



Refinement in
Object-Oriented
Development: Sequential
Java

Augusto Sampaio and Paulo Borba (CIn-UFPE, Brazil)

General context and motivation

- Program semantics
 - Operational, denotational, algebraic
- The algebraic approach
 - Properties expressed as (in)equations relating operators
 - No explicit mathematical model
 - Modularity and easy mechanisation
 - What about soundness and completeness?

General context and motivation

The algebraic approach: previous and ongoing work

- Laws of [imperative] programming
- Laws of occam (and CSP) [concurrency and communication]
- Laws of functional programming
- Laws of logic programming
- Compiler design (for software and hardware)
- Hardware/software codesign
- Semantic models for proving the laws ...

Objectives

- Algebraic presentation
 - Laws of OO programming
- Completeness
 - A comprehensive set of laws
- Applications
 - Refactorings
 - Patterns

Particular context

- The CO-OP project
 - Formal (weakest precondition) semantics
 - Soundness of the laws
 - Further applications: compilation of OO features

Agenda

- Background
 - Algebraic laws and reasoning
 - Refinement algebra
 - Specifications and program development
 - Data refinement
 - Semantic models
- The language
- Algebraic laws
- Completeness
- Application: refactoring
- Application: patterns
- Discussion

Agenda

- Background
 - Algebraic laws and reasoning
 - Refinement algebra
 - Specifications and program development
 - Data refinement
 - Semantic models

Algebraic laws and reasoning

- Language
 - skip
 - $x := e$
 - $c_1; c_2$

Assignment: $x := e$

- Multiple (parallel) assignment

$$x, y := y, x$$

- The variables in the list on the left must be distinct
- The two lists of variables must be equal-length
- Well-defined expressions
- No side-effects

Example of laws

Law 1 ⟨Void assignment⟩

$$(x := x) = \mathbf{skip}$$

Law 2 ⟨Identity assignment⟩

$$(x, y := e, y) = (x := e)$$

Law 3 ⟨Assignment symmetry⟩

$$(x, y := e, f) = (y, x := f, e)$$

Example of laws

Law 4 ⟨Combine assignments⟩

$$(x := e; x := f) = (x := f[e/x])$$

$f[e/x]$ denotes the substitution of e for x in f .

Law 5 ⟨Composition identity⟩

$$(\mathbf{skip}; c) = c = (c; \mathbf{skip})$$

Reasoning style

- Laws allow to identify syntactically different programs
- Reasoning framework: conditional (in)equational rewriting
- Easy automation

- **To prove $c_1 = c_2$ if $cexp$ typically involves**

$$\begin{aligned} & c_1 \\ &= [\text{Law } X] \\ & \dots \\ &= [\text{Law } Y] \\ & c_2 \end{aligned}$$

Reasoning style: example

Consider x , y , w and z are distinct identifiers. Then

$$(x := y; w := z) = (w := z; x := y)$$

Proof:

$$x := y; w := z$$

$$= [\text{Law } \mathbf{2}]$$

$$x, w := y, w; w, x := z, x$$

$$= [\text{Law } \mathbf{3}]$$

$$w, x := w, y; w, x := z, x$$

$$= [\text{Law } \mathbf{4}]$$

$$w, x := z, y$$

Reasoning style: example

$$w, x := z, y$$
$$= [\text{Law 4}]$$
$$w, x := z, x; w, x := w, y$$
$$= [\text{Law 3}]$$
$$w, x := z, x; x, w := y, w$$
$$= [\text{Law 2}]$$
$$w := z; x := y$$

Extending the language

The notation

`var x : T • c end`

declares x with type T and scope c

In a program like

`var y : T • $x := y$ end`

we say that y is *bound* (or *local*) and x is *free* (or *global*)

Laws of declaration

Law 6 ⟨Void declaration⟩

$(\mathbf{var} \ x : T \bullet c \ \mathbf{end}) = c$ provided x is not free in c

Law 7 ⟨Assignment elimination⟩

$(\mathbf{var} \ x : T \bullet c; \ x, y := e, f \ \mathbf{end}) = (\mathbf{var} \ x : T \bullet c; \ y := f \ \mathbf{end})$

Exercise

Exercise 1 Assuming that z is not free in $x := y$ and that these variables have type T , prove the following equivalence:

$$(\mathbf{var} \ z : T \bullet z := y; x := z \ \mathbf{end}) = (x := y)$$

Additional programming constructs

Notation: `abort`

Law 8 ⟨Composition zero⟩

$(\text{abort}; c) = \text{abort}$

Conditional

Notation:

if $\square i \bullet \psi_i \rightarrow c_i$ **fi**

Example:

if $(x \leq y) \rightarrow z := y$

$\square (y \leq x) \rightarrow z := x$

fi

Some laws of conditional

Law 9 ⟨Conditional elimination⟩

If $\bigvee i \bullet \psi_i = \mathbf{true}$, then

$$\mathbf{if} \ [\ i \bullet \psi_i \rightarrow c \ \mathbf{fi} = c$$

Law 10 ⟨Conditional commutative⟩

If π is any permutation of $1 \dots n$, then

$$\mathbf{if} \ [\ i \bullet \psi_i \rightarrow c_i \ \mathbf{fi} = \mathbf{if} \ [\ i \bullet \psi_{\pi(i)} \rightarrow c_{\pi(i)} \ \mathbf{fi}$$

Iteration

Notation:

do $\llbracket i \bullet \psi_i \rightarrow c_i$ **od**

Example:

$r := 1$; **do** $(n > 1) \rightarrow r := r * n; n := n - 1$ **od**

Nondeterministic choice

Notation: $c_1 \sqcap c_2$

Example:

$x := 1 \sqcap x := 2;$

if $(x = 1) \rightarrow$ **skip**

[] $(x = 2) \rightarrow$ **abort**

fi

Law 11 \langle Nondeterministic choice zero \rangle

$(\mathbf{abort} \sqcap c) = \mathbf{abort}$

Agenda

- Background
 - Algebraic laws and reasoning
 - **Refinement algebra**
 - Specifications and program development
 - Data refinement
 - Semantic models

Refinement algebra

- Equality is generally too strong for development
- A program may behave possibly better than its specification
- An ordering relation $c_1 \sqsubseteq c_2$ holds when:
 - c_2 satisfies every specification satisfied by c_1
 - c_2 possibly works for more inputs and states than c_1
 - c_2 possibly produces a more precise (deterministic) result than c_1

Refinement algebra

A definition in terms of nondeterministic choice

$$(c_1 \sqsubseteq c_2) \hat{=} (c_1 \sqcap c_2 = c_1)$$

Refinement algebra

Example of refinement laws

Law 12 ⟨abort is the worst program⟩

abort \sqsubseteq c

Law 13 ⟨Declaration initialisation⟩

(var $x : T \bullet c$ end) \sqsubseteq **(var $x : T \bullet x := e; c$ end)**

Refinement algebra

\sqsubseteq is a partial ordering

Law 14 ⟨Refinement reflexive⟩

$$c \sqsubseteq c$$

Law 15 ⟨Refinement transitive⟩

$$(c_1 \sqsubseteq c_2) \wedge (c_2 \sqsubseteq c_3) \Rightarrow (c_1 \sqsubseteq c_3)$$

Law 16 ⟨Refinement antisymmetric⟩

$$(c_1 \sqsubseteq c_2) \wedge (c_2 \sqsubseteq c_1) \Rightarrow (c_1 = c_2)$$

Refinement algebra

Programming operators are monotonic wrt \sqsubseteq

For example, $c_1 \sqsubseteq c_2$ implies

$$c_1; c_3 \sqsubseteq c_2; c_3$$

In general:

Law 17 ⟨Refinement compositional⟩

$$(c_1 \sqsubseteq c_2) \Rightarrow (F(c_1) \sqsubseteq F(c_2))$$

Agenda

- Background
 - Algebraic laws and reasoning
 - Refinement algebra
 - Specifications and program development
 - Data refinement
 - Semantic models

Specifications and program development

- Modern approaches: programs embedded into specification space
- Uniformity: single notation
- Program development reduces to specification transformation
- Examples: calculi by Back, Morgan and Morris

Morgan's calculus

- Distinguishing feature is the *specification statement*

$$w : [pre, post]$$

- Example:

$$s, r : [e \notin s, s = s_0 \cup \{e\} \wedge r = \text{"Okay"}]$$

Some extreme specifications

- The worst possible specification

$$\text{abort} \hat{=} x : [\text{false}, \text{true}]$$

- The best possible specification

$$\text{miracle} \hat{=} x : [\text{true}, \text{false}]$$

- skip as a specification

$$\text{skip} \hat{=} : [\text{true}, \text{true}]$$

- Assumption

$$\{b\} \hat{=} : [b, \text{true}]$$

Laws of specification statements

- Notion of refinement captured by the laws:

Law 18 ⟨Precondition weakening⟩

$w : [pre, post] \sqsubseteq w : [pre', post]$ **provided** $pre \Rightarrow pre'$

Law 19 ⟨Postcondition strengthening⟩

$w : [pre, post] \sqsubseteq w : [pre, post']$ **provided**

$pre[w_0/w] \wedge post' \Rightarrow post$

Exercise

Exercise 2 Prove the refinement:

$$s, r : [e \notin s, s = s_0 \cup \{e\} \wedge r = \text{“Okay”}]$$

$$\sqsubseteq$$

$$s, r : [\mathbf{true}, (e \notin s_0 \wedge s = s_0 \cup \{e\} \wedge r = \text{“Okay”}) \vee \\ (e \in s_0 \wedge s = s_0 \wedge r = \text{“AlreadyMember”})]$$

Laws for refining into code

Law 20 ⟨Assignment introduction⟩

$w, v : [pre, post] \sqsubseteq v := e$ **provided**

$(v = v_0) \wedge pre \Rightarrow post[e/v]$

Law 21 ⟨Following assignment⟩

$w, v : [pre, post] \sqsubseteq w, v : [pre, post[e/v]]; v := e$

Exercise

Exercise 3 Prove the following refinement:

$$\begin{aligned} & s, r : [e \notin s, s = s_0 \cup \{e\} \wedge r = \text{“Okay”}] \\ & \sqsubseteq (s := s \cup \{e\}; r := \text{“Okay”}) \end{aligned}$$

Agenda

- Background
 - Algebraic laws and reasoning
 - Refinement algebra
 - Specifications and program development
 - **Data refinement**
 - Semantic models

Data refinement

- Complementary to algorithmic refinement
- Supports coherent change of data representation
- Example: Consider a set inclusion operation

$$s : [e \notin s, s = s_0 \cup \{e\}]$$

and a possible data refinement

$$t : [e \notin \text{set } t, t = t_0 \hat{\wedge} \langle e \rangle]$$

- Intuitive refinement, but how to prove it?

Data refinement

- Missing connection is a relation between *abstract* and *concrete states*:

$$s = \text{set } t$$

- Often, this relation is functional: *abstraction function*
- In our example, the abstraction function is set

Data refinement in Morgan's calculus

- Formulated at the level of programming modules
- A module includes state variables, initialisation and procedures
- The technique involves
 - adding the concrete variables to the module
 - making the abstract variables auxiliary
 - removing the auxiliary (abstract) variables
- When the relation is functional, these steps are combined

Data refinement in Morgan's calculus

- To proceed with the data refinement, transformations are proposed to deal with
 - initialisation
 - assignments
 - specification statements
 - ...

Transformation for specification statements

A specification statement

$$w, x : [pre, post]$$

becomes

$$w, z : [pre[af\ z/x], post[af\ z_0, af\ z/x_0, x]]$$

where

- af stands for the abstraction function
- x for the abstract variables
- z for the concrete variables

Exercise

Exercise 4 Formalise the data transformation of the statement previously presented: $s : [e \notin s, s = s_0 \cup \{e\}]$ into $t : [e \notin \text{set } t, t = t_0 \hat{\cap} \langle e \rangle]$ considering the abstraction function: $s = \text{set } t$.

Agenda

- Background
 - Algebraic laws and reasoning
 - Refinement algebra
 - Specifications and program development
 - Data refinement
 - Semantic models

Semantic models

- Programming laws define (an algebraic) semantics
- But postulating laws can give rise to inconsistencies
- Solution: link algebraic semantics with a mathematical model

Weakest preconditions

- Model is a predicate transformer, usually called wp
- Programs as functions from predicates to predicates
- $wp.p.\psi$ is the weakest precondition that guarantees that program p establishes postcondition ψ
- Total correctness

Weakest precondition semantics of our simple language

$wp.\mathbf{skip}.\psi$	ψ
$wp.\mathbf{abort}.\psi$	false
$wp.x := e.\psi$	$\psi[e/x]$
$wp.(c_1; c_2).\psi$	$wp.c_1.(wp.c_2.\psi)$
$wp.(\mathbf{if} \ \square i \bullet \psi_i \ \rightarrow \ c_i \ \mathbf{fi}).\psi$	$(\bigvee i \bullet \psi_{-i}) \wedge (\bigwedge i \bullet \psi_i \Rightarrow wp.c_i.\psi)$
$wp.(c_1 \sqcap c_2).\psi$	$(wp.c_1.\psi) \wedge (wp.c_2.\psi)$
$wp.(\mathbf{var} \ x : T \bullet c).\psi$	$\forall x : T \bullet wp.c.\psi$

Weakest precondition definition of refinement

$$(c_1 \sqsubseteq c_2) \hat{=} wp.c_1.\psi \Rightarrow wp.c_2.\psi$$