Let's Unify Theories of Programming

He Jifeng
Software Engineering Institute
East China Normal University
Shanghai China

Syllabus

- A simple theory
- Relational programming
- Link with wp-calculus
- Design calculus
- Implementation
- Design capturing
- Linking theories and Galois connection
- Algebra of programs
- An operational model
- Probabilistic programming
- Link semantic models

Theories of Programming

A theory of programming relates each program to the specification of what it is intended to achieve.

A unifying theory is applicable to a general paradigm of computing, supporting the classification of many programming languages as correct instances of the paradigm.

A Diversity of Presentation

1. A denotational method denotes each notation as some *value* in a mathematical domain. It is a link between specifications and programs.

2. An algebraic method says if two different written programs happen to mean the same thing. It is widely used for program optimisation and verification.

3. An operational presentation describes how a program can be executed by an abstract machine. It supports compiler design, complexity analysis and program testing.

Unit 1: A Simple Theory

Ingredients

(1) Alphabet: concepts related to the real world.

(2) Signature : primitives and combinators

(3) Laws : a set of equations that are provable.

Alphabet

Free variables are used to stand for possible measurement taken from the experiment.

1. initial state $\{x, y, \ldots, z\}$.
2. final state $\{x', y', \ldots, z'\}$.
3. status of programs $\{ok, st\}$.
5. interaction $\{tr\}$.
6. synchronisation $\{ref\}$.
7. control $\{l, start, finish\}$

Observation and Specification

An observation consists of a set of equations ascribing particular constant values to each variable

$$x = 4 \ \wedge \ldots \wedge z = false$$

A specification gives a set of allowable observations

$$x' = x + 1 \wedge y' = y \wedge z' = z$$

can be used to describe the effect of execution of assignment $x := x + 1$, where x, y and z are program variables used as global ones in the program.

Conjunction

In general, a specification is written as a conjunction of a set of predicates

$$S_1 \ \wedge \ S_2 \wedge \ldots$$

For example, the operation **swap**$(x,\, y)$ can be specified by the predicate

$$(x' = y) \ \wedge \ (y' = x)$$

The sorting program can be described by

$$\mathbf{bag}(A') \ = \ \mathbf{bag}(A) \ \wedge$$
$$\neg \exists i,\, j \in \mathbf{domain}(A) \bullet i < j \ \wedge \ A'[i] > A'[j]$$

Laws

Conjunction is idempotent, symmetric and associative.

$(\wedge - 1)\ P \wedge P = P$

$(\wedge - 2)\ P \wedge Q = Q \wedge P$

$(\wedge - 3)\ P \wedge (Q \wedge R) = (P \wedge Q) \wedge R$

$\wedge$ has **true** as its unit, **false** as its zero

$(\wedge - 4)\ P \wedge \mathbf{true} = P$

$(\wedge - 5)\ P \wedge \mathbf{false} = \mathbf{false}$

Water Tank

t: the time since the the start of the apparatus

v_t: the volume of the liquid in the tank at time t

in_t, out_t: the total amount of liquid poured and drained up to time t

a_t, b_t: the setting of the input valve and output valve at time t

Water tank is governed by the following laws:

(1) $in_t + v_0 \;=\; out_t + v_t$

(2) $0 \leq a_t \leq amax, \qquad 0 \leq b_t \leq bmax$

(3) $\dot{in} = k \times a, \qquad \dot{out} = k \times b + \delta \times v$

Correctness

Let P and S have the same alphabet $\{x, \ldots, z\}$.

P satisfies S if

$$\forall x, \ldots, z \bullet (P(x, \ldots, z) \Rightarrow S(x, \ldots, z))$$

We rewrite it as $[P \Rightarrow S]$ (or $P \sqsupseteq S$)

For example

$$(x' = x + 1 \wedge y' = y + 2) \sqsupseteq (x' \geq x \wedge y' \geq y)$$

Theorem $\sqsupseteq$ is a pre-order.

(1) $P \sqsupseteq P$

(2) If $P \sqsupseteq Q$ and $Q \sqsupseteq R$, then $P \sqsupseteq R$

Refinement supports correct design

- **Stepwise Refinement**

  $((D \sqsupseteq S) \ \wedge \ (P \sqsupseteq D))$ implies $(P \sqsupseteq S)$

- **Piecewise Refinement**

  $((P \sqsupseteq S) \ \wedge \ (Q \sqsupseteq S))$ implies $((P \vee Q) \sqsupseteq S)$

- **Devide-and-Conquer**

  $(((S_1 \wedge S_2) \sqsupseteq S) \ \wedge \ (P_1 \sqsupseteq S_1) \ \wedge \ (P_2 \sqsupseteq S_2))$
  implies $(P_1 \wedge P_2 \sqsupseteq S)$

- **Component Replacement**

  $(P \sqsupseteq S)$ implies $(\mathcal{F}(P) \sqsupseteq \mathcal{F}(S))$

Reusable Components

Let $S(x, y)$ be a specification, and $Q(y)$ an existing component. We wish to deliver a product of form $P(x) \wedge Q(y)$, and use R/Q to denote the weakest product $P(x)$.

Theorem $(P \wedge Q) \sqsupseteq R$ **iff** $P \sqsupseteq R/Q$

$(/-1)$ $(R_1 \wedge R_2)/Q \;=\; (R_1/Q) \wedge (R_2/Q)$

$(/-2)$ $R/(Q_1 \wedge Q_2) \;=\; (R/Q_1)/Q_2$

$(/-3)$ $\mathbf{true}/Q \;=\; \mathbf{true}$

$(/-4)$ $R/\mathbf{false} \;=\; \mathbf{true}$

Linking Models

Let $S(a)$ be a specification, and $P(c)$ a product. P is an implementation of S with respect to an interpretation $A(c, a)$ if

$$(\exists c \bullet P(c) \wedge A(c, a)) \sqsupseteq S(a)$$

We denote the above fact by $P \sqsupseteq_A S$

Theorem If $P \sqsupseteq_{A_1} Q$ and $Q \sqsupseteq_{A_2} S$ then $P \sqsupseteq_{A_1 \circ A_2} S$, where

$$A_1 \circ A_2 =_{df} \exists b \bullet A_1(c, b) \wedge A_2(b, a)$$

Unit 2 : A Theory of Relational Programming

Relations

A relation is a triple (IN, OUT, P) where

- IN and OUT are disjoint sets of names, and
- P is a predicate of free variables of the set $IN \cup OUT$

We define $in\alpha P =_{df} IN$, $out\alpha P = OUT$.

Signature of Relational Programming

Primitives

1. *assignment*: $x := e$ and *empty*: *skip*
2. *top*: $\top$ and *bottom*: $\bot$

Combinators

1. *conditional*: $P \lhd b \rhd Q$
2. *composition*: $P; Q$
3. *nondeterminism*: $P \sqcap Q$
4. *recursion*: $\mu X \bullet \mathcal{F}(X)$

Testable Conditions

Any non-trivial program requires a facility to select between alternative actions in accordance with the truth or falsity of some *testable condition* b.

The restriction that b contains no dashed variables ensures that it can be tested before starting either of the action.

Conditional

If P and Q are predicates describing two programs with the same alphabet, then the conditional

$$P \lhd b \rhd Q$$

describes a program which behaves like P if b is true initially, or like Q otherwise.

$$\alpha(P \lhd b \rhd Q) \;=_{df}\; \alpha P$$

$$P \lhd b \rhd Q \;=_{df}\; (b \wedge P) \vee (\neg b \wedge Q)$$

Laws of Conditional

Conditional choice is idempotent, skew-symmetric and associative.

$(cond-1)\ P \lhd b \rhd P \ = \ P$

$(cond-2)\ P \lhd b \rhd Q \ = \ Q \lhd \neg b \rhd P$

$(cond-3)\ (P \lhd b \rhd Q) \lhd c \rhd R \ = \ P \lhd b \wedge c \rhd (Q \lhd c \rhd R)$

Conditional choice distributes over conditional, and has $\lhd true \rhd$ as its unit.

$(cond-4)\ P \lhd b \rhd (Q \lhd c \rhd R) \ = \ (P \lhd b \rhd Q) \lhd c \rhd (P \lhd b \rhd R)$

$(cond-5)\ P \lhd \mathbf{true} \rhd Q \ = \ P$

Additional Law

$$(cond-6)\ P \lhd b \rhd (Q \lhd b \rhd R) \ = \ P \lhd b \rhd R$$

Proof

$$\begin{aligned} & LHS && \{(cond-2)\} \\ = \ & (Q \lhd b \rhd R) \lhd \neg b \rhd P && \{(cond-3)\} \\ = \ & Q \lhd false \rhd (R \lhd \neg b \rhd P) && \{(cond-2,\ 5)\} \\ = \ & RHS \end{aligned}$$

Additional Law

$$(cond-7)\ P \lhd b \rhd (P \lhd c \rhd Q) \ = \ P \lhd b \vee c \rhd Q$$

Proof

$$\begin{aligned} & LHS && \{(cond-2)\} \\ = \ & (Q \lhd \neg c \rhd P) \lhd \neg b \rhd P && \{(cond-3,\ 1)\} \\ = \ & Q \lhd \neg c \wedge \neg b \rhd P && \{(cond-2)\} \\ = \ & RHS \end{aligned}$$

Composition

The most characteristic combinator of a sequential language is *sequential composition*, often denoted by demicolon. If P and Q are predicates describing two programns, their composition

$$P; Q$$

describes a program which may be executed by first executing P, and when P terminates then Q is started. The final state of P is passed on as the initial state of Q but this is only an intermediate state of $(P; Q)$, and cannot be directly observed.

Composition

Let $out\alpha P = in\alpha Q$ (say $\{v\}$). Define

$$P(v'); Q(v) =_{df} \exists m \bullet (P(m) \wedge Q(m))$$

$$in\alpha(P; Q) =_{df} in\alpha P$$

$$out\alpha(P; Q) =_{df} out\alpha Q$$

The bound variables m record the intermediate values of the program variables v, and so represent the intermediate state as control passes from P to Q.

Laws of Composition

Composition is associative.

$(;-1)\ P;(Q;R) \ = \ (P;Q);R$

Composition distributes leftward over conditional.

$(;-2)\ (P \lhd b \rhd Q);R \ = \ (P;R) \lhd b \rhd (Q;R)$

Assignment

The assignment is the basic action in all procedural programming languages. Every assignment should technically be marked by its alphabet A, which is needed to define an important part of its meaning – that all the variables not mentioned on the left hand side remain unchanged.

Let $A = \{x, \ldots, z\}$, and $FreeVar(e) \subseteq A$.

$$x :=_A e \;\; =_{df} \;\; (x' = e \;\wedge\; \ldots \;\wedge\; z' = z)$$

$$\alpha(x :=_A e) \;\; =_{df} \;\; A + \{v' \mid v \in A\}$$

Define $skip_A \;\; =_{df} \;\; (x' = x \;\wedge \ldots \wedge z' = z)$

Laws of Assignment

$(asgn-1)\ x := e \ = \ x, y := e, y$

The order of the listing is immaterial.

$(asgn-2)\ \ (x, y, z := e, f, g) \ = \ (y, x, z := f, e, g)$

Evaluation of an expression or a condition uses the value most recently assigned to its variables.

$(asgn-3)\ \ (x := e\,;\, x := f(x)) \ = \ x := f(e)$

$(asgn-4)\ \ x := e\ ;(P \triangleleft b(x) \triangleright Q) \ =$

$$(x := e; P) \triangleleft b(e) \triangleright (x := e; Q)$$

skip is the unit of composition.

$(asgn-5)\ \ P; skip \ = \ P \ = \ skip; P$

Non-determinism

If P and Q are predicates describing the behaviour of programs with the same alphabet. The notation $P \sqcap Q$ stands for a program which is executed either P or Q, but with no indication which one will be chosen.

$$P \sqcap Q \;=_{df}\; P \vee Q$$

$$\alpha(P \sqcap Q) \;=_{df}\; \alpha P$$

Nondeterministic choice may be easily implemented by arbitrary selection of either of the operands, and the selection may be made at any time, either before or after the program is compiled or even after it starts execution.

Laws of $\sqcap$

$\sqcap$ is symmetric, associative and idempotent.

$(\sqcap - 1)\ P \sqcap Q \ = \ Q \sqcap P$

$(\sqcap - 2)\ P \sqcap (Q \sqcap R) \ = \ (P \sqcap Q) \sqcap R$

$(\sqcap - 3)\ P \sqcap P \ = \ P$

Theorem (Link between $\sqsupseteq$ and $\sqcap$)

$$(P \sqsupseteq Q) \ \textbf{iff} \ (P \sqcap Q) \ = \ Q$$

Additional Law

$\sqcap$ distributes through itself.

$(\sqcap - 4)\ P \sqcap (Q \sqcap R) \ = \ (P \sqcap Q) \sqcap (P \sqcap R)$

Proof

$$\begin{aligned} & RHS && \{(\sqcap - 1,\ 2)\} \\ = \ & (P \sqcap P) \sqcap (Q \sqcap R) && \{(\sqcap - 3)\} \\ = \ & LHS \end{aligned}$$

Distributivity

All programming combinators defined so far distribute through $\sqcap$.

$(\sqcap - 5)\ P \lhd b \rhd (Q \sqcap R) \ = \ (P \lhd b \rhd Q) \sqcap (P \lhd b \rhd R)$

$(\sqcap - 6)\ (P \sqcap Q); R \ = \ (P; R) \sqcap (Q; R)$

$(\sqcap - 7)\ P; (Q \sqcap R) \ = \ (P; Q) \sqcap (P; R)$

$(\sqcap - 8)\ P \sqcap (Q \lhd b \rhd R) \ = \ (P \sqcap Q) \lhd b \rhd (P \sqcap R)$

Corollary All combinators are monotonic in all arguments.

Lower And Upper Bounds

Let $\mathcal{S}$ be a set of relations with the same alphabet.

$\sqcap\mathcal{S}$ denotes the *greatest lower bound* of relations of $\mathcal{S}$, and is defined by

(**bound-1**) $(\sqcap\mathcal{S}) \sqsupseteq R$ **iff** $P \sqsupseteq R$ for all $P \in \mathcal{S}$

The *least upper bound* of relations of $\mathcal{S}$ is defined by

(**bound-2**) $(\sqcup\mathcal{S}) \sqsubseteq R$ **iff** $P \sqsubseteq R$ for all $P \in \mathcal{S}$

Top and Bottom

The top element of the lattice is

$$\top \; =_{df} \; \sqcap \{\} \; = \; \textbf{false}$$

The bottom one is

$$\bot \; =_{df} \; \sqcup \{\} \; = \; \textbf{true}$$

Distributivity and Disjunctivity

(**bound-3**) $(\sqcup\mathcal{S}) \sqcap Q \;=\; \sqcup\{X \sqcap Q \mid X \in \mathcal{S}\}$

(**bound-4**) $(\sqcap\mathcal{S}) \sqcup Q \;=\; \sqcap\{X \sqcup Q \mid X \in \mathcal{S}\}$

Composition is *universally disjunctive*

(**bound-5**) $(\sqcap\mathcal{S})\,;Q \;=\; \sqcap\{X;Q \mid X \in \mathcal{S}\}$

(**bound-6**) $P;(\sqcap\mathcal{S}) \;=\; \sqcap\{P;X \mid X \in \mathcal{S}\}$

Tarski's Fixed Point Theorem

If $\mathcal{F}$ is a monotonic mapping on a complete lattice $(\mathcal{C}, \sqsupseteq)$, then the solutions of equation $X = \mathcal{F}(X)$ form a complete lattice, where

$$\mu\mathcal{F} =_{df} \sqcap \{P \mid P \sqsupseteq \mathcal{F}(P)\}$$

is the weakest solution, and

$$\nu\mathcal{F} =_{df} \sqcup \{P \mid P \sqsubseteq \mathcal{F}(P)\}$$

is the strongest solution. In particular

$$\mu X = \mathbf{true} = \bot, \quad \nu X = \mathbf{false} = \top$$

Laws of Fixed Points

$(\mu - 1)$ $(P \sqsupseteq \mathcal{F}(P))$ **iff** $P \sqsupseteq \mu\mathcal{F}$

$(\mu - 2)$ $\mathcal{F}(\mu\mathcal{F}) = \mu\mathcal{F}$

$(\nu - 1)$ $(P \sqsubseteq \mathcal{F}(P))$ **iff** $P \sqsubseteq \nu\mathcal{F}$

$(\nu - 4)$ $\mathcal{F}(\nu\mathcal{F}) = \nu\mathcal{F}$

Proof of $\mu - 2$

$$\forall Y \bullet (Y \sqsupseteq \mathcal{F}(Y)) \Rightarrow (Y \sqsupseteq \mu\mathcal{F})$$

implies $\forall Y \bullet (Y \sqsupseteq \mathcal{F}(Y)) \Rightarrow (\mathcal{F}(Y) \sqsupseteq \mathcal{F}(\mu\mathcal{F}))$

implies $\forall Y \bullet (Y \sqsupseteq \mathcal{F}(Y)) \Rightarrow (Y \sqsupseteq \mathcal{F}(\mu\mathcal{F}))$

implies $\sqcap\{Y \mid Y \sqsupseteq \mathcal{F}(Y)\} \sqsupseteq \mathcal{F}(\mu\mathcal{F})$

equiv $\mu\mathcal{F} \sqsupseteq \mathcal{F}(\mu\mathcal{F})$

implies $(\mu\mathcal{F} \sqsupseteq \mathcal{F}(\mu\mathcal{F})) \wedge (\mathcal{F}(\mu\mathcal{F}) \sqsupseteq \mathcal{F}^2(\mu\mathcal{F}))$

implies $(\mu\mathcal{F} \sqsupseteq \mathcal{F}(\mu\mathcal{F})) \wedge (\mathcal{F}(\mu\mathcal{F}) \sqsupseteq \mu\mathcal{F})$

equiv $\mu\mathcal{F} = \mathcal{F}(\mu\mathcal{F})$

Unit 3: Link with wp-Calculus

Precondition and Postcondition

A condition p has been defined as a predicate not containing dashed variables. It describes the values of global variables of a program Q before its execution starts. Such a condition is therefore called a *precondition* of the program.

If r is a condition, Let r' be the result of placing a dash on all its variables. As a result, r' describes the values of the variables of a program Q when it terminates. Such a condition is therefore called a *postcondition* of the program.

Hoare Triple

The overall specification of the program can often be formalised as a simple implication

$$p \implies r'$$

Define

$$p\{Q\}r \;=_{df}\; Q \sqsupseteq (p \implies r')$$

A Model for Hoare Logic

(1) If $p\{Q\}r$ and $p\{Q\}s$, then $p\{Q\}(r \wedge s)$

(2) If $p\{Q\}r$ and $q\{Q\}r$, then $\{(p \vee q)\{Q\}r$

(3) If $p\{Q\}r$ then $(p \wedge q)\{Q\}(r \vee s)$

(4) $r(e)\{x := e\}r(x)$

(5) If $(p \wedge b)\{Q_1\}r$ and $(p \wedge \neg b)\{Q_2\}r$,
then $p\{Q_1 \lhd b \rhd Q_2\}r$

(6) If $p\{Q_1\}s$ and $s\{Q_2\}r$, then $p\{Q_1; Q_2\}r$

(7) If If $p\{Q_1\}r$ and $p\{Q_2\}r$ then $p\{Q_1 \sqcap Q_2\}r$

(8) If $(b \wedge c)\{Q\}c$ then $c\{\nu X \bullet Q; X \lhd b \rhd skip\}(\neg b \wedge c)$

Proof of (8)

Let $Y =_{df} (c \implies \neg b' \wedge c')$. By $(\nu - 1)$ it is sufficient to prove

$$(Q; Y) \triangleleft b \triangleright skip \implies Y$$

Assume the antecedents

$$\begin{array}{lll} & (Q; Y) \triangleleft b \triangleright skip) \wedge c & \\ = & (b \wedge c \wedge Q); Y \vee (\neg b \wedge c \wedge skip) & \{(6)\} \\ \implies & \neg b' \wedge c' & \end{array}$$

Condition vs Annotation

Conditions can play a vital role in explaining the meaning of a program if they are included at appropriate points as an integral part of the program documentation. Such a condition is *asserted* or *expected* to be true at the point at which it is written.

It is known as a Floyd assertion; formally it is defined to have no effect if it is true, but to cause failure if it is false.

Assertion And Assumption

$$c_\perp \;=_{df}\; skip \triangleleft c \triangleright \perp \qquad \text{(assertion)}$$

$$c^\top \;=_{df}\; skip \triangleleft c \triangleright \top \qquad \text{(assumption)}$$

Theorem

(1) $b_\perp ; c_\perp \;=\; (b \wedge c)_\perp$

(2) $b_\perp \sqcap c_\perp \;=\; (b \wedge c)_\perp$

(3) $c_\perp \triangleleft b \triangleright d_\perp \;=\; (c \triangleleft b \triangleright d)_\perp$

(4) $b_\perp ; b^\top \;=\; b_\perp$

Correctness

Definition of correctness can be rewritten in a number of different way

$$
\begin{aligned}
& [Q \implies (p \implies r')] \\
= \quad & [p \implies ((Q \implies r'] \\
= \quad & [p \implies (\forall v' \bullet Q \implies r')] \\
= \quad & [p \implies \neg \exists v' (Q \wedge \neg r')] \\
= \quad & [p \implies \neg (Q; \neg r)]
\end{aligned}
$$

This last reformulation gives an answer to the following question: What is the *weakest* precodition under which execution of Q is guaranteed to achieve the condition r'?

Weakest Precondition

$$Q\,\mathbf{wp}\,r \;=_{df}\; \neg(Q;\neg r)$$

$$(wp-1)\;\; (x := e)\,\mathbf{wp}\,r(x) \;=\; r(e)$$

$$(wp-2)\;\; (P;Q)\,\mathbf{wp}\,r \;=\; P\,\mathbf{wp}\,(Q\,\mathbf{wp}\,r)$$

$$(wp-3)\;\; (P \lhd b \rhd Q)\,\mathbf{wp}\,r \;=\; (P\,\mathbf{wp}\,r) \lhd b \rhd (Q\,\mathbf{wp}\,r)$$

$$(wp-4)\;\; (P \sqcap Q)\,\mathbf{wp}\,r \;=\; (P\,\mathbf{wp}\,r) \wedge (Q\,\mathbf{wp}\,r)$$

Healthiness Conditions

Theorem

(1) If $[r \implies s]$ then $[Q\,\mathbf{wp}\,r \implies Q\,\mathbf{wp}\,s]$

(2) If $[Q \implies S]$, then $[S\,\mathbf{wp}\,r \implies Q\,\mathbf{wp}\,r]$

(3) $Q\,\mathbf{wp}\,(r \wedge s) = Q\,\mathbf{wp}\,r \wedge Q\,\mathbf{wp}\,s$

(4) $Q\,\mathbf{wp}\,false = false$

provided that $Q;\mathbf{true} = \mathbf{true}$

Right Inverse of Composition

Define

$$S/Q \;=_{df}\; Q \,\mathbf{wp}\, S$$

Theorem $(P;Q) \sqsupseteq S$ iff $P \sqsupseteq S/Q$

Theorem

(1) $S/(Q_1;Q_2) \;=\; (S/Q_2)/Q_1$

(2) $S/(Q_1 \lhd b \rhd Q_2) \;=\; (S/Q_1) \lhd b \rhd (S/Q_2)$

(3) $S/(Q_1 \sqcap Q_2) \;=\; (S/Q_1) \sqcup (S/Q_2)$

(4) $(S_1 \wedge S_2)/Q \;=\; (S_1/Q) \sqcup (S_2/Q)$

Left Inverse of Composition

Let P and S are predicates describing two programs. The notation $P \backslash S$ denotes the weakest postspecification of P with respect to S, and is defined by

$$(P; Q) \sqsupseteq S \quad \textbf{iff} \quad Q \sqsupseteq P \backslash S$$

Theorem $(P \backslash S)/Q \;=\; P \backslash (S/Q)$

Unit 4: Design Calculus

Incomplete Observation

Consider the program

$$\bot \; ; \; (x, \, y, \, z := 1, \, 2, \, 3)$$

In any normal implementation, it would fail to terminate, and so be equal to $\bot$. Unfortunately, our theory gives

$$\bot; (x' = 1 \; \wedge \; y' = 2 \; \wedge \; z' = 3) \; =$$

$$(x' = 1 \; \wedge \; y' = 2 \; \wedge \; z' = 3)$$

This is the same as if the prior non-terminating program had been omitted.

Required Laws

What we need are the following two laws

$$\bot; P \;=\; \bot \qquad \text{(left zero law)}$$

and

$$P; \bot \;=\; \bot \qquad \text{(right zero law)}$$

Initiation and Termination

We add two logical variables into the alphabet

ok records the observation that the program has been started.

ok' records the observation that the program has terminated.

Remark:

ok and ok' are not global variables held in the store of any program; and it is assumed that they will *never* be mentioned in any expression or assignment of the program text.

Design

A design D is a relation which can be expressed by

$$(ok \wedge P) \Rightarrow (ok' \wedge Q)$$

We write $D = (P \vdash Q)$, where predicates P and Q contain no ok or ok', and

(1) P is an assumption which the designer can rely on when D is initiated.

(2) Q is the commitment which must be true when D terminates.

$\perp$ can be rewritten as

$$\perp = \mathbf{true} = (\mathbf{false} \vdash \mathbf{true})$$

Refinement

Theorem

$[(P1 \vdash Q1) \Rightarrow (P2 \vdash Q2)]$

iff

$[P2 \Rightarrow P1]$ **and** $[(P2 \wedge Q1) \Rightarrow Q2]$

Proof

$$
\begin{aligned}
& LHS \\
\equiv\ & [D1[true,\ false/ok,\ ok'] \Rightarrow D2[true,\ false/ok,\ ok']] \\
& \wedge \\
& [D1[true,\ true/ok,\ ok'] \Rightarrow\ D2[true,\ true/ok,\ ok']] \\
& \wedge \\
& [D1[false/ok] \Rightarrow D2[false/ok]] \\
\equiv\ & [\neg P1 \Rightarrow \neg P2]\ \wedge\ [(P1 \Rightarrow Q1) \Rightarrow (P2 \Rightarrow Q2)] \\
\equiv\ & [P2 \Rightarrow P1]\ \wedge\ [(P1 \Rightarrow Q1) \wedge P2\ \Rightarrow Q2] \\
\equiv\ & RHS
\end{aligned}
$$

Equivalence

$$(D1 \equiv D2) \;=_{df}\; (D1 \sqsupseteq D2) \textbf{ and } (D2 \sqsupseteq D1)$$

Theorem

(1) $(P \vdash Q) \;\equiv\; (P \vdash (P \wedge Q))$

(2) $(P \vdash Q) \;\equiv\; (P \vdash (P \Rightarrow Q))$

(3) $(P \vdash Q) \equiv (P \vdash R)$ **iff**
$[(P \wedge Q) \Rightarrow R]$ **and** $[R \Rightarrow (P \Rightarrow Q)]$

(4) $(\textbf{false} \vdash \textbf{false}) \;\equiv\; (\textbf{false} \vdash \textbf{true})$

Nondeteminism

Theorem

$$(P1 \vdash Q1) \sqcap (P2 \vdash Q2) \;=\; (P1 \wedge P2) \vdash (Q1 \vee Q2)$$

Proof LHS

$$\begin{aligned}
&= \; (P1 \vdash Q1) \;\vee\; (P2 \vdash Q2) \\
&= \; \neg ok \;\vee\; \neg P1 \;\vee\; (ok' \wedge Q1) \;\vee \\
&\quad\; \neg ok \;\vee\; \neg P2 \;\vee\; (ok' \wedge Q2) \\
&= \; \neg ok \;\vee\; \neg (P1 \wedge P2) \\
&\quad\; \vee\; ok' \;\wedge\; (Q2 \vee Q2) \\
&= \; RHS
\end{aligned}$$

Conditional and Composition

Theorem

$$(P1 \vdash Q1) \lhd b \rhd (P2 \vdash Q2) \;=\; (P1 \lhd b \rhd P2) \vdash (Q1 \lhd b \rhd Q2)$$

Theorem

$$(P1 \vdash Q1); (P2 \vdash Q2) \;=$$
$$(\neg(\neg P1; \mathbf{true}) \wedge \neg(Q1; \neg P2)) \;\vdash\; (Q1; Q2)$$

Proof

Proof LHS

$$
\begin{aligned}
&= D1[false/ok'];D2[false/ok] \ \vee \\
&\quad D1[true/ok'];D2[true/ok] \\
&= (\neg ok \ \vee \ \neg P1);\mathbf{true} \ \vee \\
&\quad (\neg ok \ \vee \ \neg P1 \ \vee \ Q1);D2[true/ok] \\
&= \neg ok \ \vee \ (\neg P1);\mathbf{true} \ \vee \\
&\quad Q1;(\neg P2 \ \vee \ ok' \wedge Q2) \\
&= RHS
\end{aligned}
$$

Left Zero Law

$$\bot ; (P \vdash Q) \;=\; \bot$$

Proof

$$\begin{aligned}
& \bot ; (P \vdash Q) \\
= \;& (\mathbf{false} \vdash \mathbf{false}) ; (P \vdash Q) \\
= \;& \neg(\mathbf{true} ; \mathbf{true}) \wedge \neg(\mathbf{false} ; \neg P1) \;\vdash\; (\mathbf{false} ; Q2) \\
= \;& \mathbf{false} \vdash \mathbf{false} \\
= \;& \bot
\end{aligned}$$

Left Unit Law

$$skip; D \;=\; D$$

where

$$skip \;=_{df}\; \mathbf{true} \vdash (x' = x \wedge \ldots \wedge z' = z)$$

Proof

$$skip; (P \vdash Q)$$

$$= \; (\neg((\neg \mathbf{true}); \mathbf{true})$$

$$\wedge \neg((x' = x \wedge \ldots \wedge z' = z); \neg P)$$

$$\vdash \; (x' = x \wedge \ldots \wedge z' = z); Q$$

$$= \; P \vdash Q$$

Complete Lattice of Designs

Theorem

(1) $\sqcap_i (P_i \vdash Q_i) \;=\; (\wedge_i P_i) \vdash (\vee_i Q_i)$

(2) $\sqcup_i (P_i \vdash Q_i) \;=\; (\vee_i P_i) \vdash \wedge_i (P_i \Rightarrow Q_i)$

The top design is defined by

$$\top_{\mathbf{D}} \;=_{df}\; \mathbf{true} \vdash \mathbf{false} \;=\; \neg ok$$

It is a left zero of sequential composition in design calculus

$$\top_{\mathbf{D}}; D \;=\; \top_{\mathbf{D}}$$

Assignment Revisited

If the evaluation of expression e always yields a value, then

$$(x := e) \;=_{df}$$

$$\mathbf{true} \vdash (x' = e \;\wedge\; \ldots \;\wedge\; z' = z)$$

Well-definedness of Expression

Let $\mathbf{WD}(e)$ be a predicate which is true in just those circumstances in which e can be successfully evaluated

$$\mathbf{WD}(x) \;=\; true$$

$$\mathbf{WD}(e1 + e2) \;=\; \mathbf{WD}(e1) \wedge \mathbf{WD}(e2)$$

$$\mathbf{WD}(e1/e2) \;=\; \mathbf{WD}(e1) \wedge \mathbf{WD}(e2) \wedge (e2 \neq 0)$$

Define

$$(x := e) \;=_{df}\; \mathbf{WD}(e) \vdash (x' = e \;\wedge\; \ldots \;\wedge\; z' = z)$$

Algebraic Laws of Assignment

We have to reestablish the following laws

$(asgn-1)\ x := e \ = \ x, y := e, y$

$(asgn-2)\ \ (x, y, z := e, f, g) \ = \ (y, x, z := f, e, g)$

$(asgn-3)\ \ (x := e\,;\, x := f(x)) \ = \ x := f(e)$

$(asgn-4)\ \ x := e\ ;(P \triangleleft b(x) \triangleright Q) \ =$

$$(x := e; P) \triangleleft b(e) \triangleright (x := e; Q)$$

$(asgn-5)\ \ P; skip \ = \ P \ = \ skip; P$

Variable Declaration

To introduce a new variable x we use the form of *declaration* **var** x which permits the variable x to be used in the portion of the program that follows it.

The complementary operation (called *undeclaration*) takes the form **end** x and terminates the region of permitted use of x. The portion of program Q in which a variable x may be used is called its *scope*; it is bracketed on the left and on the right by the declaration and undeclaration

$$\textbf{var}\, x;\ Q;\ \textbf{end}\, x$$

Definition of Declaration

Let A be an alphabet which includes x and x'. Then

$$\begin{aligned}
\mathbf{var}\, x &=_{df} \exists x \bullet skip_A \\
\mathbf{end}\, x &=_{df} \exists x' \bullet skip_A \\
\alpha(\mathbf{var}\, x) &=_{df} A \setminus \{x\} \\
\alpha(\mathbf{end}\, x) &=_{df} A \setminus \{x'\}
\end{aligned}$$

Note that the alphabet constraints forbid the redeclaration of a variable within its own scope. For example, $\mathbf{var}\, x; \mathbf{var}\, x$ is disallowed because

$$x' \in OUT\alpha(\mathbf{var}\, x) \quad \text{but} \quad x \notin IN\alpha(\mathbf{var}\, x)$$

Declaration vs Existential Quantifier

Declaration and undeclaration act exactly like existential quantification over their scope

$$\mathbf{var}\, x \,;\, Q \;=\; \exists x \bullet Q$$

$$Q \,;\, \mathbf{end}\, x \;=\; \exists x' \bullet Q$$

As a result, the algebraic laws for declaration closely match those for existential quantification.

Algebraic Laws

Both declaration and undeclaration are commutative.

(**var-1**) $(\mathbf{var}\, x;\ \mathbf{var}\, y) \ = \ (\mathbf{var}\, y;\ \mathbf{var}\, x)$

(**var-2**) $(\mathbf{end}\, x;\ \mathbf{end}\, y) \ = \ (\mathbf{end}\, y; \mathbf{end}\, x)$

(**var-3**) $(\mathbf{var}\, x;\ \mathbf{end}\, y) \ = \ (\mathbf{end}\, y;\ \mathbf{var}\, x)$

provided that x and y are distinct.

(**var-4**) If x is not free in b, then

$$\mathbf{var}\, x;\ (P \lhd b \rhd Q) \ = \ (\mathbf{var}\, x;\ P) \lhd b \rhd (\mathbf{var}\, x; Q)$$

$$\mathbf{end}\, x;\ (P \lhd b \rhd Q) \ = \ (\mathbf{end}\, x;\ P) \lhd b \rhd (\mathbf{end}\, x;\ Q)$$

Composition of declaration and undeclaration

var x followed by **end** x has no effect whatsoever.

(**var-5**) **var** x; **end** $x \;=\; skip$

The sequential composition of **end** x with **var** x has no effect whenever it is followed by an update of x that does not rely on the previous value ofd x

(**var-6**) (**end** x; **var** $x := e$) $\;=\;$ $(x := e)$

provided that x does not occur in e.

Assignment to a variable just before the end of its scope is irrelevant.

(**var-7**) $(x := e$; **end** $x) \;=\;$ **end** x

Example: Compilation of assignments

Many computers can execute only the simplest assignment, with just two or three operands, and most of these operands must be selected from among the available machine registers (such as a, b, c).

$$\begin{aligned}
\textbf{effect}(load\, x) &= a, b := x, a \\
\textbf{effect}(store\, x) &= x, a := a, b \\
\textbf{effect}(add) &= a := a + b \\
\textbf{effect}(subtract) &= a := a - b \\
\textbf{effect}(multiply) &= a := a \times b
\end{aligned}$$

Machine Code for $(x := y \times z + w)$

var $a,\ b$	$a,\ b := y,\ a$	$load\, y$
	$a,\ b := z,\ a$	$load\, z$
	$a := a \times b$	$multiply$
	$a,\ b := w,\ a$	$load\, w$
	$a := a + b$	add
	$x,\ a := a,\ b$	$store\, x$

Unit 5: Implementation

Machine Language

- Alphabet: Machine state
- Operations: Instruction set

Machine State

1. A code memory m occupied can be modelled as a function from addresses to machine instructions

$$m : rom \rightarrow instruction$$

Once a machine code is stored in m, any update on m is forbidden.

2. A data memory M maps addresses to integers:

$$M : ram \rightarrow INT$$

The state of M can be updated by executing a machine instruction.

Remark

In practice, both m and M are part of storage provided by the machine. It is the resposibility of the designer of a compiler to ensure that the storage is properly devided to accomodate both target code and data of a sourse program.

Machine State (Cont'd)

3. A program pointer $P : rom$ always points to the current instruction. By updating the contents of P the computer can choose to execute a specific branch of the machine code.

4. A stack of data registers

$$A, B, C : INT$$

are seen as the extension of data memory.

State Update

Machine instructions are defined as assignments that update the state of the data memory and stack of registers.

$ldc(w)$ has a constant w as its operand, and its execution pushes w into the stack of registers

$$ldc(w) =_{df} A, B, C, P := w, A, B, P + 1$$

$ldl(w)$ pushes the contents of $M[w]$ to the stack of registers

$$ldl(w) =_{df} A, B, C, P := M[w], A, B, P + 1$$

State Update (Cont'd)

$stl(w)$ sends the contents of A back into the data memory location w

$$stl(w) =_{df} M[w],\ P := A,\ P + 1$$

$swap(R_1,\ R_2)$ swaps the contents of the data registers R_1 and R_2

$$swap(R_1,\ R_2) =_{df} R_1,\ R_2 := R_2,\ R_1$$

Evaluation of Expressions

$$
\begin{aligned}
add &=_{df} A,\, P := B + A,\, P + 1 \\
sub &=_{df} A,\, P := B - A,\, P + 1 \\
mul &=_{df} A,\, P := B \times A,\, P + 1 \\
divi &=_{df} A,\, P := A/B,\, P + 1 \\
or &=_{df} A,\, P := 0,\, P + 1 \lhd A = 0 \wedge B = 0 \rhd \\
&\qquad A,\, P := 1,\, P + 1 \\
and &=_{df} A,\, P := 1,\, P + 1 \lhd A = 1 \wedge B = 1 \rhd \\
&\qquad A,\, P := 0,\, P + 1
\end{aligned}
$$

Evaluation of Expressions (Cont'd)

$$
\begin{aligned}
not &=_{df} A,\, P := 0,\, P+1 \lhd A = 1 \rhd \\
&\qquad A,\, P := 1,\, P+1 \\
eqc(w) &=_{df} A,\, P := 1,\, P+1 \lhd A = w \rhd \\
&\qquad A,\, P : 0,\, P+1 \\
gt &=_{df} A,\, P := 1,\, P+1 \lhd B > A \rhd \\
&\qquad A,\, P := 0,\, P+1
\end{aligned}
$$

Machine Programs

Suppose a machine program is stored in the memory m between locations s and f. Its behaviour is the combined effect of running the sequence of machine instructions

$$\mathcal{I}(s,\ f,\ m)\ =_{df} \left(\begin{array}{l} P := s \\ (s \leq P < f)\ *\ m[P] \\ (P = f)_{\perp} \end{array} \right)$$

Symbol Table

At compiler time there are a number of items to which storage must be allocated in order to execute the target code. In our case we asume that the target code always resides in m, and the program variables are stored in the data memory M.

We use Ψ to denote a symbol table which maps each program variables to an address of the data memory M. Because no location can be allocated to two identifiers, Ψ has to be injective.

Linking Program State with Machine State

Define

$$\hat{\Psi} \;=_{df}\; \left(\begin{array}{l} \textbf{var}\, x,\, ..,\, z\,; \\ x,\, ..\, z \;:=\; M[\Psi x],\, ...,\, M[\Psi z]\,; \\ \textbf{end}\, M,\, A,\, B,\, C \end{array} \right)$$

$$\hat{\Psi}^{-1} \;=_{df}\; \left(\begin{array}{l} \textbf{var}\; M,\, A,\, B,\, C\,; \\ M[\Psi x],\, ...,\, M[\Psi z] \;:=\; x,\, ..,\, z\,; \\ \textbf{end}\; x,\, ..,\, z \end{array} \right)$$

Valid Implementation

A program state can be recovered after it is converted to a machine state and then retrieved back:

Theorem

(1) $\hat{\Psi}^{-1}; \hat{\Psi} \;=\; skip$

(2) $\hat{\Psi}\,;\, \hat{\Psi}^{-1} \sqsubseteq skip$

A machine program stored in m is regarded as a realisation of Q if

$$\hat{\Psi}\,;\, Q \sqsubseteq \mathcal{I}(s,\, f,\, m)\,;\, \hat{\Psi}^{-1}$$

Weakest Specification of Valid Implementation

For any source program Q, we define a *machine program* which acts like Q except that it operates on the data memory M:

$$[\Psi](Q) \;=_{df}\; \hat{\Psi}\,;\, Q\,;\, \hat{\Psi}^{-1}$$

Theorem

$(\hat{\Psi}\,;\, Q) \sqsubseteq (\mathcal{I}(s,\, f,\, m)\,;\, \hat{\Psi}^{-1})$ **iff** $[\Psi](Q) \sqsubseteq \mathcal{I}(s,\, f,\, m)$

Proof

$$
\begin{array}{lll}
 & (\hat{\Psi}\,;\,Q) \sqsubseteq (\mathcal{I}(s,\,f,\,m);\hat{\Psi}) & \{;\ is\ monotonic\} \\
\Longrightarrow & (\hat{\Psi};Q;\hat{\Psi}^{-1}) \sqsubseteq & \\
 & (\mathcal{I}(s,\,f,\,m);\hat{\Psi};\hat{\Psi}^{-1}) & \{\hat{\Psi};\hat{\Psi}^{-1} \sqsubseteq skip\} \\
\Longrightarrow & [\Psi](Q) \sqsubseteq \mathcal{I}(s,\,f,\,m) & \{;\ is\ monotonic\} \\
\Longrightarrow & ([\Psi](Q);\hat{\Psi}) \sqsubseteq (\mathcal{I}(s,\,f,\,m)\,;\,\hat{\Psi}) & \{\hat{\Psi}^{-1};\hat{\Psi} = skip\} \\
\Longrightarrow & (\hat{\Psi}\,;\,Q) \sqsubseteq (\mathcal{I}(s,\,f,\,m);\hat{\Psi}) &
\end{array}
$$

Implementation of Assignment

Define $e_\Psi \ =_{df} \ e[M[\Psi x][/x, .., M[\Psi z]/x]$

Theorem $[\Psi](x := e) \sqsubseteq M[\Psi x] := e_\Psi$

Proof

$$
\begin{array}{lll}
 & LHS & \{\mathbf{end}\, M\,;\, x := e \;=\; x := e;\, \mathbf{end}\, M\} \\
= & \mathbf{var}\, x, .., z\,; & \\
 & x, .., z := M[\Psi x], .., M[\Psi z]\,; & \\
 & x := e\,;\, \mathbf{end}\, M,\, A,\, B,\, C\,;\, \hat{\Psi}^{-1} & \{merge\ assignment\} \\
\sqsubseteq & \mathbf{var}\, x, .., z \;:=\; e_\Psi, .., M[\Psi z] & \\
 & M[\Psi x], .., M[\Psi z] := x, .., z\,;\, \mathbf{end}\, x, .., z & \{merge\ assignment\} \\
\sqsubseteq & \mathbf{var}\, x, .., z := e_\Psi, .., M[\Psi z] & \\
 & \mathbf{end}\, x,\, .., z\,; & \\
 & M[\Psi x], .., M[\Psi z] := e_\Psi, .., M[\Psi z] & \{\mathbf{var} x := e; \mathbf{end}\, x \;=\; skip\} \\
= & M[\Psi x] := e_\Psi &
\end{array}
$$

Implementation of Sequential Composition

Theorem $[\Psi](Q;R) \;=\; [\Psi](Q);[\Psi](R)$

Proof

$$
\begin{array}{lll}
 & LHS & \{def\ of\ [\Psi]\} \\
= & \hat{\Psi}^{-1};Q;R;\hat{\Psi} & \{\hat{\Psi}^{-1};\hat{\Psi} \;=\; skip\} \\
= & \hat{\Psi};Q;\hat{\Psi}^{-1};\hat{\Psi};R;\hat{\Psi}^{-1} & \{def\ of\ [\Psi]\} \\
= & RHS &
\end{array}
$$

Implementation of Conditional

Theorem $[\Psi](Q \lhd b \rhd R) \;=\; [\Psi](Q) \lhd b_\Psi \rhd [\Psi](R)$

Proof

LHS $\qquad\qquad \{(Q \lhd b \rhd R); W = (Q; W) \lhd b \rhd (R; W)\}$

$= \quad \hat{\Psi}; (Q; \hat{\Psi}^{-1}) \lhd b \rhd (R; \hat{\Psi}^{-1}) \qquad \{\mathbf{end}\, w; (Q \lhd b \rhd R) \;=$

$(\mathbf{end}\, w\,;\, Q) \lhd b \rhd (\mathbf{end}\, w\;\; R)\}$

$= \quad \mathbf{var}\; x, \,..\, z\,;$

$(x, .., z := M[\Psi x], .., M[\Psi z];$

$\mathbf{end}\, M,\, A,\, B,\, C,\, P;\, Q\,;\, \hat{\Psi}^{-1})$

$\lhd b_\Psi \rhd$

$(x, .., z := M[\Psi x], .., M[\Psi z]; \qquad \mathbf{var}\, w; (Q \lhd b \rhd R) \;=$

$\mathbf{end}\, M,\, A,\, B,\, C,\, P;\, R\,;\, \hat{\Psi}^{-1}) \qquad (\mathbf{var}\, w\,;\, Q) \lhd b \rhd (\mathbf{var}\, w\;\; R)\}$

$= \quad RHS$

Implementation of Iteration

Theorem $[\Psi](b * Q) \sqsubseteq b_\Psi * [\Psi](Q)$

Proof $(\hat{\Psi}^{-1}; RHS; \hat{\Psi})$ $\{(Q \lhd b \rhd R); W = (Q; W) \lhd b \rhd (R; W)\}$

$= \quad \hat{\Psi}^{-1};$ $\{x := e; \lhd Q \lhd b \rhd R = (x := e; Q) \lhd b[e/x] \rhd (x := e; R)\}$

$([\Psi](Q); RHS; \hat{\Psi}) \lhd b_\Psi \rhd \hat{\Psi}$

$= \quad (\hat{\Psi}^{-1}; [\Psi](Q); RHS; \hat{\Psi})$

$\lhd b \rhd (\hat{\Psi}^{-1}; \hat{\Psi})$ $\{\hat{\Psi}^{-1}; \hat{\Psi} = skip\}$

$= \quad (Q; (\hat{\Psi}^{-1}; RHS; \hat{\Psi})) \lhd b \rhd skip$

which implies

$(\hat{\Psi}; RHS; \hat{\Psi}^{-1} \sqsupseteq b * Q)$

$\Longrightarrow \quad (RHS \sqsupseteq \hat{\Psi}^{-1}; (b * Q); \hat{\Psi})$

Empty Machine Program

The empty machine program is the unit of sequential composition.

Theorem

$\mathcal{I}(s,\, s,\, m);\mathcal{I}(s,\, f,\, m) \;=\; \mathcal{I}(s,\, f,\, m) \;=\; \mathcal{I}(s,\, f,\, m);\mathcal{I}(f,\, f,\, m)$

Proof $\mathcal{I}(s,\, s,\, m)$ $\{\text{Def of } \mathcal{I}\}$

$= \quad P := s\,;\, (s \le P < s) * m[P]\,;\, (P = s)_{\perp}$ $\{false * Q \;=\; skip\}$

$= \quad P := s\,;\, (P = s)_{\perp}$ $\{Q \lhd true \rhd R \;=\; Q\}$

$= \quad P := s$

Entry of Machine Code

Theorem If $s \leq j < f$, then

$\mathcal{I}(j,\, f,\, m) \sqsubseteq\ P := j\,;\, (s \leq P < f) * m[P]\,;\, (P = f)_\perp$

Proof

RHS $\quad \{(b \vee c) * Q\ =\ b * Q\,;\, (b \vee c) * Q\}$

$=$ $\quad P := j\,;\, (j \leq P < f) * m[P]\,;$

$\quad (s \leq P < f) * m[P]\,;\, (P = f)_\perp \quad \{(P = f)_\perp = (P = f)_\perp; (P; = f)\}$

$\sqsupseteq$ $\quad \mathcal{I}(j,\, f,\, m)\,;\, (P := f)\,;$

$\quad (s \leq P < f) * m[P]\,;\, (P = f)_\perp \quad \{false * Q\ =\ skip\}$

$=$ $\quad \mathcal{I}(j,\, f,\, m); P := f; (P = f)_\perp \quad \{b_\perp; b_\perp\ =\ b_\perp\}$

$=$ $\quad LHS$

Elimination of Machine Code

Theorem If $s < f$ and $m[s] = jump(f - s - 1)$, then
$\mathcal{I}(s,\, f,\, m) = \mathcal{I}(f,\, f,\, m)$

Proof

$$
\begin{array}{lll}
 & LHS & \{Unfolding\ of\ b * Q\} \\
= & P := s\,;\, m[s]\,; & \\
 & (s \leq P < f) * m[P]\,;\, (P = f)_{\perp} & \{def\ of\ jump(w)\} \\
= & P := s\,;\, P := P + f - s & \\
 & (s \leq P < f) * m[P]\,;\, (P = f)_{\perp} & \{false * Q \;=\; skip\} \\
= & P := f\,;\, (P = f)_{\perp} & \{false * Q \;=\; skip\} \\
= & P := f; & \\
 & (f \leq P < f) * m[P];\, (P = f)_{\perp} &
\end{array}
$$

Void Initial State

If a machine program contains an instruction jumping backwards to its start address, then it does not matter whether the execution starts at its first instruction or that jump instruction.

Theorem If $s \leq j < f$, and $m[j] = jump(s - j - 1)$ then

$$(P := j\,;\, (s \leq P < f) * m[P]) \;=\; (P := s\,;\, (s \leq P < f) * m[P])$$

Proof

	LHS	$\{unfolding\ iteration\}$
$=$	$P := j\,;\, m[j]\,;$	
	$(s \leq P < f) * m[P]$	$\{def\ of\ jump(w)\}$
$=$	$P := j\,;\, P := P + s - j\,;$	
	$(s \leq P < f) * m[P]$	$\{merge\ assignment\}$
	RHS	

Concatenation

The concatenation of two machine programs is a refinement of their sequential composition, since it is easy for the former to materialise its normal exit commitment.

Theorem If $s \leq j \leq f$ then

$$\mathcal{I}(s,\, f,\, m) \sqsupseteq \mathcal{I}(s,\, j,\, m); \mathcal{I}(j,\, f,\, m)$$

Proof

$$
\begin{array}{lll}
 & RHS & \{def\ of\ \mathcal{I}\} \\
= & P := s; (s \le P < j) * m[P]\,;\, (P = j)_{\perp}; & \\
 & P := j; (j \le P < f) * m[P];\, (P = f)_{\perp} & \{Entry\ of\ machine\ code\} \\
\sqsubseteq & P := s; (s \le P < j) * m[P]\,;\, (P = j)_{\perp}; & \\
 & P := j; (s \le P < f) * m[P];\, (P = f)_{\perp} & \{(P = j)_{\perp} \;=\; (P = j)_{\perp}; (P := j\} \\
\sqsubseteq & P := s\,;\, (s \le P \le j) * m[P]\,;\, skip & \\
 & (s \le P < f) * m[P]\,;\, (P = f)_{\perp} & \{b * Q\,;\, (b \vee c) * Q \;=\; (b \vee c) * Q\} \\
= & P := s\,;\, (s \le P < f) * m[P]\,;\, (P = f)_{\perp} & \{def\ of\ \mathcal{I}\} \\
= & LHS &
\end{array}
$$

Conditional Jump vs Conditional

A conditional jump can be used to choose a branch of machine codes according to the initial state of the register A

Theorem If $s < j - 1 < f$ and $m[s] = tjump(j - s - 1)$ and $m[j - 1] = jump(f - j - 1)$, then

$$\mathcal{I}(s,\, f,\, m) \sqsupseteq \left(\begin{array}{c} \mathcal{I}(j,\, f,\, m) \\ \\ \lhd A = 0 \rhd \\ \\ (\mathcal{I}(s+1,\, j-1,\, m); \mathcal{I}(f,\, f,\, m)) \end{array} \right)$$

Proof

$$
\begin{array}{lll}
 & LHS & \{unfolding\ iteration\} \\
= & P := s\,;\, m[s]\,; & \\
 & (s \le P < f) * m[P]\,;\, (P = f)_{\perp} & \{def\ of\ tjump\} \\
= & (P := j\,;\, (s \le P < f) * m[P]\,;\, (P = f)_{\perp}) & \\
 & \lhd A = 0 \rhd\ (P := s + 1; & \\
 & (s \le P < f) * m[P]\,;\, (P = f)_{\perp}) & \{Entry\ of\ machine\ code\} \\
\sqsupseteq & \mathcal{I}(j,\, f,\, m) \lhd A = 0 \rhd\ (P := s + 1\,; & \\
 & (s \le P < j - 1) * m[P]\,; & \\
 & (s \le P < f) * m[P]\,;\, (P = f)_{\perp}) & \{(P = k)_{\perp}; P := j\ =\ (P = k)_{\perp}\} \\
\sqsupseteq & \mathcal{I}(j,\, f,\, m) \lhd A = 0 \rhd\ (P := s + 1\,; & \\
 & (s \le P < j - 1) * m[P]\,;\, (P = j - 1)_{\perp}\,; & \\
 & P := j - 1\,; (s \le P < f) * m[P]\,; (P = f)_{\perp}) & \{Void\ initial\ state\} \\
\sqsupseteq & RHS &
\end{array}
$$

Backward Jump vs Iteration

Theorem Assume that $s < j < f - 1$.

If $\mathcal{I}(s,\, j,\, m) \sqsupseteq (A := 1 \lhd b \rhd A := 0)$

and $m[j] = tjump(f - j - 1)$ and

$m[f - 1] = jump(s - f)$, then

$$\mathcal{I}(s,\, f,\, m) \sqsupseteq$$

$$\mu X \bullet \left(\begin{array}{c} (A := 1,\, ;\, \mathcal{I}(j + 1,\, f - 1,\, m)\, ;\, X) \\ \\ \lhd b \rhd \\ \\ (A,\, P \,:=\, 0,\, f) \end{array} \right)$$

Proof

$$
\begin{array}{lll}
 & LHS & \{b * Q\,;\,(b \vee c) * Q \;=\; (b \vee c) * Q\} \\
\sqsupseteq & \mathcal{I}(s,\,j,\,m)\,;\,(P := j)\,; & \\
 & (s \leq P < f) * m[P]\,;\,(P = f)_{\perp} & \{\mathcal{I}(s,\,j,\,m) \sqsupseteq (A := 1 \lhd b \rhd A := 0)\} \\
\sqsupseteq & \mathcal{I}(s,\,j,\,m)\,; & \\
 & (\mathcal{I}(f,\,f,\,m) \lhd A = 0 \rhd & \\
 & (\mathcal{I}(j+1,\,f-1,\,m); (P := f); & \\
 & (s \leq P < f) * (s,\,f,\,m)\,;\,(P = f)_{\perp}) & \{\textit{Sequential composition}\} \\
= & (A := 1) \lhd b \rhd (A := 0)\,; & \\
 & (\mathcal{I}(f,\,f,\,m) \lhd A = 0 \rhd & \\
 & (\mathcal{I}(j+1,\,f-1,\,m); \mathcal{I}(s,\,f,\,m)) & \{;- \lhd \rhd \textit{ distribuitivity}\} \\
= & (A := 1; \mathcal{I}(j+1,\,f-1,\,m); \mathcal{I}(s,\,f,\,m)) & \\
 & \lhd b \rhd (A,\,P \;:=\; 0,\,f) &
\end{array}
$$

Unit 6 : Design Capturing

Healthiness Condition 1

R makes no prediction on the final values of variables until the program has started.

H1 $R = (ok \Rightarrow R)$

Theorem H1 (Algebraic Representation)

R satisfies **H1 iff** R satisfies the left zero and left unit laws.

Proof of Theorem H1

If R

$$
\begin{aligned}
&= skip; R \\
&= (\neg ok \vee skip); R \\
&= (\neg ok; R) \vee (skip; R) \\
&= (\neg ok; \mathbf{true}; R) \ \vee \ (skip; R) \\
&= (\neg ok; \mathbf{true}) \ \vee \ R \\
&= \neg ok \ \vee \ R
\end{aligned}
$$

Proof of Theorem H1

Only if

$$\begin{aligned} & \mathbf{true}; R \\ = \; & \mathbf{true}; (\neg ok \vee R) \\ = \; & (\mathbf{true}; \neg ok) \;\vee\; (\mathbf{true}; R) \\ = \; & \mathbf{true} \end{aligned}$$

Healthiness Condition 2

Non-termination is something that is never wanted.

H2 $[R[false/ok'] \Rightarrow R[true/ok']]$

Theorem H2 (Syntactical Representation)

A predicate satisfies the healthiness conditions **H1** and **H2 iff** it is a design.

Proof of Theorem H2

Only if

$$
\begin{aligned}
& R \\
= \;& \neg ok \;\vee\; R \\
= \;& \neg ok \;\vee\; (\neg ok' \wedge R[false/ok']) \vee \\
& (ok' \wedge R[true/ok']) \\
= \;& \neg ok \;\vee\; (\neg ok' \wedge R[false/ok']) \vee \\
& (ok' \;\wedge (R[false/ok'] \vee R[true/ok'])) \\
= \;& \neg ok \vee R[false/ok'] \vee \\
& (ok' \wedge R[true/ok']) \\
= \;& \neg R[false/ok'] \;\vdash\; R[true/ok']
\end{aligned}
$$

Healthiness Condition 3

If R fails to terminate, then its behaviour is unpredictable

H3 $R \;=\; R; skip$

Theorem H3

$(P \vdash Q)$ satisfies **H3 iff** $P \;=\; \forall v' \bullet P$

Proof of Theorem H3

Proof $(P \vdash Q); skip$

$$= (P \vdash Q); (\mathbf{true} \vdash (v' = v))$$

$$= \neg(\neg P; \mathbf{true}) \vdash Q$$

which implies that $P \vdash Q$ satisfies $H3$ **iff**

$$P = \neg(\neg P; \mathbf{true}) = \forall v' \bullet P$$

An assumption P satisfying the above condition is called a *precondition* which it only refers the initial values of variables.

Healthiness Condition 4

If the precondition of a program is satisfied, the program is required to deliver final values for the program variables.

H4 $R; \mathbf{true} = \mathbf{true}$

Theorem H4

$b \vdash Q$ satisfies **H4 iff** $[b \Rightarrow (Q; \mathbf{true})]$

Proof of Theorem H4

Proof $(b \vdash Q); \mathbf{true}$

$$= \quad (b \wedge \neg(Q; \mathbf{true})) \vdash (Q; \mathbf{true})$$

which implies that $b \vdash Q$ satisfies **H4 iff**

$$b \wedge \neg(Q; \mathbf{true}) \;=\; \mathbf{false}$$

or equivalently

$$[b \Rightarrow (Q; \mathbf{true})]$$

Closure of Healthy Predicates

Theorem

If both P and Q satisfy **Hi**, so are $P;Q$, $P \sqcap Q$ and $P \lhd b \rhd Q$.

Proof of Closure Theorem

Proof Let P and Q in **H1**. We are going to show that $P;Q$ satisfies left unit and left zero laws.

$$
\begin{aligned}
& \mathbf{true};(P;Q) \\
= \;& \mathbf{true};Q \\
= \;& \mathbf{true}
\end{aligned}
$$

$$
\begin{aligned}
& skip;(P;Q) \\
= \;& (skip;P);Q \\
= \;& P;Q
\end{aligned}
$$

Mapping Relations To Designs

A mapping $\mathcal{M}$ from relations to designs is a *retraction* if it satisfies the following conditions

- **Monotonic**: $P_1 \sqsupseteq P_2$ implies $\mathcal{M}(P_1) \sqsupseteq \mathcal{M}(P_2)$
- **Idempotent**: $\mathcal{M}^2(P) = \mathcal{M}(P)$
- **Weakening**: $P \sqsupseteq \mathcal{M}(P)$

We are going to show the healthiness conditions can be characterised by the corresponding retraction.

H1-healthy Predicates

Define

$$H1(P) \quad =_{df} \quad (ok \Rightarrow P)$$

Theorem

$H1$ is monotonic, idempotent and weakening

$$P \sqsupseteq H1(P) \qquad \text{for all } P$$

Theorem

P satisfies the healthiness condition **H1 iff** $P \;=\; H1(P)$

Homomorphism

Theorem

$H1(P \textbf{ op } Q) \sqsupseteq (H1(P) \textbf{ op } H1(Q))$

Proof

$$H1(P \textbf{ op } Q)$$

$$\sqsupseteq \; H1(H1(P) \textbf{ op } H1(Q))$$

$$= \; H1(P) \textbf{ op } H1(Q)$$

Closure of Recursion

Theorem

$\mu X \bullet F(X)$ satisfies the healthiness condition **H1**, where F is formed by programming combinators.

Proof $H1(\mu X \bullet F(X))$

$= \quad H1(F(\mu X \bullet F(X)))$

$\sqsupseteq \quad F(H1(\mu X \bullet F(X)))$

which implies that $H1(\mu X \bullet F(X)) \sqsupseteq \mu X \bullet F(X)$

Because $H1$ is weakening one has

$\mu X \bullet F(X) \ \sqsupseteq H1(\mu X \bullet F(X))$

H2-healthy Predicates

Define

$$H2(P) \quad =_{df} \quad P; ((ok \Rightarrow ok') \ \wedge \ (x' = x \ \wedge \ \ldots \ \wedge \ z' = z))$$

Theorem

(1) $H2$ is monotonic, idempotent and weakening.

(2) P satisfies **H2 iff** $P \ = \ H2(P)$

(3) $H2(P \ \mathbf{op} \ Q) \ \sqsupseteq \ (H2(P) \ \mathbf{op} \ H2(Q))$

(4) $\mu X \bullet F(X)$ satisfies **H2**, where F is formed by programming combinators.

H3-healthy Predicates

Define

$$H3(P) \quad =_{df} \quad (P; skip)$$

Theorem

(1) $H3$ is monotonic, idempotent and weakening.

(2) P satisfies **H3 iff** $P \;=\; H3(P)$

(3) $H3(P \textbf{ op } Q) \;\sqsupseteq\; (H3(P) \textbf{ op } H3(Q))$

(4) $\mu X \bullet F(X)$ satisfies **H3**, where F is formed by programming combinators.

H4-healthy predicates

Define

$$H4(P) \quad =_{df} \quad ((P; \mathbf{true}) \Rightarrow P)$$

Theorem

(1) $H4$ is idempotent and weakening.

(2) P satisfies **H4 iff** $P \;=\; H4(P)$

(3) $H4(P \;\mathbf{op}\; Q) \;\sqsupseteq\; (H4(P) \;\mathbf{op}\; H4(Q))$

(4) $\mu X \bullet F(X)$ satisfies **H4**, where F is formed by programming combinators.

Unit 7: Linking Theories

Subset theories

If $\mathbf{T}$ is a subset of $\mathbf{S}$, there is always a link from $\mathbf{T}$ to $\mathbf{S}$

$$R \quad =_{df} \quad \lambda X \in \mathbf{T} \bullet \; X$$

A more interesting mapping is to be sought in the other direction: it is an endofunction

$$L \; : \; \mathbf{S} \rightarrow \mathbf{S}$$

which ranges over all the members of $\mathbf{T}$

$$range(L) \quad = \quad \mathbf{T}$$

Examples

$$id_{\mathbf{S}} \quad =_{df} \quad \lambda X \in \mathbf{S} \bullet \; X$$

$$or_P \quad =_{df} \quad \lambda X \in \mathbf{S} \bullet \; (P \vee X)$$

$$pre_P \quad =_{df} \quad \lambda X \in \mathbf{S} \bullet \; (P; X)$$

$$post_Q \quad =_{df} \quad \lambda X \in \mathbf{S} \bullet \; (X; Q)$$

Endofunctions

Theorem (range of an idempotent)

If L is idempotent, then $X = L(X)$ **iff** X lies in the range of L.

Theorem (monotonic endofunctions)

If both $L1$ and $L2$ are monotonic, so is their composition $L1 \circ L2$.

Theorem (weakening endofunctions)

If both $L1$ and $L2$ are monotonic weakening, so is their composition $L1 \circ L2$.

Closure of Idempotent

Theorem (idempotent endofunctions)
If $L1$ and $L2$ are idempotent and satisfy

$$L1 \circ L2 \;=\; L2 \circ L1$$

then $L1 \circ L2$ is an idempotent.

Examples

(1) $\Psi \;=_{df}\; (H1 \circ H2 \circ H3)$ is an idempotent.

(2) $\Psi \circ H4$ is also idempotent.

Link and Recursion

L is a *link* if it is idempotent and weakening.

Let $\mathbf{S}$ be a complete lattice, and $\mathbf{T} \subseteq \mathbf{S}$ also a complete lattice. Assume that F is a monotonic mapping on $\mathbf{S}$, and closed in $\mathbf{T}$.

Theorem

(1) If $L : \mathbf{S} \to \mathbf{S}$ is a link and satisfies $L \circ F \sqsupseteq F \circ L$, then

$$\mu_{\mathbf{S}} X \bullet F(X) \quad = \quad \mu_{\mathbf{T}} X \bullet F(X)$$

(2) If L is monotonic idempotent, and satisfies $L \circ F \ \sqsupseteq \ F \circ L$, then

$$L(\mu_{\mathbf{S}} X \bullet F(X)) \ = \ \mu_{\mathbf{T}} X \bullet F(X)$$

Proof of (1)

$$
\begin{aligned}
& L(\mu_{\mathbf{S}} X \bullet F(X)) \\
= \;& L(F(\mu_{\mathbf{S}} X \bullet F(X))) \\
\sqsupseteq \;& F(L(\mu_{\mathbf{S}} X \bullet F(X)))
\end{aligned}
$$

which and $L(\mu_{\mathbf{S}} X \bullet F(X)) \in \mathbf{T}$ imply that

$$\mu_{\mathbf{T}} X \bullet F(X) \;\sqsubseteq\; L(\mu_{\mathbf{S}} X \bullet F(X)) \;\sqsubseteq \mu_{\mathbf{S}} X \bullet F(X)$$

Proof of (2)

$$\mu_{\mathbf{T}} X \bullet F(X) \;\sqsupseteq\; \mu_{\mathbf{S}} X \bullet F(X)$$

$$\Rightarrow\; L(\mu_{\mathbf{T}} X \bullet F(X)) \;\sqsupseteq\; L(\mu_{\mathbf{S}} X \bullet F(X))$$

$$\Rightarrow\; \mu_{\mathbf{T}} X \bullet F(X) \;\sqsupseteq\; L(\mu_{\mathbf{S}} X \bullet F(X))$$

$$\mu_{\mathbf{S}} X \bullet F(X) \;=\; F(\mu_{\mathbf{S}} X \bullet F(X))$$

$$\Rightarrow\; L(\mu_{\mathbf{S}} X \bullet F(X)) \;=\; L(F(\mu_{\mathbf{S}} X \bullet F(X)))$$

$$\Rightarrow\; L(\mu_{\mathbf{S}} X \bullet F(X)) \;\sqsupseteq\; F(L(\mu_{\mathbf{S}} X \bullet F(X)))$$

$$\Rightarrow\; L(\mu_{\mathbf{S}} X \bullet F(X)) \;\sqsupseteq\; \mu_{\mathbf{T}} X \bullet F(X)$$

Goal of a Subset Theory

A frequent goal in the definition of a subset theory is to ensure that the validity on that subset of certain additional algebraic laws. For example, the desired law may be of the form

$$X \;=\; L(X)$$

Now if L happens to be idempotent, the goal is simply achieved: just take the link L itself and define

$$\mathbf{T} \;=_{df}\; range(L)$$

Goal of a Subset Theory

As an additional advantage, it is frequently found that other useful laws become true in $\mathbf{T}$.

For example, let

$$L =_{df} H1 = \lambda X \bullet (ok \Rightarrow X)$$

In the subset $\mathbf{T} =_{df} range(L)$, both left zero law and the left unit law are valid.

Unit 8: Galois Connection

Galois Connection

Let $(\mathbf{S},\ \sqsupseteq_{\mathbf{S}})$ and $(\mathbf{T},\ \sqsupseteq_{\mathbf{T}})$ be complete lattices.

$L : \mathbf{S} \to \mathbf{T}$ and $R : \mathbf{T} \to \mathbf{S}$ form a *Galois connection*

if for all $X \in \mathbf{S}$ and all $Y \in \mathbf{T}$

$$(L(X)\ \sqsupseteq_{\mathbf{T}}\ Y) \quad \textbf{iff} \quad (X\ \sqsupseteq_{\mathbf{S}}\ R(Y))$$

Examples

(1) $(P \wedge X) \sqsupseteq Y$ **iff** $X \sqsupseteq (P \Rightarrow Y)$

(2) (weakest post-specification)

$(P; X) \sqsupseteq Y$ **iff**

$X \sqsupseteq \forall m \bullet (P(m, v) \Rightarrow Y(m, v'))$

(3) (weakest prespecification)

$(X; Q) \sqsupseteq Y$ **iff**

$X \sqsupseteq \forall m \bullet (Q(v', m) \Rightarrow Y(v, m))$

Galois Connection and Inverse

Theorem

If L and R are monotonic, then they form a Galois connection **iff**

$$(L \circ R) \ \sqsupseteq_{\mathbf{T}} \ id_{\mathbf{T}} \quad \text{and} \quad (R \circ L) \ \sqsubseteq_{\mathbf{S}} \ id_{\mathbf{S}}$$

Theorem

If $(L_i,\ R_i)$ for $i = 1,\ 2$ are Galois connections, then

$$L_1 \ = \ L_2 \quad \textbf{iff} \quad R_1 \ = \ R_2$$

Proof

Assume that $L_1 = L_2$

$$
\begin{aligned}
& X \sqsupseteq R_1(Y) \\
\equiv\ & L_1(X) \sqsupseteq Y \\
\equiv\ & L_2(X) \sqsupseteq Y \\
\equiv\ & X \sqsupseteq R_2(Y)
\end{aligned}
$$

Disjunctivity and Conjunctivity

$L : \mathbf{S} \to \mathbf{T}$ is *universally disjunctive* if for all sets U

$$L(\sqcap_{\mathbf{S}}\ U) \ = \ \sqcap_{\mathbf{T}}\ \{L(X) \mid X \in U\}$$

L is *disjunctive* if the above equation holds for all nonempty sets U.

Examples

(1) $\lambda X \bullet (P; X)$ is fully disjunctive.

(2) $\lambda X \bullet (X; Q)$ is fully disjunctive.

(3) $\lambda X \bullet (P \vee X)$ is disjunctive, but not fully disjunctive.

Galois Connection vs Disjunctivity

Theorem

There exists R such that (L, R) is a Galois connection **iff** L is universally disjunctive.

Proof

$$
\begin{aligned}
(\Rightarrow) \qquad & L(\sqcap_{\mathbf{S}}\, U) \;\; \sqsupseteq_{\mathbf{T}} \;\; Y \\
\equiv \;\; & (\sqcap_{\mathbf{S}}\, U) \;\; \sqsupseteq_{\mathbf{S}} \;\; R(Y) \\
\equiv \;\; & \forall X \in U \bullet \; (X \;\; \sqsupseteq_{\mathbf{S}} \;\; R(Y)) \\
\equiv \;\; & \forall X \in U \bullet \; (L(X) \;\; \sqsupseteq_{\mathbf{T}} \;\; Y) \\
\equiv \;\; & \sqcap_{\mathbf{T}}\{L(X) \mid X \in U\} \;\; \sqsupseteq_{\mathbf{T}} \;\; Y
\end{aligned}
$$

Proof (cont'd)

Let $R(Q) =_{df} \sqcap_{\mathbf{S}}\{X \mid L(X) \sqsupseteq_{\mathbf{T}} Q\}$

$(\Leftarrow)$ $\quad L(P) \sqsupseteq_{\mathbf{T}} Q$

$\equiv$ $\quad P \in \{X \mid L(X) \sqsupseteq_{\mathbf{T}} Q\}$

$\Rightarrow$ $\quad P \sqsupseteq_{\mathbf{S}} R(Q)$

$\Rightarrow$ $\quad L(P) \sqsupseteq_{\mathbf{T}} L(R(Q))$

$\equiv$ $\quad L(P) \sqsupseteq_{\mathbf{T}} \sqcap_{\mathbf{T}}\{L(X) \mid L(X) \sqsupseteq Q\}$

$\Rightarrow$ $\quad L(P) \sqsupseteq_{\mathbf{T}} Q$

Simulation and Galois Connection

Let D and U be designs satisfying

$$(D;U) \sqsupseteq \mathit{skip} \quad \textbf{and} \quad \mathit{skip} \sqsupseteq (U;D)$$

In this case, D is called a *simulation* and U a *co-simulation*.

Theorem

If (D, U) is a pair of simulations, then

$$L(X) =_{df} (D;X;U) \qquad \text{and} \qquad R(Y) =_{df} (U;Y;D)$$

form a Galois connection over designs.

Galois Connection and Recursion

Theorem

Let F and G be monotonic, and let (L, R) be a Galois connection.

If $R \circ F = G \circ R$, then

$$R(\mu X \bullet F(X)) = \mu X \bullet G(X)$$

Unit 9: Algebraic Representation

The Notations of the Language

$$
\begin{aligned}
&< program > \quad ::= \quad \mathbf{true} \,|\, skip \\
&| \; < variable\ list > := < exp \ \ list > \\
&| \; < program > \lhd bool - exp \rhd < program > \\
&| \; < program > ; < program > \\
&| \; < program > \sqcap < program > \\
&| \; < recursive \ \ identifier > \\
&| \, \mu < recursive \ \ identifier > \bullet < program >
\end{aligned}
$$

Assignment Normal Form

A total assignment is the first of normal forms

$$x,\, y,\, \ldots,\, z \;:=\; e,\, f,\, \ldots,\, g$$

L1 $(x,\, y \ldots := e,\, f, \ldots) \;=\; (x,\, y,\, \ldots,\, z := e,\, f,\, \ldots,\, z)$

L2 $(x, \ldots,\, z,\, y, \ldots := \; e, \ldots,\, g,\, f, \ldots) \;=$
$(x, \ldots,\, y,\, z, \ldots := \; e, \ldots,\, f,\, g, \ldots)$

L3 $(v := g\,;\, v := f(v)) \;=\; (v := f(g))$

L4 $(v := f) \lhd b \rhd (v := g) \;=\; (v := (f \lhd b \rhd g))$

L5 $(v := f) \;=\; (v := g) \quad \textbf{iff} \quad [f = g]$

Non-deterministic Normal Form

$$(v := f) \sqcap \ldots \sqcap (v := h)$$

Let A and B be finite sets of total assignments

L1 $(\sqcap A) \ \sqcap \ (\sqcap B) \ = \ \sqcap \ (A \cup B)$

L2 $(\sqcap A) \lhd b \rhd (\sqcap B) \ = \ \sqcap \ \{v := g \lhd b \rhd h \mid (v := g) \in A \ \wedge \ (v := h) \in B\}$

L3 $(\sqcap A); (\sqcap B) \ = \ \sqcap \ \{(v := g(h)) \mid (v := h) \in A \ \wedge \ (v := g) \in B\}$

L4 $(\sqcap A) \sqsupseteq_{AL} R$ **iff** $\forall (v := e) \in A \bullet ((v := e) \sqsupseteq_{AL} R)$

L5 $(v := e) \sqsupseteq_{AL} (v := g \sqcap \ldots \sqcap v := h)$ **iff** $[e \in \{g, \ldots, h\}]$

Non-termination Normal Form

$$b \ \vee \ (\sqcap A)$$

L1 $(b \vee P) \ \sqcap \ (c \vee Q) \ = \ (b \vee c) \ \vee \ (P \sqcap Q)$

L2 $(b \vee P) \lhd d \rhd (c \vee Q) \ = \ (b \lhd d \rhd c) \ \vee \ (P \lhd d \rhd Q)$

L3 $(b \vee P); (c \vee Q) \ =$

$$(b \ \vee \ \bigvee\{c(e) \mid (v := e) \in P\}) \ \vee \ (P; Q)$$

L4 $(b \vee P) \sqsupseteq_{AL} (c \vee Q)$ **iff** $[b \Rightarrow c] \ \wedge \ (P \sqsupseteq_{AL} (c \vee Q))$

L5 $(v := e) \sqsupseteq_{AL} (c \vee (v := g \sqcap \ldots \sqcap v := h))$ **iff**

$$[c \ \vee \ e \in \{g, \ldots, h\}]$$

Infinite Normal Form

$$S \;=\; \{S_i \mid i \in \mathcal{N}\}$$

where each S_{i+1} is stronger than its predecessor

$$(S_{i+1} \sqsupseteq S_i), \qquad \text{for } i \in \mathcal{N}$$

L1 $(\sqcup S) \sqcap P \;=\; \sqcup\; (S_i \sqcap P)$

L2 $(\sqcup S) \lhd b \rhd P \;=\; \sqcup_i\, (S_i \lhd b \rhd P)$

L3 $(\sqcup S); P \;=\; \sqcup_i\, (S_i; P)$

L4 $P; (\sqcup S) \;=\; \sqcup_i\, (P; S_i)$

Continuity

L5 $(\sqcup S) \sqcap (\sqcup T) \;=\; \sqcup_i \, (S_i \sqcap T_i)$

L6 $(\sqcup S) \lhd b \rhd (\sqcup T) \;=\; \sqcup_i \, (S_i \lhd b \rhd T_i)$

L7 $(\sqcup S) \,;\, (\sqcup T) \;=\; \sqcup_i \, (S_i \,;\, T_i)$

L8 $\mu X \bullet (\sqcup_i S_i(X)) \;=\; \sqcup_i \, (\mu X \bullet S_i(X))$

L9 $\mu X \bullet F(X) \;=\; \sqcup_i \, (F^i(\mathbf{true}))$

where

$$F^0(X) =_{df} \mathbf{true}, \quad \text{and} \quad F^{n+1}(X) =_{df} F(F^n(X))$$

Computability

How to compute the limit $\sqcup(b_n \ \vee \ P_n)$

The execution depends on the property

$$(b_n \vee P_n) \ = \ (b_n \ \vee \ P_{n+k})$$

Let $P \ = \ (v := e_1 \sqcap \ldots \sqcap v := e_n)$
and $Q \ = \ (v := f_1 \sqcap \ldots \sqcap v := f_m)$.

$(b \vee P) \ \leq \ (c \vee Q) \ =_{df}$

$[c \Rightarrow b]$ **and** $[\neg b \Rightarrow (\{e_1, \ldots, e_n\} = \{f_1, \ldots, f_m\})]$

Strong Monotonicity

Theorem

If $X \leq Y$ then $F(X) \leq F(Y)$, for all programming operators F.

Theorem

$\{F^n(\mathbf{true}) \mid n \in \mathcal{N}\}$ is a $\leq$-descending chain.

L1 $\sqcup S \sqsubseteq_{AL} \sqcup T$ **iff** $S_n \sqsubseteq_{AL} (\sqcup T)$ for all n

L2 Let $P = (v := e_1 \sqcap \ldots \sqcap v := e_m)$
and $Q_n = (v := f_1^n \sqcap \ldots \sqcap v := f_k^n)$
$(b \vee P) \sqsubseteq_{AL} \sqcup(c_n \vee Q_n)$ **iff**

$[(\wedge_k c_k) \Rightarrow b]$ **and** $[c_n \vee b \vee (\{f_1^n, \ldots, f_k^n\} \subseteq \{e_1, \ldots, e_m\})]$

Completeness

A reduction to normal form gives a method of testing the truth of any proposed implication between any pair of programs: reduce both of them to normal form, and test whether the inequation satisfies the conditions of previous two laws.

The converse conclusion can also be drawn.

Theorem

$[P \Rightarrow Q]$ **iff** $(P \sqsupseteq_{AL} Q)$

Denotational Semantics

If j is a list of constants, $(state := j); P(state, state')$ describes all possible observations of the final state of an execution of P that starts in initial state j.

The implication

$$[(state := k) \Rightarrow (state := j); P]$$

means the state k is among the possible final state, i.e.

$$P(j,\, k)$$

$[P \Rightarrow Q]$ **iff** $[t \Rightarrow (s; P)]$ implies $[t \Rightarrow (s; Q)]$

for all constant assignment.

Defining Equations

$$\mathcal{R}(\mathbf{true}) \;=\; \{(j,\, k) \mid j,\, k \in STATE\}$$

$$\mathcal{R}(v := e(v)) \;=\; \{(ok,\, v),\, (ok',\, v') \mid ok \Rightarrow (ok' \wedge v' = e(v))\}$$

$$\mathcal{R}(P;Q) \;=\; \{(j,\, k) \mid \exists m \bullet (j,\, m) \in \mathcal{R}(P) \;\wedge\; (m,\, k) \in \mathcal{R}(Q)\}$$

$$\mathcal{R}(P \lhd b \rhd Q) \;=\; \{(j,\, k) \mid (state := j); b \;\wedge\; (j,\, k) \in \mathcal{R}(P) \vee (state := j); \neg b \;\wedge\; (j,\, k) \in \mathcal{R}(Q)\}$$

$$\mathcal{R}(P \sqcap Q) \;=\; \{(j,\, k) \mid (j,\, k) \in \mathcal{R}(P) \;\vee\; (j,\, k) \in \mathcal{R}(Q)\}$$

$$\mathcal{R}(\mu X \bullet F(X)) \;=\; \{(j,\, k) \mid \forall n \in \mathcal{N} \bullet (j,\, k) \in \mathcal{R}(F^n(\mathbf{true}))\}$$

From Algebraic to Denotational Presentation

Define

$$\mathcal{A}(P) \;=_{df}\; \neg d(v) \vdash Q(v,\, v')$$

where

$d(j) \;=\; true$ **iff** $(v := j; P) \;=\;$ **true** is provable, and

$Q(j,\, k) \;=\; true$ **iff** $(v := j; P) \sqsubseteq_{AL} (v := k)$ is provable.

Theorem

$\mathcal{A}$ satisfies the defining equation of $\mathcal{R}$

Unit 10 : An Operational Model

Operational Semantics

An operational semantics suggests a complete set of individual steps which may be taken in the execution

$$m \longrightarrow m'$$

where m and m' are of the form (`s`, `P`)

(1) `s` is a text, defining the data state as a constant assignment to all variables.

(2) `P` is a program text, representing the rest of program that remains to be executed. When this is `II`, there is no more program to be executed.

Algebraic Specification of a Step

$$((\mathtt{s},\ \mathtt{P}) \rightarrow (\mathtt{t},\ \mathtt{Q})) \quad =_{df} \quad ((s; P) \sqsubseteq_{AL} (t; Q))$$

(1) $(\mathtt{s},\ \mathtt{v} := \mathtt{e}) \rightarrow (\mathtt{v} := \mathtt{s}; \mathtt{e},\ \mathtt{II})$

(2) $(\mathtt{s},\ \mathtt{II}; \mathtt{R}) \rightarrow (\mathtt{s},\ \mathtt{R})$

(3) If $(\mathtt{s},\ \mathtt{P}) \rightarrow (\mathtt{t},\ \mathtt{Q})$, then $(\mathtt{s},\ \mathtt{P}; \mathtt{R}) \rightarrow (\mathtt{t},\ \mathtt{Q}; \mathtt{R})$

(4) $(\mathtt{s},\ \mathtt{P} \sqcap \mathtt{Q}) \rightarrow (\mathtt{s},\ \mathtt{P})$

$(\mathtt{s},\ \mathtt{P} \sqcap \mathtt{Q}) \rightarrow (\mathtt{s},\ \mathtt{Q})$

(5) $(\mathtt{s},\ \mathtt{P} \lhd \mathtt{b} \rhd \mathtt{Q}) \rightarrow (\mathtt{s},\ \mathtt{P})$, whenever $s; b$

$(\mathtt{s},\ \mathtt{P} \lhd \mathtt{b} \rhd \mathtt{Q}) \rightarrow (\mathtt{s},\ \mathtt{Q})$, whenever $s; \neg b$

(6) $(\mathtt{s},\ \mu\mathtt{X} \bullet \mathtt{F}(\mathtt{X})) \rightarrow (\mathtt{s},\ \mathtt{F}(\mu\mathtt{X} \bullet \mathtt{F}(\mathtt{X})))$

(7) $(\mathtt{s},\ \mathbf{true}) \rightarrow (\mathtt{s},\ \mathbf{true})$

Correctness

There are two kinds of error which may occur in design of an operational semantics.

(1) There may be too few transitions. As a result, the set of steps is incomplete. For example we omit the rule

$$(\mathtt{s},\ \mathtt{P} \sqcap \mathtt{Q}) \rightarrow (\mathtt{s},\ \mathtt{Q})$$

thereby eliminating non-determinism from the language.

(2) There are too many transitions. For example

$$(\mathtt{s},\ \mathtt{P}) \rightarrow (\mathtt{s},\ \mathtt{P})$$

is entirely consistent since it expresses reflexivity of $\sqsubseteq$. However such rule can lead to an infinite execution.

The Reflexive Transitive Closure

Define $\xrightarrow{*}$ as the reflexive transitive closure of $\rightarrow$, and define

$$\xrightarrow{1} \quad =_{df} \quad \rightarrow$$

$$\xrightarrow{n+1} \quad =_{df} \quad (\xrightarrow{1} \ ; \ \xrightarrow{n})$$

A machine state $(\mathtt{s},\ \mathtt{P})$ is divergent if it can lead to an infinite execution

$$(\mathtt{s},\ \mathtt{P})\uparrow \quad =_{df} \quad \forall n \bullet \exists \mathtt{t},\ \mathtt{Q} \bullet (\mathtt{s},\ \mathtt{P}) \xrightarrow{n} (\mathtt{t},\ \mathtt{Q})$$

Ordering Relation

Define a relation $\sqsubseteq_{STATE}$ over states

$(\mathtt{s},\ \mathtt{P}) \sqsubseteq_{STATE} (\mathtt{t},\ \mathtt{Q}) \ =_{df}$

$(\mathtt{s},\ \mathtt{P})\uparrow \ \vee\ \forall \mathtt{u} \bullet ((\mathtt{t},\ \mathtt{Q}) \xrightarrow{*} (\mathtt{u},\ \mathtt{II})) \Rightarrow ((\mathtt{s},\ \mathtt{P}) \xrightarrow{*} (\mathtt{u},\ \mathtt{II}))$

Define a relation $\sqsubseteq_{OP}$ over program texts by

$$P \sqsubseteq_{OP} Q \ =_{df}\ \forall \mathtt{s} \bullet (\mathtt{s},\ \mathtt{P}) \sqsubseteq_{STATE} (\mathtt{s},\ \mathtt{Q})$$

Theorem

$\sqsubseteq_{OP}$ is a pre-order.

Define $\mathtt{P} \sim \mathtt{Q} \ =_{df}\ (\mathtt{P} \sqsubseteq_{OP} \mathtt{Q})$ **and** $(\mathtt{Q} \sqsubseteq_{OP} \mathtt{P})$

From Operational to Algebraic Semantics

Lemma 1

$(\mathtt{s},\ \mathtt{P}) \uparrow$ **iff** $(s; P) = \mathbf{true}$ is algebraically provable.

Lemma 2

$(\mathtt{s},\ \mathtt{P}) \uparrow$ or $(\mathtt{s},\ \mathtt{P}) \xrightarrow{*} (\mathtt{t},\ \mathtt{II})$ **iff**

$(s; P) \sqsubseteq_{AL} t$ is algebraically provable

Theorem

$\mathtt{P} \sim \mathtt{Q}$ **iff** $P = Q$ is algebraically provable.

From Operational to Denotational Semantics

Define

$$\mathcal{O}(\mathtt{P}) \;=_{df}\; \neg d(v) \vdash Q(v,\, v')$$

where

$$d(j) \;=_{df}\; (\mathtt{v} := \mathtt{j},\ \mathtt{P}) \uparrow$$

$$Q(j,\, k) \;=_{df}\; (\mathtt{v} := \mathtt{j},\ \mathtt{P}) \xrightarrow{*} (\mathtt{v} := \mathtt{k},\ \mathtt{II})$$

Theorem

$\mathcal{O}$ satisfies the defining equation of $\mathcal{R}$.