrow Q) \Rightarrow \Box P \Rightarrow \Box Q$	Also:	$\Box(P \Rightarrow Q) \Rightarrow \Diamond P \Rightarrow \Diamond Q$

Match certain properties in functional predicate calculus. Noteworthy addition:

$$\Box_{\infty}(P \vee Q) \equiv \Box_{\infty}P \vee \Box_{\infty}Q$$

for general predicates  $P$  and  $Q$  satisfying  $\mathcal{D}P = \mathcal{D}Q$ . For temporal predicates:

$$\Box(\Diamond(P \vee Q)) \equiv \Box(\Diamond P) \vee \Box(\Diamond Q) \quad \text{dual:} \quad \Diamond(\Box(P \wedge Q)) \equiv \Diamond(\Box P) \wedge \Diamond(\Box Q)$$

## Two important observations

- FTC is entirely formulated within functional predicate calculus, without a separate temporal logic language. The operators are predicate transformers.
- FTC captures the essence of temporal logic, and can thereby serve as an archetype for studying other temporal logics, an issue addressed next. Derivations & results from FTC “carried over”, replacing  $P\beta$  by  $\beta \models \varphi$ . Most crucial: FTC “imports” calculational reasoning in temporal logics.

## 7.2 Role of FTC in understanding and using other temporal logics

Chosen example: TLA<sup>+</sup> [Leslie Lamport, *Specifying Systems*]

- a. **Conventions** State space  $\mathbb{S}$  (system variables), behaviors have type  $\mathbb{S}^\infty$ .

Arithmetic, relational and logical operators assumed available. Taxonomy:

$\mathcal{E}$  state expressions       $\mathcal{B}$  state propositions

$\mathcal{E}'$  transition expressions     $\mathcal{A}$  transition propositions, called “actions”

$\mathcal{X}$  temporal expressions     $\mathcal{F}$  temporal propositions (“temporal formulas”)

State expressions: variables unprimed. Transition expressions: also primed.

Temporal expressions: with temporal operators ( $\mathcal{E} \subset \mathcal{E}' \subset \mathcal{X}$ ).

Conventions for tuple  $s$  of state variables etc. as for calculational semantics.

## b. Action operators

$— \cdot — : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$	$a \cdot b \equiv \exists t : \mathbb{S} . a[t^{s'} \wedge b[t^s]$
$[—]_— : \mathcal{A} \times \mathcal{E} \rightarrow \mathcal{A}$	$[a]_e \equiv a \vee e = e'$
$\langle — \rangle_— : \mathcal{A} \times \mathcal{E} \rightarrow \mathcal{A}$	$\langle a \rangle_e \equiv a \wedge e \neq e'$
UNCHANGED : $\mathcal{E} \rightarrow \mathcal{A}$	UNCHANGED $e \equiv e = e'$
ENABLED : $\mathcal{A} \rightarrow \mathcal{B}$	ENABLED $a \equiv \exists s' : \mathbb{S} . a$

Legend (as before):

- $\mathcal{E}$  state expressions       $\mathcal{B}$  state propositions
- $\mathcal{E}'$  transition expressions     $\mathcal{A}$  transition propositions, called “actions”
- $\mathcal{X}$  temporal expressions     $\mathcal{F}$  temporal propositions (“temporal formulas”)

### c. Temporal operators

Characterized by endosemantic function  $\models$ , defined recursively ( $\mathcal{E} \subset \mathcal{E}' \subset \mathcal{X}$ ).

Basis: state and transition expressions  $e$  in  $\mathcal{E}$  and  $\mathcal{E}'$ , for which

$$\beta \models e = e_{\beta 0, \beta 1}^{[s, s']}. .$$

For temporal expressions (in  $\mathcal{X}$ ) and formulas (in  $\mathcal{F}$ ),

$\begin{aligned} \circ : \mathcal{X} &\rightarrow \mathcal{X} & \beta \models \circ e &= \sigma \beta \models e & (\circ \text{ does not appear in TLA}^+) \\ \square : \mathcal{F} &\rightarrow \mathcal{F} & \beta \models \square \varphi &\equiv \forall n : \mathbb{N}. \sigma^n \beta \models \varphi \\ \diamond : \mathcal{F} &\rightarrow \mathcal{F} & \beta \models \diamond \varphi &\equiv \exists n : \mathbb{N}. \sigma^n \beta \models \varphi \\ \forall_- : \mathcal{V} &\rightarrow \mathcal{F} \rightarrow \mathcal{F} & \beta \models \forall_v \varphi &\equiv \forall \gamma : \mathbb{S}_{\models \varphi}^\infty. (\natural \gamma)_{\neq i}^T = (\natural \beta)_{\neq i}^T \text{ where } i = \underline{s}^- v \\ \exists_- : \mathcal{V} &\rightarrow \mathcal{F} \rightarrow \mathcal{F} & \beta \models \exists_v \varphi &\equiv \exists \gamma : \mathbb{S}_{\models \varphi}^\infty. (\natural \gamma)_{\neq i}^T = (\natural \beta)_{\neq i}^T \text{ where } i = \underline{s}^- v \end{aligned}$
---

Temporal quantifiers  $\forall$  and  $\exists$  for completeness only.

The *compacting operator*  $\natural$  removes successive duplicates (stuttering),

$$\natural \beta = ++ n : \mathcal{D} \beta. (n > 0 \wedge \beta(n-1) = \beta n) ? \varepsilon \dagger \tau(\beta n) ,$$

d. Boolean combinations of temporal formulas are defined by distributivity:

$$\begin{array}{ll} \beta \models \neg \varphi \equiv \neg (\beta \models \varphi) & \beta \models \forall (x: X . \varphi) \equiv \forall x: X . \beta \models \varphi \\ \beta \models (\varphi \star \psi) \equiv \beta \models \varphi \star \beta \models \psi & \beta \models \exists (x: X . \varphi) \equiv \exists x: X . \beta \models \varphi \end{array}$$

Here  $\star$  is any infix logical operator in  $\{\Rightarrow, \equiv, \neq, \oplus, \wedge, \vee\}$ .

Remarks

- To the left of  $\equiv$ , operators  $\star$ ,  $\neg$ ,  $\forall$ ,  $\exists$  are TLA<sup>+</sup>; to the right, just logic.
- Because temporal formulas appear only syntactically, we adopt the syntax of the target language, e.g., optional parentheses:  $\square \diamond \varphi$  stands for  $\square (\diamond \varphi)$ .

**General observation** “Proof of the pudding” is not mere survival, but taste.

Here this means: elegant calculational reasoning.

Experience: approach has replaced semi-formal traditional proofs by formal calculational derivations that are shorter, clearer (structure), and don't skip steps.

## Next topic

09:00–09:20 0. Motivation: synergy FM-SP; value of defect-free formalisms

### **Part I — Unifying the mathematics of SP and FM**

09:20–09:40 1. Review of the basics and outline of the formalism used

09:40–10:10 2. Inspiration from SP to FM: functions and functionals

10:10–10:30 3. Calculation with predicates and quantifiers for engineers

10:30–11:00 (Half-hour break)

### **Part II — Application examples to modeling in SP and FM**

11:00–11:40 4. Predicate calculus and generic functionals applied to classical SP

11:40–11:45 5. Brief intermezzo about endosemantic functions

11:45–12:05 6. Modeling programs by program equations

12:05–12:25 7. Reasoning about temporal behavior by temporal operators

12:25–12:30 8. Final considerations

## 8 **Final considerations**

- What we have shown

- Notational and methodological unification of CS and classical engineering
- Unification of system modeling and reasoning styles in SP and FM
- Unification also extends to a large part of mathematics

- Ramifications

Generic functionals and (functional) predicate calculus are as important for future SP and FM engineers as differential and integral calculus are for “classical” engineers.