Refinement Algebra for an O-O Language with References

Augusto Sampaio

(joint work with G. Lucero and D. Naumann)

Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil

InfoBlender Seminar
HASLab/INESC Tec & Universidade do Minho

April 2015

General context and motivation

Reasoning with pointers and references is difficult

- Aliasing and sharing

New techniques for spatial separation of pointers

- Separation logic, [implicit] dynamic frames, . . .

Few algebraic approaches

- No comprehensive set of algebraic laws for references

General context and motivation

Program semantics

- Operational, denotational, algebraic

The algebraic approach

- Properties as (in)equations (laws) relating operators
- No explicit mathematical model
- Modularity and easy mechanisation
- Soundness and completeness?

Refinement algebra versus refinement calculus

Calculi by Back, Morgan, Morris, . . .

- Focus on program derivation

Refinement algebra

- Semantic framework
- Applications of program transformation
  - compilation, hw/sw codesign, optimisation, refactorings, patterns, test generation, . . .

Algebra of imperative programming

Language example (cf. Laws of Programming [Hoare *et al*])

$Skip$	do nothing
abort	unpredictable behaviour
$x, y := e, f$	assignment (possibly multiple)
$c_1 ;\ c_2$	sequential composition
$c_1 \sqcap c_2$	nondeterminism
$c_1 \ \triangleleft b \triangleright \ c_2$	conditional
$\mu\ X \bullet c$	recursive program X with body c
X	recursive call
var $v \bullet c$	declaration of v for use in program c

Examples of laws

Law 1 (Sequence associative)
$(c_1;\ c_2);\ c_3 = c_1;\ (c_2;\ c_3)$

Law 2 (Combine assignments)
$(x := e;\ x := f) = (x := f[e/x])$

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

Refinement

- Equality is generally too strong
- Refinement allows more applicable and deterministic implementations
  $$(c_1 \sqsubseteq c_2) \;\widehat{=}\; (c_1 \sqcap c_2 = c_1)$$
- $\sqsubseteq$ is a partial ordering
- Program operators are monotonic wrt $\sqsubseteq$

A few more laws: recursion

Law 3 (Fixed point) $F(\mu\ X \bullet F(X)) \ = \ \mu\ X \bullet F(X)$

Law 4 (Least fixed point) $F(Y) \sqsubseteq Y \ \Rightarrow \ \mu\ X \bullet F(X) \sqsubseteq Y$

where F stands for an arbitrary context

Algebra of O-O programming (copy semantics)

[Borba, Sampaio]

Law 5 (Move original method to superclass)

class B **extends** A $\quad ads$ $\quad$ **meth** $m \mathrel{\widehat=} (sig \bullet b)$ $\quad mts$ **end** **class** C **extends** B $\quad ads'$ $\quad$ **meth** $m \mathrel{\widehat=} (sig \bullet b')$ $\quad mts'$ **end**	$=_{cds,c}$	**class** B **extends** A $\quad ads$ $\quad$ **meth** $m \mathrel{\widehat=} (\ sig \bullet$ $\qquad b \triangleleft \mathbf{not}(\mathbf{self\ is}\ C) \triangleright b'$ $\qquad)$ **end** **class** C **extends** B $\quad ads'$ $\quad mts'$ **end**

provided ...

Algebra of imperative programming with references

cd	$::=$	**class** A	class declaration (record)
		$\overline{f : T}$	field declarations
		end	
c	$::=$	$\ldots$	
		$\mid \overline{le} := \overline{e}$	multiple assignment
		$\mid x \leftarrow$ **new** A	new instance
le	$::=$	$x \mid e.f$	variable, field

$\ldots$

Aliasing

Let x be a variable, d and e expressions and p and q left expressions

$$
\begin{aligned}
alias[x, x] &\stackrel{def}{=} \mathbf{true} \\
alias[d.f, e.f] &\stackrel{def}{=} d == e \\
alias[p, q] &\stackrel{def}{=} \mathbf{false} \quad \text{otherwise}
\end{aligned}
$$

Field Substitution [Morris, Bornat]

Field Substitution

$$e_d^{e1.f} \stackrel{def}{=} e\,[[e1.f := d]\ /\ f]$$

Conditional Field

$$e.[e1.f := d] \stackrel{def}{=} d \triangleleft e == e_1 \triangleright e.f$$

Field Substitution on Left Expressions

$$(e.f)_d^{\hat{}e_1.g} \stackrel{def}{=} (e_d^{e1.g}).f \qquad \text{including when } f \equiv g$$

$\ldots$

Example of field substitution

Effect of $x.f := e$ on $x.f + y.g + z.f$

$$
\begin{aligned}
&(x.f + y.g + z.f)^{x.f}_{e} = && \text{(substitution def)}\\
&\qquad x.[x.f := e] + y.g + z.[x.f := e] = && \text{(cond field def)}\\
&\qquad (e \triangleleft x == x \triangleright x.f) + y.g + (e \triangleleft x == z \triangleright z.f) = && (x == x \text{ is true})\\
&\qquad e + y.g + (e \triangleleft x == z \triangleright z.f)
\end{aligned}
$$

Assertions

$$[e] \stackrel{def}{=} \textbf{skip} \triangleleft e \triangleright \perp$$

Assertions are used to record and spread alias information

$$\begin{aligned} b \triangleleft e \triangleright c &= ([e];\ b) \triangleleft e \triangleright c \\ b \triangleleft e \triangleright c &= b \triangleleft e \triangleright ([\textbf{not}\ e];\ c) \\ [e];\ (b \triangleleft d \triangleright c) &= ([e];\ b) \triangleleft d \triangleright ([e];\ c) \\ &\ldots \end{aligned}$$

We use $[\mathfrak{D} e]$ to assert the definedness of expression e

Laws of Assignment

$$[alias[p, q_e^{\hat{}p}]];\ p := e;\ q := d \ = \ [alias[p, q_e^{\hat{}p}]];\ [\mathfrak{D}\, e];\ p := d_e^p$$

Laws of Assignment

$$[alias[p, q_e\hat{}^p]];\ p := e;\ q := d \ = \ [alias[p, q_e\hat{}^p]];\ [\mathfrak{D}e];\ p := d_e^p$$

$$\overline{p}, q, q := \overline{e}, d_1, d_2 \ = \ \overline{p}, q := \overline{e}, d_1 \ \sqcap \ \overline{p}, q := \overline{e}, d_2$$

Laws of Assignment

$$[alias[p, q_e^{\hat{}p}]];\ p := e;\ q := d \ = \ [alias[p, q_e^{\hat{}p}]];\ [\mathfrak{D} e];\ p := d_e^p$$

$$\overline{p}, q, q := \overline{e}, d_1, d_2 \ = \ \overline{p}, q := \overline{e}, d_1 \ \sqcap \ \overline{p}, q := \overline{e}, d_2$$

$$[\textbf{not}\ alias[p, q]];\ p, q := e, q \ = \ [\mathfrak{D} q \wedge \textbf{not}\ alias[p, q]];\ p := e$$

Laws of Assignment

$$[alias[p, \hat{q_e^p}]];\ p := e;\ q := d \ = \ [alias[p, \hat{q_e^p}]];\ [\mathfrak{D} e];\ p := d_e^p$$

$$\overline{p}, q, q := \overline{e}, d_1, d_2 \ = \ \overline{p}, q := \overline{e}, d_1 \ \sqcap \ \overline{p}, q := \overline{e}, d_2$$

$$[\textbf{not}\ alias[p, q]];\ p, q := e, q \ = \ [\mathfrak{D} q \wedge \textbf{not}\ alias[p, q]];\ p := e$$

$$p := e;\ (b \triangleleft d \triangleright c) \ = \ (p := e;\ b) \triangleleft d_e^p \triangleright (p := e;\ c)$$

$$\ldots$$

Laws of **new**

$$x \leftarrow \textbf{new}\, A \;=\; x \leftarrow \textbf{new}\, A;\; \overline{x.f} := \overline{default(T)}$$

Laws of **new**

$$x \leftarrow \textbf{new}\, A \;=\; x \leftarrow \textbf{new}\, A;\; \overline{x.f} := \overline{default(T)}$$

$$x \leftarrow \textbf{new}\, A \;= x \leftarrow \textbf{new}\, A;\; [x \neq \textbf{null}]$$

$$\ldots$$

Relative Completeness

Theorem Let c be a command in which h does not occur free and assume $refs = freeRefs(c)$. We have

$$c \;=\; \textbf{var}\ h : Heap \bullet load(h, refs);\ S(c, h);\ store(h)$$

where $S(c, h)$ is the simulation of c using the explicit heap h

Algebra of O-O programming (reference semantics)

```
class A extends B
  priv f:int
  meth m = ( x:T @ c1 )
end
class B extends C
  priv g:D
  meth m = ( x:T @  c2 )
end
...
@ ...; x.m(e); ...
```

Laws of classes and O-O commands

```
class A extends B
  pub f:int
end
class B extends C
  pub g:D
end
...
@ ...; ( var x:T @ x:=e;  (μ m @  ...)); ...
```

Laws of commands with references

```
class A extends B
  pub f:int
end
class B extends C
  pub g:D
end
...
@ var h:Heap @  load(h,...);  ...;  store(h)
```

Example of derived law

Law 6 (Replace field by temporary)

Consider that the class T declares a field $f : T_f$, then

$$\begin{array}{l} \textbf{var}\ x : T \bullet x \leftarrow \textbf{new}\ T;\ c\ = \\ \qquad \textbf{var}\ x : T, t : T_f \bullet x \leftarrow \textbf{new}\ T;\ t := x.f;\ c[t/x.f];\ x.f := t \end{array}$$

provided

(1) *t is a fresh variable in c;*

(2) *x is read only, not used as argument nor assigned to variables or fields in c*

(3) *if $e.f$ occurs in c then $e \equiv x$.*

Rule: Replace Method with Method Object

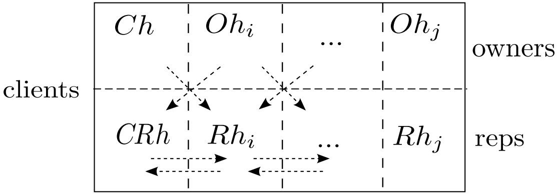
$$
\begin{array}{l}
\mathtt{class}\ A\ \mathtt{extends}\ C \\
\quad ads \\
\quad \mathtt{meth}\ m\ \widehat{=}\ (\ \overline{x : T} \bullet \\
\qquad \mathtt{var}\ \overline{t : R} \bullet \\
\qquad\quad c[\mathbf{self}, \overline{x}, \overline{t}]\ ) \\
\quad mts \\
\mathtt{end}
\end{array}
$$

$$=_{cds,b}$$

$$
\begin{array}{l}
\mathtt{class}\ A\ \mathtt{extends}\ C \\
\quad ads \\
\quad \mathtt{meth}\ m\ \widehat{=}\ (\ \overline{x : T} \bullet \\
\qquad \mathtt{var}\ s : M \\
\qquad\quad s \leftarrow \mathbf{new}\ M(\mathbf{self}, \overline{x}); \\
\qquad\quad s.m()\ ) \\
\quad mts \\
\mathtt{end} \\
\mathtt{class}\ M\ \mathtt{extends}\ \mathtt{Object} \\
\quad \mathtt{pri}\ \mathrm{a : A} \\
\quad \mathtt{pri}\ \overline{x : T} \\
\quad \mathtt{pri}\ \overline{t : R} \\
\quad \mathtt{meth}\ \mathtt{ctr}\ \widehat{=}\ (\ a : A, \overline{x : T} \bullet \\
\qquad \mathbf{self}.a, \mathbf{self}.\overline{x} := a, \overline{x}\ ) \\
\quad \mathtt{meth}\ m\ \widehat{=}\ (\bullet \\
\qquad c[\mathbf{self}.a, \mathbf{self}.\overline{x}, \mathbf{self}.\overline{t}]) \\
\mathtt{end}
\end{array}
$$

Ownership and Confinment

Data refinement based on confinement notions



Definition. A *local coupling* is a relation between two different representations of *Own* (one using *Rep* and other using *Rep′*)

Definition. A *simulation* is a local coupling that is preserved at the creation of *Own* instances and also at the end of every method call to *Own* instances.

Change of data Representation

Based on a notion of ownership confinement

Law 7 (Data Refinement of class hierarchies)

$$cs =_{cds,c} cs'$$

provided

- *cs and cs' are hierarchies with root Own, and cds has no subclasses of Own;*
- *cds, cs is confined for Own, Rep;*
- *cds, cs' is confined for Own, Rep';*
- $\ldots$
- *There exists a simulation R.*

Rule 1 (Pull up field)

$$
\begin{array}{|l|}
\hline
\textbf{class}\ M\ \textbf{extends}\ N \\
\quad ads_m \\
\quad mts_m \\
\textbf{end} \\
\textbf{class}\ L\ \textbf{extends}\ M \\
\quad \textbf{prot}\ x : T;\ \ ads_l \\
\quad mts_l \\
\textbf{end} \\
\textbf{class}\ K\ \textbf{extends}\ M \\
\quad \textbf{prot}\ y : T;\ \ ads_k \\
\quad mts_k \\
\textbf{end} \\
cds_1 \\
\hline
\end{array}
\;=_{cds,c}\;
\begin{array}{|l|}
\hline
\textbf{class}\ M\ \textbf{extends}\ N \\
\quad \textbf{prot}\ z : T;\ \ ads_m \\
\quad mts_m \\
\textbf{end} \\
\textbf{class}\ L\ \textbf{extends}\ M \\
\quad ads_l \\
\quad mts'_l \\
\textbf{end} \\
\textbf{class}\ K\ \textbf{extends}\ M \\
\quad ads_k \\
\quad mts'_k \\
\textbf{end} \\
cds'_1 \\
\hline
\end{array}
$$

where $mts'_l = mts_l[z/x]$, $mts'_k = mts_k[z/y]$
similar for cds'_1 ...

Some proof steps

(1) Apply law to move attributes x and y to class M

(2) Apply data refinement law with $M = Own$, no $Reps$ and local coupling:

$$
\begin{array}{ll}
 & type(\mathbf{self}) = type(\mathbf{self}') \\
\wedge & (\mathbf{self\ is}\ L \Rightarrow \mathbf{self}'.z = \mathbf{self}.x) \\
\wedge & (\mathbf{self\ is}\ K \Rightarrow \mathbf{self}'.z = \mathbf{self}.y) \\
\wedge & \forall f \in fields(type(\mathbf{self})) \\
 & \quad f \neq x \wedge f \neq y \Rightarrow \mathbf{self}'.f = \mathbf{self}.f
\end{array}
$$

(3) Prove that this is a simulation

Summary: overall reasoning framework

Patterns		
Refactoring rules		
Command laws	Class laws	Data refinement
Semantics		

Ongoing and future work

- Permissions for framing
  - [implicit] dynamic frames, separation logic
- Ownership + data refinement as in Morgan's calculus
- Proofs of the laws on a relational model
  - Extension of that for Laws of Programming
- More refactorings and design patterns
  - Observer, Flyweight, creational patterns, . . .
- Other applications: compiler optimisations
- Mechanisation
  - A major challenge is dealing with refactoring provisos

Refinement Algebra for an O-O Language with References

Augusto Sampaio

(joint work with G. Lucero and D. Naumann)

Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil

InfoBlender Seminar
HASLab/INESC Tec & Universidade do Minho

April 2015

More on the semantics of **new**

$$
\begin{aligned}
&[y == \textbf{alloc}];\ x \leftarrow \textbf{new}\, A \ = \\
&\qquad [y == \textbf{alloc}];\ x \leftarrow \textbf{new}\, A;\ [x \notin y \wedge \textbf{alloc} == y \cup \{x\}]
\end{aligned}
$$

More on the semantics of **new**

$$
\begin{aligned}
&[y == \textbf{alloc}];\ x \leftarrow \textbf{new}\, A \ = \\
&\qquad [y == \textbf{alloc}];\ x \leftarrow \textbf{new}\, A;\ [x \notin y \wedge \textbf{alloc} == y \cup \{x\}]
\end{aligned}
$$

As a consequence:

$$x \leftarrow \textbf{new}\, A;\ x \leftarrow \textbf{new}\, A \ \neq\ x \leftarrow \textbf{new}\, A$$

Algebra of concurrent programming

CSP, occam, ...and respective laws

Some applications:

- Hardware compilers [He *et al*] [Perna *et al*]
- Hardware/software codesign [Silva, Sampaio] [He et al]
  $S \sqsubseteq (c_1 \mid\mid c_2 \mid\mid \ldots \mid\mid c_n) \sqsubseteq SW \mid\mid HW_1 \mid\mid \ldots \mid\mid HW_k$
- Test case generation using CSP and FDR [Nogueira, Sampaio]
    - **assert** $model \sqsubseteq model_{marks}$
    - Industrial partners: Motorola and Embraer