Theory Of Programs

Bertrand Meyer, 23 November 2015

Axiomatic descriptions

LAWS OF PROGRAMMING

A complete set of algebraic laws is given for Dijkstra's nondeterministic sequential programming language. Iteration and recursion are explained in terms of Scott's domain theory as fixed points of continuous functionals. A calculus analogous to weakest preconditions is suggested as an aid to deriving programs from their specifications.

C. A. R. HOARE, I. J. HAYES, HE JIFENG, C. C. MORGAN, A. W. ROSCOE, J. W. SANDERS, I. H. SORENSEN, J. M. SPIVEY, and B. A. SUFRIN

“Laws”

(1) Clearly, it does not make any difference in what order a choice is offered. “Tea or coffee?” is the same as “coffee or tea?.”

$$P \cup Q = Q \cup P \qquad \text{(symmetry)}$$

(2) A choice between three alternatives (tea, coffee, or cocoa) can be offered as first a choice between one alternative and the other two, followed (if necessary) by a choice between the other two, and it does not matter in which way the choices are grouped.

$$P \cup (Q \cup R) = (P \cup Q) \cup R \qquad \text{(associativity)}$$

(3) A choice between one thing and itself offers no choice at all (Hobson’s choice).

$$P \cup P = P \qquad \text{(idempotence)}$$

(4) The *ABORT* command already allows completely arbitrary behavior, so an offer of further choice makes no difference to it.

$$\bot \cup P = \bot \qquad \text{(zero } \bot\text{)}$$

“Laws”

The Laws of Programming Unify Process Calculi*

Tony Hoare[1] and Stephan van Staden[2]

[1] Microsoft Research, Cambridge, United Kingdom
[2] ETH Zurich, Switzerland
Stephan.vanStaden@inf.ethz.ch

Abstract. We survey the well-known algebraic laws of sequential programming, and propose some less familiar laws for concurrent programming. On the basis of these laws, we derive the rules of a number of classical programming and process calculi, for example, those due to Hoare, Milner, and Kahn. The algebra is simpler than each of the calculi derive it, and stronger than all the calculi put together. We end with a s describing the role of unification in Science and Engineering.

Table 1. Basic properties of the operators

	$\vee$	$\wedge$	$;$	$\parallel$
Commutative	yes	yes	no	yes
Associative	yes	yes	yes	yes
Idempotent	yes	yes	no	no
Unit	$\bot$	$\top$	*skip*	*skip*
Zero	$\top$	$\bot$	$\bot$	$\bot$

1073×939

In addition to such laws, distribution laws state the relationships between two (or more) operators. All the binary operators in the table distribute through ($\vee$), i.e. for $\circ \in \{\vee, \wedge, ;, \parallel\}$ we have:

- $P \circ (Q \vee R) = (P \circ Q) \vee (P \circ R)$
- $(P \vee Q) \circ R = (P \circ R) \vee (Q \circ R)$

Another distribution law, analogous to the exchange law of category theory, specifies how sequential and concurrent composition interact:

- $(P \parallel Q) ; (R \parallel S) \subseteq (P ; R) \parallel (Q ; S)$

“Natural” semantics

Natural Semantics

G. Kahn
INRIA, Sophia–Antipolis
06565 Valbonne CEDEX, FRANCE

Abstract

During the past few years, many researchers have begun to present semantic specifications in a style that has been strongly advocated by Plotkin in [19]. The purpose of this paper is to introduce in an intuitive manner the essential ideas of the method that we call now Natural Semantics, together with its connections to ideas in logic and computing. Natural Semantics is of interest *per se* and because it is used as a semantics specification formalism for an interactive computer system that we are currently building at INRIA.

5

“Natural” semantics laws

$$\rho \vdash \text{number } N \Rightarrow N$$

$$\rho \vdash \text{true} \Rightarrow \mathit{true}$$

$$\rho \vdash \text{false} \Rightarrow \mathit{false}$$

$$\rho \vdash \lambda P.E \Rightarrow [\![\lambda P.E, \rho]\!]$$

$$\frac{\rho \overset{\text{val\_of}}{\vdash} \text{ident } I \mapsto \alpha}{\rho \vdash \text{ident } I \Rightarrow \alpha}$$

$$\frac{\rho \vdash E_1 \Rightarrow \mathit{true} \qquad \rho \vdash E_2 \Rightarrow \alpha}{\rho \vdash \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 \Rightarrow \alpha}$$

$$\frac{\rho \vdash E_1 \Rightarrow \mathit{false} \qquad \rho \vdash E_3 \Rightarrow \alpha}{\rho \vdash \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 \Rightarrow \alpha}$$

$$\frac{\rho \vdash E_1 \Rightarrow \alpha \qquad \rho \vdash E_2 \Rightarrow \beta}{\rho \vdash (E_1, E_2) \Rightarrow (\alpha, \beta)}$$

$$\frac{\rho \vdash E_1 \Rightarrow [\![\lambda P.E, \rho_1]\!] \qquad \rho \vdash E_2 \Rightarrow \alpha \qquad \rho_1 \cdot P \mapsto \alpha \vdash E \Rightarrow \beta}{\rho \vdash E_1\ E_2 \Rightarrow \beta}$$

$$\frac{\rho \vdash E_2 \Rightarrow \alpha \qquad \rho \cdot P \mapsto \alpha \vdash E_1 \Rightarrow \beta}{\rho \vdash \text{let } P = E_2 \text{ in } E_1 \Rightarrow \beta}$$

$$\frac{\rho \cdot P \mapsto \alpha \vdash E_2 \Rightarrow \alpha \qquad \rho \cdot P \mapsto \alpha \vdash E_1 \Rightarrow \beta}{\rho \vdash \text{letrec } P = E_2 \text{ in } E_1 \Rightarrow \beta}$$

6

The axiomatic method

Bertrand Russell (cited by Hoare et al.): an axiomatic approach (i.e. postulating the laws)

has the advantages of theft over honest toil

Hoare et al.:

of course, the mathematician should also design a model of the language, to check completeness and consistency of the laws, to provide a framework for the specifications of programs, and for proofs of correctness

7

Defining functions

Real functions ξ and σ, such that:

$$\int \sigma = -\xi$$

$$\int \xi = \sigma$$

$$\xi(x)^2 + \sigma(x)^2 = 1$$

etc.

8

Programs

A **program** (or **specification**) over a state space S is given by

- A relation $post : S \leftrightarrow S$ -- Postcondition
- A set $Pre \subseteq S$ -- Precondition

Notation: $S \leftrightarrow S$
-- Relations on S, i.e. $\mathbb{P}(S \times S)$

For given program p, write these $post_p$ and Pre_p

Conversely, <post, Pre> is the program defined from post and Pre

9

Programs vs specifications

Examples:

- $x := 1$
- $\textbf{Result}^2 = Input$

10

Programs

A **program** (or **specification**) over a state space S is given by

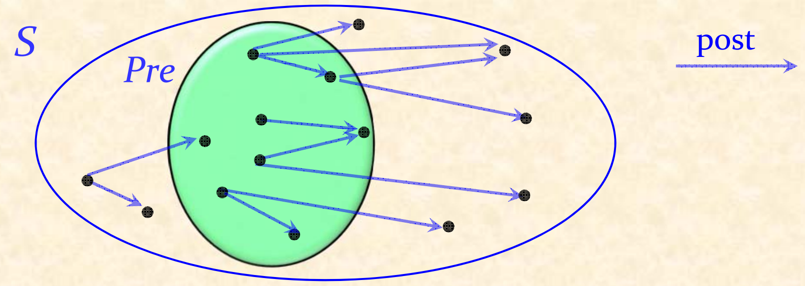
- A relation post : $S \leftrightarrow S$ -- Postcondition
- A set Pre $\subseteq S$ -- Precondition

Notation: $S \leftrightarrow S$
-- Relations on S, i.e. $\mathbb{P}(S \times S)$

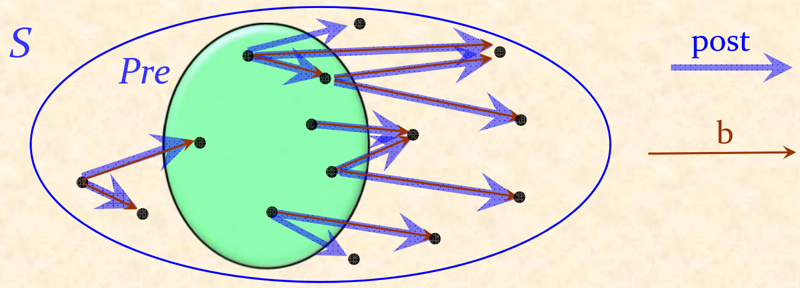
For given program p, write these $post_p$ and Pre_p

Conversely, <post, Pre> is the program defined from post and Pre

Program/specification



Determinism



A program is **deterministic** if b is a function*

The same concept applies to specifications

*"Functions" are possibly partial; total functions are always marked as such

13

Programming language, specification language

Set of predefined relations and combinators for building programs/specifications

14

Varieties of programs

Deterministic if $post_p$ is a function

Non-deterministic otherwise

Functional if every subset C of S is disjoint from $post_p$ (C)

Imperative otherwise

Object-oriented if if S is of the form $0..n \rightarrow O$ for an integer n and a set O of "objects"

Procedural otherwise

Notation: r (A)
-- Image of a set by a relation

A **program** (or **specification**) over a state space S is given by

- A relation post : $S \leftrightarrow S$ -- Postcondition
- A set Pre $\subseteq$ S -- Precondition

Equivalence

Two programs are equivalent if they have the same Pre and the same post / Pre

Notation: restriction and corestriction of a relation
r / X -- $r \cap (X \times S)$
r \ Y -- $r \cap (S \times Y)$

A **program** (or **specification**) over a state space S is given by

- A relation post : $S \leftrightarrow S$ -- Postcondition
- A set Pre $\subseteq$ S -- Precondition

Feasibility

A program is **feasible** if $Pre \subseteq \underline{post}$

Notation: $\underline{r}$, $\overline{r}$
-- Domain, codomain of a relation

A **program** (or **specification**) over a state space S is given by

- A relation $post : S \leftrightarrow S$ -- Postcondition
- A set $Pre \subseteq S$ -- Precondition

17

Refinement

p_2 **refines** p_1 if:

- $post_2 \subseteq_{Pre_1} post_1$ -- Strengthening
- $Pre_2 \supseteq Pre_1$ -- Weakening

Notation: $r \subseteq_X r'$ means $(r / X) \subseteq r'$

- Also: $S_2 \supseteq S_1$ -- Specialization

Refinement is a preorder over specifications/programs
(partial order modulo program equivalence)

18

Implementation

An **implementation** of p is a feasible refinement of p

19

Implementation theorem

A specification having an implementation is feasible

In other words, if a specification has a feasible refinement, it is itself feasible

Proof: let p be the specification and i its implementation; we must prove that $Pre_p \subseteq \underline{post}_p$. We have:

/1/	Pre_p	$\subseteq$	Pre_i	-- Weakening
/2/	Pre_i	$\subseteq$	$\underline{post}_i$	-- Feasibility of i
/3/	Pre_p	$\subseteq$	$\underline{post}_i$	-- From /1/ and /2/
/4/	$post_i$	$\underset{Pre_p}{\subseteq}$	$post_p$	-- Strengthening
/5/	$\underline{post}_i \cap Pre_p$	$\subseteq$	$\underline{post}_p$	-- From /4/
/6/	Pre_p	$\subseteq$	$\underline{post}_p$	-- From /3/and /5/

20

Refinement safety

An operator § is refinement-safe if

$$q_1 \subseteq p_1 \text{ and } q_2 \subseteq p_2 \text{ implies } (q_1 \S q_2) \subseteq (p_1 \S p_2)$$

Theorem: all the operators introduced in this discussion are refinement-safe

Contracted programs

If p is a program, the notation

require Pre **do** p **ensure** post **end**

(a "contracted program")

states that p is an implementation of <post, Pre>

Reminder: <post, Pre> is the program of postcondition post and precondition Pre

A (contracted) program is a proof obligation

Programming language, specification language

Set of predefined relations and combinators for building programs/specifications

23

Fundamental combinators

Name	**Notation**	**Postcondition**	**Intuition**
Choice (union)	$p_1 \cup p_2$	$post_1 \cup post_2$	Performs like p_1 or like p_2
Composition (sequence, compound, ...)	$p_1 ; p_2$	$post_1 ; post_2$	Performs like p_1 then like p_2
Restriction (guarded command)	C: p (also: p / C)	$post_p$ / C	Performs like p on C
Corestriction	p \ C	$post_p$ \ C	Corestriction

Operations on relations:
r ; r' -- Composition
r / r' -- Restriction
r \ r' -- Corestriction

24

Theorems

- $C_1 : (C_2 : p) \quad = \quad C_2 : (C_1 : p)$ -- 1
- $C_1 : (C_2 : p) \quad = \quad (C_1 \cap C_2) : p$ -- 2
- $C : (p_1 \cup p_2) \quad = \quad (C : p_1) \cup (C : p_2)$ -- 3
- $C : (p_1 ; p_2) \quad = \quad (C : p_1) ; p_2$ -- 4
- $q ; (p_1 \cup p_2) \quad = \quad (q ; p_1) \cup (q ; p_2)$ -- 5
- $(p_1 \cup p_2) ; q \quad = \quad (p_1 ; q) \cup (p_2 ; q)$ -- 6
- $(p_1 \cup p_2) \setminus C \quad = \quad (p_1 \setminus C) \cup (p_2 \setminus C)$ -- 7
- If $D \subseteq C$, then $(C : p) \subseteq (D : p)$ -- 8
- If $q \subseteq p$, then $(C : q) \subseteq (C : p)$ -- 9

25

Extreme programs

Skip: <Identity, S>

-- where Identity is $\lambda\ x \mid x$

Havoc: $<S \times S, S>$

Fail: $< \varnothing, \varnothing >$

Notation: <post, Pre> is the program of postcondition post and precondition Pre

26

More theorems

	$(p \setminus C)$	$= (p ; (C: Skip)$	
	$(p ; Skip)$	$= (Skip ; p)$	$= p$
	$(p \cup Fail)$	$= (Fail \cup p)$	$= p$
	$(p; Fail)$	$= (Fail ; p)$	$= Fail$
	$(p \cup Havoc)$	$= (Havoc \cup p)$	$= Havoc$
	$(p ; havoc)$	$= (Pre_p: Havoc)$	
	p	$\subseteq (C: p)$	
If $q_1 \subseteq p_1$ and $q_2 \subseteq p_2$:	$(q_1 \cup q_2)$	$\subseteq (p_1 \cup p_2)$	
If $q_1 \subseteq p_1$ and $q_2 \subseteq p_2$:	$(q_1 ; q_2)$	$\subseteq (p_1 ; p_2)$	
For any p:	p	$\subseteq (Pre_p: Havoc)$	
For any total p:	p	$\subseteq Havoc$	
If and only if $p = Fail$:	p	$\subseteq Fail$	
If and only if $p = Fail$:	$Fail$	$\subseteq p$	

27

Atomic concurrency

Name	**Notation**	**Postcondition**	**Intuition**
Atomic concurrency	$p_1 \parallel p_2$	$(p_1 ; p_2)$ $\cup$ $(p_2 ; p_1)$	Performs once like each of p_1 and p_2

Theorems:

$p_1 \parallel (p_2 \cup p_3)$	$=$	$(p_1 \parallel p_2) \cup (p_1 \parallel p_3)$
$(p_1 \cup p_2) \parallel p_3)$	$=$	$(p_1 \parallel p_3) \cup (p_2 \parallel p_3)$
$(C: p_1 \parallel p_2)$	$=$	$(C: p_1) \parallel (C: p_2)$
$(p_1 \parallel p_2) \setminus C$	$=$	$(p_1 \setminus C) \parallel (p_2 \setminus C)$
$(p_1 ; p_2)$	$\subseteq$	$(p_1 \parallel p_2)$
$(p_2 ; p_1)$	$\subseteq$	$(p_1 \parallel p_2)$

Two programs **commute** if $(p_1 ; p_2) = (p_2 ; p_1)$

28

Fine-grain concurrency

Ternary operator:

$(p_1, p_2) \mid\mid q$

defined as:

$((p_1 \mid\mid q) ; p_2) \cup (p_1 ; (p_2 \mid\mid q))$

Some theorems:

- $(p_1, p_2) \mid\mid q \;=\; (q ; p_1 ; p_2) \cup (p_1 ; q ; p_2) \cup (p_1 ; p_2 ; q)$
- $(p_1 ; p_2) \mid\mid q \;\subseteq\; (p_1, p_2) \mid\mid q$
- $p_1 ; (p_2 \mid\mid q) \;\subseteq\; (p_1, p_2) \mid\mid q$ -- "Laws of exchange"
- $(p \mid\mid q_1) ; q_2 \;\subseteq\; (q_1, q_2) \mid\mid p$ -- (Hoare/Van Staden)

29

Conditionals

Name	**Notation**	**Definition**
Guarded conditional	**if** $C_1 : p_1$ [] $C_2 : p_2$ **end**	$(C_1 : p_1) \cup (C_2 : p_2)$
If-then-else	**if** C **then** p_1 **else** p_2 **end**	$(C : p_1) \cup (C' : p_2)$

Notation: C'
-- Complement of a set

30

More theorems

Notation: C' -- Set complement

(C: p)	=	**if** C: p **end**
if C_1: p_1 [] C_2: p_2 **end**	⊆	C_1: p_1
D: (**if** C_1: p_1 [] C_2: p_2 **end**)	⊆	(**if** $(D \cap C_1)$: p_1 [] $(D \cap C_2)$: p_2 **end**)
if C **then** p_1 **else** p_2 **end**	=	**if** C: p_1 [] C': p_2 **end**
if C **then** p_1 **else** p_2 **end**	=	**if** C' **then** p_2 **else** p_1 **end**

If $D_1 \subseteq C_1$ and $D_2 \subseteq C_2$, then
if D_1: p [] D_2: q **end** ⊆ **if** C_1: p [] C_2: q **end**

If $q_1 \subseteq p_1$ and $q_2 \subseteq p_2$, then
if C_1: q_1 [] C_2: q_2 **end** ⊆ **if** C_1: p_1 [] C_2: p_2 **end**

If $q_1 \subseteq p_1$ and $q_2 \subseteq p_2$, then
if C **then** q_1 **else** q_2 **end** ⊆ **if** C **then** p_1 **else** p_2 **end**

31

Special conditions

True is another name for S
False is another name for ∅
and is another name for ∩ , **or** another name for ∪, **implies** another name for ⊆
Theorems:

- (True: p) = p
- (False: p) = Fail
- p \ **True** = p
- p \ **False** = Fail
- (**if** True **then** p_1 **else** p_2 **end**) = p_1
- (**if** False **then** p_1 **else** p_2 **end**) = p_2
- **and**, **or**, **not**, **implies** distribute over choice, restriction and conditionals

32

Loops

Name	Notation	Definition
Fixed repetition	p^i	$p^0 = \underline{p}$: Skip $p^{i+1} = (p ; p^i)$
Arbitrary repetition	**loop** p **end**	$\bigcup_{i \geq 0} p^i$
"While loop"	**from** a **until** C **loop** b **end**	a ; (**loop** C': b **end**)\ C

Invariants

A condition I is an invariant of a program/specification p if

$$post_p (I \cap Pre_p) \subseteq I$$

Notation:
r (A) -- Image of a set by a relation

Theorems

- Any I disjoint from Pre_p is an invariant of p
- If I and J are invariants of p, so are $I \cap J$ an d $I \cup J$

Invariant refinement theorem

- If I is an invariant of p and $q \subseteq p$, then I is an invariant of q / Pre_p

Invariant preservation

All operators seen so far are invariant-preserving in the following sense: an invariant of the operands is also an invariant of the result

Loop invariant

A loop invariant of

from a **until** C **loop** b **end**

is a subset of $\overline{a}$ that is an invariant of C': b

Notations: $\underline{r}$, $\overline{r}$
-- Domain, codomain of a relation

Loop correctness theorem

If I is a loop invariant of the loop

L = (**from** a **until** C **loop** b)

then

$\overline{L} \subseteq C \cap I$

37

Loop feasibility theorem

For feasible a and b, the loop

from a **until** C **loop** b

is feasible if both:

- $\underline{b} \cup C$ is a loop invariant
- C': $post_b$ is well-founded

38

Contracted programs

If p is a program, the notation

require Pre **do** p **ensure** post **end**

states that p is an implementation of <post, Pre>

A (contracted) program is a proof obligation

39

Contract refinement theorem

If

post $\subseteq$ post'
Pre' $\subseteq$ Pre

and the following is a contracted program:

require Pre **do** p **ensure** post **end**

then so is

require Pre' **do** p **ensure** post' **end**

40

Properties of programs

Name	Notation	Definition
Strongest postcondition of b for Pre	b **sp** post	$post_b$ / Pre
Weakest precondition of b for post	b **wp** post	$\underline{b}$ — $\underline{post_b \text{ — } post}$

b	**sp**	**False**	=	**Fail**
b	**wp**	**Fail**	=	**False**
Fail	**sp**	C	=	**Fail**
Fail	**wp**	p	=	**False**
b	**sp**	$(p \cup q)$	=	(b **sp** p) $\cup$ (b sp q)
b	**wp**	$(p \cup q)$	$\supseteq$	(b **wp** p) $\cup$ (b **wp** q)

Theorems

- *Pre* **sp** *i* is the smallest relation *post* such that *Pre, i* and *post* define a correct program
- *i* **wp** *post* is the largest set *Pre* such that *Pre, i* and *post* define a correct program
- Any implementation of the MAI (Most Abstract Implementation) of a specification p is an implementation of p
- If p is feasible, its MAI is an implementation of p
- The MAI is the largest relation *i* such that *Pre, i* and *post* define a correct program

A project: FLIP

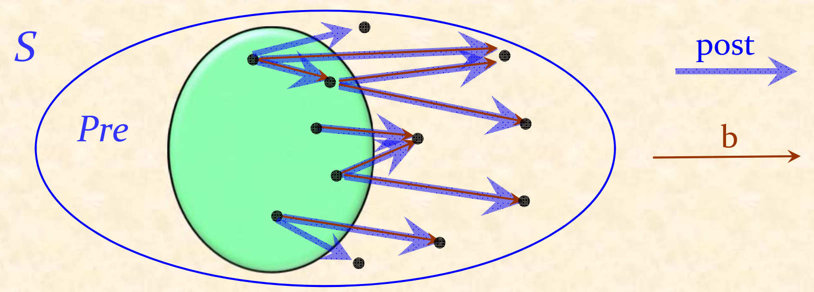
Formal Language Innovation Platform

Eiffel library:

- Basic classes representing key mechanisms: aggregation, alternation…
- Notion of proof
- Deferred classes representing the notions discussed earlier: environment, state, instruction, expression…
- Proof mechanisms
- Effective classes representing common notions, e.g. assignment, state in a Pascal-like language, state in an OO language…
- Pre-packaged proof

43

A program as a proof obligation



44

Definition: Programming

Programming is the process of devising interesting contract-implementation pairs and discharging the associated proof obligations

Program = specification + implementation + proof obligation

45

Programming

46