

## 0. The Issues

- a. (D)SP ISSUE: **Declarativity over Algorithms**
- Advantages:
    - More compact, clearer (no extraneous detail)
    - Problem-oriented (algorithm = implementation)
    - Crucial*: best suited to reasoning & calculation
  - Sloppy nomenclature (“smurf”) is not harmless:
    - E.g., passing off algorithms as “specifications”
    - ⇒ loss of opportunities (in modeling, design etc.)
- b. BROADER ISSUE: **Unified Basis for Engineering**
- Unifying classical Engineering with CS (→ **ECE**):
    - Similar models for circuits and programs
  - Unified basis for “continuous” and “discrete” math

## 1. A formalism, part I: the language

Four constructs suffice due to *functional basis*.

0. Identifier: almost anything (a lexical technicality)  
 Introduced by a *binding*:  $i : S \wedge p$ . Legend:  
 new identifier(s)  $i$ , set  $S$ , (optional) proposition  $p$   
*Example*:  $n : \mathbb{N}$  is the same as  $n : \mathbb{Z} \wedge 0 \leq n$
1. Function application:  $f e$  (or  $f(e)$  traditionally)  
 Other forms declarable in binding (e.g.,  $— \star —$ )
2. Function abstraction:  $binding . expr$  Axioms:  
 Domain:  $v \in \mathcal{D}(v : S \wedge p . e) \equiv v \in S \wedge p$   
 Map:  $v \in \mathcal{D}(v : S \wedge p . e) \Rightarrow (v : S \wedge p . e) v = e$
3. Tupling:  $e, e', e''$  Domain:  $\mathcal{D}(e, e', e'') = \{0, 1, 2\}$   
 Map:  $(e, e', e'') 0 = e$  and  $(e, e', e'') 1 = e'$  etc.  
**Synthesizes all familiar forms without their defects.**

## 2. A formalism, part II: the rules

- a. Main aspect: support *calculational* reasoning

$$\begin{array}{l} expression \ R \ \langle \text{justification} \rangle \ expression' \\ R' \ \langle \text{justification}' \rangle \ expression'' \end{array}$$

$R, R'$  transitive, e.g., arithmetic  $=, \leq$ , logic  $\equiv, \Rightarrow$ .  
 Common in (applications of) algebra, calculus etc.  
 Lacking in *logical* arguments with *syncopation*, i.e.,  
 using symbols ( $\forall, \exists$ ) as mere notation, w/o rules

- b. Two main elements of the formalism

0. (Concrete) Generic Functionals  
 Generalizes function composition, inverse etc.
1. Functional Predicate Calculus: practical rules  
 Allows engineers to calculate smoothly with  $\forall, \exists$ .  
 A necessity in CS, a valuable opportunity in EE.

## 3. Illustration: a few typical rules

- a. **Generic Functionals**: filtering ( $\downarrow$ ), extension ( $\hat{\ }^{\ }^{\ }$ )

$$\begin{array}{l} \mathcal{D}(f \downarrow P) = \{x : \mathcal{D} f \cap \mathcal{D} P \mid P x\} \quad (f \downarrow P) x = f x \\ \mathcal{D}(f \hat{\ }^{\ }^{\ } g) = \{x : \mathcal{D} f \cap \mathcal{D} g \mid (f x, g x) \in \mathcal{D}(\star)\} \quad (f \hat{\ }^{\ }^{\ } g) x = f x \star g x \end{array}$$

- b. **Functional Predicate Calculus**: a few rules for  $\forall$

Axiom:  $\forall P \equiv P = \mathcal{D} P \bullet 1$  Typical derived rules:

Rule name	Point-free form	Pointwise form example: if variable $v$ not in $q$ ,
L-dst. $\Rightarrow \forall$	$q \Rightarrow \forall P \equiv \forall (q \overset{\leftarrow}{\Rightarrow} P)$	$\exists (v : X . p) \Rightarrow q$
R-dst. $\Rightarrow \exists$	$\exists P \Rightarrow q \equiv \forall (P \overset{\leftarrow}{\Rightarrow} q)$	$\equiv \forall (v : X . p \Rightarrow q)$
1-pnt. rule	$\forall P_{=e} \equiv e \in \mathcal{D} P \Rightarrow P e$	Note: $f_P$ stands for $f \downarrow P$
Trading	$\forall P_Q \equiv \forall (Q \Leftrightarrow P)$	

## 4. Examples I: Mathematical Analysis

Illustrates how calculation replaces syncopation

From the definitions (see paper), show by calculation closed  $P \equiv \text{ad } P = P$ .

$$\begin{array}{l} \text{closed } P \equiv \langle \text{“closed”} \rangle \text{ open } (\neg P) \\ \equiv \langle \text{“open”} \rangle \forall v : \mathbb{R}_{=P} . \exists \epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . |x - v| < \epsilon \Rightarrow \neg P x \\ \equiv \langle \text{Trdg. } \forall \rangle \forall v : \mathbb{R} . \neg P v \Rightarrow \exists \epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . |x - v| < \epsilon \Rightarrow \neg P x \\ \equiv \langle \text{Ctrps.} \rangle \forall v : \mathbb{R} . \neg \exists (\epsilon : \mathbb{R}_{>0} . \forall x : \mathbb{R} . P x \Rightarrow \neg (|x - v| < \epsilon)) \Rightarrow P v \\ \equiv \langle \text{Duality} \rangle \forall v : \mathbb{R} . \forall (\epsilon : \mathbb{R}_{>0} . \exists x : \mathbb{R} . P x \wedge |x - v| < \epsilon) \Rightarrow P v \\ \equiv \langle \text{Def. ad} \rangle \forall v : \mathbb{R} . \text{ad } P v \Rightarrow P v \\ \equiv \langle \text{Lemma} \rangle \forall v : \mathbb{R} . \text{ad } P v \equiv P v \quad \text{Lemma: } P v \Rightarrow \text{ad } P v \text{ (simple)} \end{array}$$

## 5. Examples II: Signals and Systems

Illustrates calculation with [generic] functionals

Calculate the response of an LTI system  $s$  (see paper) to  $E_c$  where  $E_c t = e^{ct}$ .

$$\begin{array}{l} s E_c(t + \tau) \equiv \langle \text{Definition } \sigma \rangle \sigma_\tau (s E_c) t \\ \equiv \langle \text{Time inv. } s \rangle s (\sigma_\tau E_c) t \\ \equiv \langle \text{Property } E_c \rangle s (E_c \tau \overset{\leftarrow}{\cdot} E_c) t \\ \equiv \langle \text{Linearity } s \rangle (E_c \tau \overset{\leftarrow}{\cdot} s E_c) t \\ \equiv \langle \text{Definition } \overset{\leftarrow}{\cdot} \rangle E_c \tau \cdot s E_c t \quad \text{Hence } s E_c = E_c \overset{\leftarrow}{\cdot} s E_c 0 \end{array}$$

## 6. Examples III: Program Dynamics

Illustrates how the same formalism covers programs

0. Describing program behaviour by program equations

State space  $S$ ; set of commands  $C$ ; define  $R_-$  and  $T_-$  (type  $C \rightarrow S^2 \rightarrow \text{IB}$ )

Command $c$	State change $R_c(s, s')$	Termination $T_c s$
$v := e$	$s' = s \overset{v}{\leftarrow}$	1
$c'; c''$	$\exists s . R_{c'}(s, s) \wedge R_{c''}(s, s')$	$T_{c'} s \wedge \forall s . R_{c'}(s, s) \Rightarrow T_{c''} s$
if $\parallel i : I . b_i \rightarrow c'_i$ fi	$\exists i : I . b_i \wedge R_{c'_i}(s, s')$	$\exists (i : I . b_i) \wedge \forall i : I . b_i \Rightarrow T_{c'_i} s$
do $b \rightarrow c'$ od	effect: if $\neg b \rightarrow \text{skip} \parallel b \rightarrow (c' ; \text{do } b \rightarrow c' \text{ od})$ fi	

1. Hoare semantics:  $\{A\} c \{P\} \equiv \forall s . \forall s' . R_c(s, s') \Rightarrow A s \Rightarrow P s'$

2. Calculating Dijkstra semantics from Hoare semantics

$Wla : C \rightarrow (S \rightarrow \text{IB}) \rightarrow (S \rightarrow \text{IB})$  with  $\{A\} c \{P\} \equiv \forall s . A s \Rightarrow Wla_c P s$

$$\begin{array}{l} \{A\} c \{P\} \equiv \langle \text{Definition} \rangle \forall s . \forall s' . R_c(s, s') \Rightarrow A s \Rightarrow P s' \\ \equiv \langle \text{Shunting } \Rightarrow \rangle \forall s . \forall s' . A s \Rightarrow R_c(s, s') \Rightarrow P s' \\ \equiv \langle \text{Ldistr. } \Rightarrow \forall \rangle \forall s . A s \Rightarrow \forall s' . R_c(s, s') \Rightarrow P s' \\ \equiv \langle \text{Var. change} \rangle \forall s . A s \Rightarrow \forall s' . R_c(s, s') \Rightarrow P s' \end{array}$$

Hence  $Wla_c P s \equiv \forall s' . R_c(s, s') \Rightarrow P s'$ ; also  $Wa_c P = Wla_c P \wedge T_c$