An exercise on confined separation logic

Luis S. Barbosa
(joint work with J. N. Oliveira and Shuling Wang)

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

Universidade do Minho

IFIP WG 1.3

Sinaia, 2-3 September 2014

Summary

Separation logic

- a logic for reasoning about shared mutable data structures
- key notion: separable conjunction $(p * q)$: p and q hold for disjoint portions of the addressed storage

The confinement extension

- Confined types: *"An object is said to be confined in a domain off all references to this object originate from objects of the domain"* [Bokowski & Vitek, 1999]
- Confined separation logic, proposed in [Wang & Qiu, 2007] as an extension to deal with problems involving dangling object references (introducing restricted forms of $*$)

Summary

Our exercise

- To discuss the semantics of such an extension by defining a relational model for the overall logic, parametric on the shapes of both the store and the heap,
- aiming at providing a simple interpretation of the new confinement connectives and helping in seeking for duals,
- as well as proving calculationally a number of properties of this logic.

Store & Heap

An interpretation state is a Store σ paired with a Heap H:

$$V \xrightarrow{\sigma} A + K$$

$$K \xrightarrow{H} A + K$$

Store & Heap

$$
\begin{tikzcd}
V \arrow[r, "\sigma"] \arrow[dr, "\in_{\mathsf{G}} \cdot \sigma"'] & \mathsf{G}(A, K) \arrow[d, "\in_{\mathsf{G}}"] \\
 & K \\
K \arrow[r, "H"'] \arrow[ur, "\in_{\mathsf{F}} \cdot H"] & \mathsf{F}(A, K) \arrow[u, "\in_{\mathsf{F}}"']
\end{tikzcd}
$$

NB: $k\,(\in_{\mathsf{G}} \cdot \sigma)\,x$ asserts that variable x currently holds reference k. Thus,

- $Reach \;=\; \in_{\mathsf{G}} \cdot \sigma$
- $Alias = \ker Reach \;=\; Reach^{\circ} \cdot Reach$

Separated union

It is a partial operator of type

$$Heap \xleftarrow{\;*\;} Heap \times Heap$$

which joins two heaps

$$H \;*\; (H_1, H_2) \;\stackrel{\mathrm{def}}{=}\; (H_1 \parallel H_2) \wedge (H = H_1 \cup H_2)$$

in case they are (domain) disjoint:

$$H_1 \parallel H_2 \;\stackrel{\mathrm{def}}{=}\; \neg\langle\exists\; b, a, k \;::\; b\; H_1\; k \wedge a\; H_2\; k\rangle$$

NB: $t\ H\ k$ means "thing t is the referent of reference k in heap H"

Separability (going pointfree)

$$
\begin{array}{cl}
 & \neg\langle \exists\ b, a, k\ ::\ b\ H_1\ k \wedge a\ H_2\ k \rangle \\
\equiv & \quad \{\ \exists\text{-nesting and relational converse}\ \} \\
 & \neg\langle \exists\ b, a\ ::\ \langle \exists\ k\ ::\ b\ H_1\ k \wedge k\ H_2^{\circ} a \rangle\rangle \\
\equiv & \quad \{\ \text{introduce relational composition}\ \} \\
 & \neg\langle \exists\ b, a\ ::\ b(H_1 \cdot H_2^{\circ})a \rangle \\
\equiv & \quad \{\ \text{de Morgan ; negation}\ \} \\
 & \langle \forall\ b, a\ ::\ b(H_1 \cdot H_2^{\circ})a \Rightarrow \textsc{False} \rangle \\
\equiv & \quad \{\ \text{empty relation: } b \perp a \text{ is always false}\ \} \\
 & \langle \forall\ b, a\ ::\ b(H_1 \cdot H_2^{\circ})a \Rightarrow b \perp a \rangle \\
\equiv & \quad \{\ \text{drop points } a, b\ \} \\
 & H_1 \cdot H_2^{\circ} \subseteq \perp
\end{array}
$$

Separability (going pointfree)

So we can redefine

$$H_1 \parallel H_2 \stackrel{\mathrm{def}}{=} H_1 \cdot H_2^\circ \subseteq \perp$$

cf diagram:

$$\begin{tikzcd} K \arrow[r, "H_1"] & F(A,K) \\ K \arrow[u, "id"] \arrow[ur, phantom, "\subseteq"] & F(A,K) \arrow[l, "H_2^\circ"] \arrow[u, "\perp"'] \end{tikzcd}$$

NB: $\parallel$ can be extended for any pair of (not necessarily simple) relations:

$$R \parallel S \stackrel{\mathrm{def}}{=} R \cdot S^\circ \subseteq \perp$$

Separability (going pointfree)

Properties of $\|$ are easily asserted by calculation, e.g.

$$
\begin{array}{ll}
& \boxed{(\mathrm{R} \cup S) \parallel T} \\
\equiv & \qquad \{ \text{ definition of } \parallel \ \} \\
& (R \cup S) \cdot T^{\circ} \subseteq \perp \\
\equiv & \qquad \{ \ \cdot T^{\circ} \text{ is a lower adjoint } \} \\
& (R \cdot T^{\circ}) \cup (S \cdot T^{\circ}) \subseteq \perp \\
\equiv & \qquad \{ \ \cup\text{-universal } \} \\
& R \cdot T^{\circ} \subseteq \perp \ \text{ and } S \cdot T^{\circ} \subseteq \perp \\
\equiv & \qquad \{ \text{ definition of } \parallel \ \} \\
& \boxed{\mathrm{R} \parallel T \text{ and } S \parallel T}
\end{array}
$$

Background: PF-transform

- Predicate calculus is expressive but difficult to manipulate
- Modelling requires descriptive notations (intuitive, domain-specific, often graphical)
- Reasoning requires compact notations (simple, generic, amenable to easy formal manipulation)

The problem is recurrent in Mathematics and typically solved by some sort of transform

The PF-transform to the calculus of binary relations (à la Tarski) leads to a domain which is simpler, algebraic and easier to calculate with.

Background: PF-transform

Thus, we'll resort to a systematic transformation

- of predicate calculus expressions ...
- into pointfree, relational notation

for example,

- dropping quantifiers as much as possible, as in eg.

$$R \subseteq S \quad \equiv \quad \langle \forall\ y, x\ ::\ y\ R\ x \Rightarrow y\ S\ x \rangle$$

- or, thanks to relational composition,

$$b(R \cdot S)c \quad \equiv \quad \langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle$$

Background: PF-transform

ϕ	$PF\ \phi$
$\langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle$	$b(R \cdot S)c$
$\langle \forall\ a, b\ ::\ b\ R\ a \Rightarrow b\ S\ a \rangle$	$R \subseteq S$
$\langle \forall\ a\ ::\ a\ R\ a \rangle$	$id \subseteq R$
$\langle \forall\ x\ ::\ x\ R\ b \Rightarrow x\ S\ a \rangle$	$b(R \setminus S)a$
$\langle \forall\ c\ ::\ b\ R\ c \Rightarrow a\ S\ c \rangle$	$a(S\ /\ R)b$
$b\ R\ a \wedge c\ S\ a$	$(b, c)\langle R, S \rangle a$
$b\ R\ a \wedge d\ S\ c$	$(b, d)(R \times S)(a, c)$
$b\ R\ a \wedge b\ S\ a$	$b\ (R \cap S)\ a$
$b\ R\ a \vee b\ S\ a$	$b\ (R \cup S)\ a$
$(f\ \ b)\ R\ (g\ \ a)$	$b(f^{\circ} \cdot R \cdot g)a$
TRUE	$b\ \top\ a$
FALSE	$b\ \bot\ a$

where R, S, id are binary relations.

Standard separation logic

Syntax:

$$
\begin{array}{rcll}
p & ::= & \ldots & \\
 & | & \mathbf{emp} & /\!*\ \text{heap is empty}\ */ \\
 & | & e \mapsto e & /\!*\ \text{singleton heap}\ */ \\
 & | & p * p & /\!*\ \text{separating conjunction}\ */ \\
 & | & p \mathrel{-\!\!*} p & /\!*\ \text{separating implication}\ */
\end{array}
$$

Semantics:

$$
\begin{array}{rcl}
[\![e]\!] & : & \mathit{Store} \to A + K \\
[\![p]\!] & : & (\mathit{Heap} \times \mathit{Store}) \to \mathbb{B}
\end{array}
$$

Semantics of separating connectives

Separating conjunction:

$$[\![p * q]\!](H, S) \stackrel{\mathrm{def}}{=} \langle \exists\ H_0, H_1\ ::\ H * (H_0, H_1) \wedge [\![p]\!](H_0, S) \wedge [\![q]\!](H_1, S) \rangle$$

Separating implication:

$$[\![p \mathrel{-\!\!*} q]\!](H, S) \stackrel{\mathrm{def}}{=} \langle \forall\ H_0\ :\ H_0 \parallel H :\ [\![p]\!](H_0, S) \Rightarrow [\![q]\!](H_0 \cup H, S) \rangle$$

A PF-relational semantics

We define

- assertion semantics as a **relation** between stores and heaps,

$$Heap \xleftarrow{[\![p]\!]} Store$$

  a natural decision since every binary predicate is nothing but a relation

- the **preorder** on assertions induced by these semantics

$$p \rightarrow q \stackrel{\mathrm{def}}{=} [\![p]\!] \subseteq [\![q]\!]$$

  so that it can be distinguished from standard logic implication $\Rightarrow$.

Separating conjunction

Reynolds original definition of separating conjunction rewrites to

$$H[\![p * q]\!]S \stackrel{\text{def}}{=} \langle \exists\ H_0, H_1\ ::\ H * (H_0, H_1) \wedge H_0[\![p]\!]S \wedge H_1[\![q]\!]S \rangle$$

which PF-transforms to

$$[\![p * q]\!] \stackrel{\text{def}}{=} (*) \cdot \langle [\![p]\!], [\![q]\!] \rangle$$

just by recalling two rules of the PF-transform: composition

$$b(R \cdot S)c \quad \equiv \quad \langle \exists\ a\ ::\ bRa \wedge aSc \rangle$$

and splitting

$$(a, b)\langle R, S \rangle c \quad \equiv \quad a\ R\ c \wedge b\ S\ c$$

Separating implication

Taking seriously the rules

> *[There are] two further rules capturing the adjunctive relationship between separating conjunction and separating implication:*
>
> $$\frac{p_1 * p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 \mathrel{-\!\!*} p_3)} \qquad \frac{p_1 \Rightarrow (p_2 \mathrel{-\!\!*} p_3)}{p_1 * p_2 \Rightarrow p_3}$$

quoted from [Reynolds, 2002],

Separating implication

entails the need to make explicit the Galois connection

$$(p * x) \to y \quad \equiv \quad x \to (p \mathrel{-\!\!*} y)$$

which we regard as an equation where we know everything apart from $\mathrel{-\!\!*}$ (the unknown), which we want to calculate:

$$
\begin{array}{ll}
& (p * x) \to y \\
\equiv & \qquad \{ \text{ semantic preorder } \} \\
& [\![p * x]\!] \subseteq [\![y]\!] \\
\equiv & \qquad \{ \text{ PF-definition } \} \\
& (*) \cdot \langle [\![p]\!], [\![x]\!] \rangle \subseteq [\![y]\!] \\
\equiv & \qquad \{ \ldots \}
\end{array}
$$

Calculation of $-\!\!*$

To proceed we resort to two Galois connections, e.g.

$$R \cdot X \subseteq S \quad \equiv \quad X \subseteq R \setminus S$$

where

$$b\ (R \setminus S)\ a \quad \equiv \quad \langle \forall\ c\ :\ c\ R\ b :\ c\ S\ a \rangle$$

and

$$\langle R, S \rangle \subseteq X \quad \equiv \quad S \subseteq R \triangleright X$$

where

$$b(R \triangleright S)a \quad \equiv \quad \langle \forall\ c\ :\ c\ R\ a :\ (c, b)\ S\ a \rangle$$

Calculation of $-\!\!*$

Then,

$$
\begin{array}{ll}
 & (*) \cdot \langle [\![p]\!], [\![x]\!] \rangle \;\subseteq\; [\![y]\!] \\
\equiv & \qquad \{ \text{ the two GCs above in a row } \} \\
 & [\![x]\!] \;\subseteq\; [\![p]\!] \;\triangleright ((*) \setminus [\![y]\!]) \\
\equiv & \qquad \{ \text{ introduce } p \mathrel{-\!\!*} y \text{ such that } [\![p \mathrel{-\!\!*} y]\!] = [\![p]\!] \;\triangleright ((*) \setminus [\![y]\!]) \} \\
 & [\![x]\!] \;\subseteq\; [\![p \mathrel{-\!\!*} y]\!] \\
\equiv & \qquad \{ \text{ semantic preorder } \} \\
 & x \to (p \mathrel{-\!\!*} y)
\end{array}
$$

We are left with the meaning of $p \;\triangleright ((*) \setminus [\![y]\!])$ ...

Calculation of $-\!\!*$

$$
\begin{array}{ll}
& H[\![p -\!\!* y]\!]S \\
\equiv & \quad \{ \text{ above } \} \\
& H([\![p]\!] \rhd ((*) \setminus [\![y]\!]))S \\
\equiv & \quad \{ \rhd \text{ pointwise } \} \\
& \langle \forall\ H_0 \ :\ H_0[\![p]\!]S :\ (H_0, H)((*) \setminus [\![y]\!])S \rangle \\
\equiv & \quad \{ \text{ left division pointwise } \} \\
& \langle \forall\ H_0 \ :\ H_0[\![p]\!]S :\ \langle \forall\ H_1 \ :\ H_1 * (H_0, H) :\ H_1[\![y]\!])S \rangle\rangle \\
\equiv & \quad \{ \text{ quantifier nesting } \}
\end{array}
$$

Calculation of $-\!\!*$

$$
\begin{array}{ll}
 & \langle \forall\ H_0, H_1 \ :\ H_0[\![p]\!]S \wedge H_1 * (H_0, H) :\ H_1[\![y]\!])S \rangle \\
\equiv & \qquad \{ \ \text{separated union} \ \} \\
 & \langle \forall\ H_0, H_1 \ :\ H_0[\![p]\!]S \wedge H_0 \parallel H \wedge H_1 = H_0 \cup H :\ H_1[\![y]\!])S \rangle \\
\equiv & \qquad \{ \ \text{quantifier one-point} \ \} \\
 & \langle \forall\ H_0 \ :\ H_0[\![p]\!]S \wedge H_0 \parallel H :\ (H_0 \cup H)[\![y]\!])S \rangle \\
\equiv & \qquad \{ \ \text{quantifier trading} \ \} \\
 & \langle \forall\ H_0 \ :\ H_0 \parallel H :\ H_0[\![p]\!]S \Rightarrow (H_0 \cup H)[\![y]\!])S \rangle
\end{array}
$$

As expected, the definition postulated in [Reynolds, 2002].

Benefits of $((*), -\!\!*)$ connection

Immediate consequences:

$$
\begin{array}{rcl}
p * (x_1 \vee x_2) & \leftrightarrow & (p * x_1) \vee (p * x_2) \\
(x_1 \vee x_2) * p & \leftrightarrow & (x_1 * p) \vee (x_2 * p) \\
p -\!\!* (x_1 \wedge x_2) & \leftrightarrow & (p -\!\!* x_1) \wedge (p -\!\!* x_2)
\end{array}
$$

plus monotonicity, cancellations,

$$
x \rightarrow (p -\!\!* (p * x))
$$
$$
p * (p -\!\!* y) \rightarrow y
$$

and some others easily derivable, eg

$$
\begin{array}{rcl}
\mathbf{emp} & \rightarrow & p -\!\!* p \\
p * x & \leftrightarrow & p * (p -\!\!* (p * x)) \\
p -\!\!* x & \leftrightarrow & p -\!\!* (p * (p -\!\!* x))
\end{array}
$$

from [Bokowski & Vitek, 1999]

A problem

Aliasing — In object-oriented programming it is difficult to control the spread and sharing of object references. This pervasive aliasing makes it nearly impossible to know accurately who owns a given object, that is to say, which other objects have references to it.

A proposal

Confinement — An object is said to be confined in a domain if and only if all references to this object originate from objects of the domain.

A question

- how do we incorporate confinement into separation logic?

[Wang & Qiu, 2007]

Propose that the notion of heap disjointness be sophisticated in three directions:

- notIn — heaps disjoint and such that no references of the first point to the other
- In — heaps disjoint and such that all references in the first do point into the other
- inBoth — heaps disjoint and such that all references in the first are confined to both.

Confined disjointness — **notIn**

No outgoing reference in heap H_1 goes into separate H_2:

$$H_1 \neg\!\rhd H_2 \quad \stackrel{\mathrm{def}}{=} \quad H_1 \parallel H_2 \wedge H_2 \cdot \in_{\mathsf{F}} \cdot H_1 \subseteq \bot$$

In a diagram: path

$$\begin{tikzcd}
K \arrow[r, harpoon, "H_1"] & F(A,K) \arrow[dl, "\in_F"] \\
K \arrow[r, harpoon, "H_2"'] & F(A,K)
\end{tikzcd}$$

is empty, that is (back to points)

$$\neg\langle \exists\ k, k' \ :\ k \in \delta\, H_1 \ \wedge\ k' \in \delta\, H_2 :\ k' \in_F (H_1\ k)\rangle$$

Confined disjointness — **In**

All outgoing references in H_1 dangle because they all go into separate H_2:

$$H_1 \rhd H_2 \overset{\text{def}}{=} H_1 \parallel H_2 \;\wedge\; \in_{\mathsf{F}} \cdot H_1 \subseteq H_2^\circ \cdot \top$$

In a diagram: dependency graph $\in_F \cdot H_1$

$$\begin{tikzcd}
F(A,K) \arrow[d, "\in_F"'] & K \arrow[l, "H_1"'] \arrow[d, "\top"] \\
K & F(A,K) \arrow[l, "H_2^\circ"]
\end{tikzcd}$$

can only lead to references in the domain of H_2 ($\top$ transforms the everywhere true predicate)

Confined disjointness — **inBoth**

H_1 and H_2 are disjoint and all outgoing references in H_1 are confined to either H_2 or itself:

$$H_1 \lhd\!\rhd H_2 \quad \stackrel{\text{def}}{=} \quad H_1 \parallel H_2 \;\wedge\; \underbrace{\in_{\mathsf{F}} \cdot H_1 \subseteq (H_1 \cup H_2)^{\circ} \cdot \top}_{\alpha}$$

Comments:

- Note how clumsy α becomes once mapped back to point-level:

$$\langle \forall\ k\ :\ \langle \exists\ k'\ :\ k' \in \delta\, H_1 :\ k \in_{\mathsf{F}} (H_1\ k') \rangle :\ k \in \delta\, H_1 \vee k \in \delta\, H_2 \rangle$$

- Clearly, **In** $\Rightarrow$ **inBoth**

Confined separation logic

Three new variants of separating conjunction:

$$\begin{array}{ccc} & (*) & \\ \text{into-both conjunction} \quad \lhd\!\rhd & & \neg\rhd \quad \text{not-into conjunction} \\ \text{into conjunction} \quad \rhd & & \end{array}$$

able to express confinement subtleties.

Confined separation logic

- Left-not-into-right conjunction:

$$[\![p \neg\triangleright q]\!] \stackrel{\text{def}}{=} (*) \cdot \Phi_{\neg\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle$$

- Left-into-right conjunction:

$$[\![p \triangleright q]\!] \stackrel{\text{def}}{=} (*) \cdot \Phi_{\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle$$

- Left-into-both conjunction:

$$[\![p \triangleleft\triangleright q]\!] \stackrel{\text{def}}{=} (*) \cdot \Phi_{\triangleleft\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle$$

What about confined implication(s)?

Very easy:

- Just stick the relevant coreflexive (eg. $\Phi_{\triangleright}$) to separated union $(*)$ and and explore the Galois connection as before.
- Once points are back into formulæ, you get separated implication for each case, for instance:

$$H[\![p \multimap\!\triangleright\, y]\!]S \stackrel{\text{def}}{=} \langle \forall\ H_0\ :\ H_0 \triangleright H :\ H_0[\![p]\!]S \Rightarrow (H_0 \cup H)[\![y]\!]S \rangle$$

together with all the properties intact.

Confinement extension properties

- Semantics of confinement can be checked against eg. what happens to standard property

$$\mathbf{emp} * p \;\leftrightarrow\; p \;\leftrightarrow\; p * \mathbf{emp}$$

arising from two facts

$$\begin{aligned} H[\![\mathbf{emp}]\!]S &\equiv H = \bot \\ H * (H', \bot) &\equiv H = H' \end{aligned}$$

Confinement extension properties

- In the confined variant $\triangleright$ calculations easily lead to

$$\mathbf{emp} \triangleright p \quad \leftrightarrow \quad p$$

and to

$$p \triangleright \mathbf{emp} \;\; \leftrightarrow \;\; p \quad \Leftarrow \quad p \rightarrow \mathbf{emp}$$

recalling

$$H_1 \triangleright H_2 \quad \stackrel{\mathrm{def}}{=} \quad H_1 \parallel H_2 \;\wedge\; \in_{\mathsf{F}} \cdot H_1 \subseteq H_2^{\circ} \cdot \top$$

- The two other variants trivially preserve the standard rule.
- Confined variants of separating conjunction behave in particular ways even wrt some standard properties. For example, that $\triangleright$ is only semi-associative,

$$(p1 \triangleright p2) \triangleright p3 \;\rightarrow\; p1 \triangleright (p2 \triangleright p3)$$

Discussion

- Our exercise did not aimed at assessing whether confined separation logic is enough for reasoning about confinement in object-oriented programs
- but to illustrate how the PF-transform helps to build a quite flexible framework for further extending/changing the logic, if necessary.
- Ther framework is parametric on the shapes of both heap and store
- Each shape has its own structural membership easy to calculate:

Background: PF-membership

$$
\begin{aligned}
\in_{\mathsf{K}} &\stackrel{\mathrm{def}}{=} \bot \\
\in_{\mathsf{Id}} &\stackrel{\mathrm{def}}{=} id \\
\in_{\mathsf{F}\times\mathsf{G}} &\stackrel{\mathrm{def}}{=} (\in_{\mathsf{F}} \cdot \pi_1) \cup (\in_{\mathsf{G}} \cdot \pi_2) \\
\in_{\mathsf{F}+\mathsf{G}} &\stackrel{\mathrm{def}}{=} [\in_{\mathsf{F}}, \in_{\mathsf{G}}] \\
\in_{\mathsf{F}\cdot\mathsf{G}} &\stackrel{\mathrm{def}}{=} \in_{\mathsf{G}} \cdot \in_{\mathsf{F}}
\end{aligned}
$$

Pointfree reasoning is useful in various aspects

Handy way of carrying out semantics-level reasoning, since, quoting [Reynolds, 2002]:

> *"[...] In its present state separation logic is not only theoretically incomplete but pragmatically incomplete."*

Clearly:

- This gives room for the PF-relational model to be used explicitly wherever the logic isn't expressive enough.
- In the PF-style we can calculate directly with semantic denotations as objects (no quantification over addresses, atoms, etc)

Pointfree reasoning is useful in various aspects

Handy characterization of [Reynolds, 2002] **classes** of assertions:

- **Intuitionistic** p iff $[\![p]\!] = \supseteq \cdot [\![p]\!]$.
- **Strictly-exact** p iff $[\![p]\!]$ is simple, that is $[\![p]\!] \cdot [\![p]\!]^\circ \subseteq id$
- **Pure** p iff $[\![p]\!]$ is a right-condition, ie. $[\![p]\!] = \top \cdot \Phi$ for some Φ

Pure assertions do not depend on the heap, thus the two conjunctions collapse. For example, we get,

$$(p \wedge q) * r \leftrightarrow p \wedge (q * r) \qquad \text{when } p \text{ is pure}$$

Example of calculation about pure assertions

$$
\begin{array}{cl}
& [\![p \wedge (q * r)]\!] \\
= & \qquad \{ \;\; p := \top \cdot \Phi \text{ since } p \text{ is pure } \} \\
& \top \cdot \Phi \cap (*) \cdot \langle [\![q]\!], [\![r]\!] \rangle \\
= & \qquad \{ \;\; \text{right-conditions: } \Phi \cdot R = R \cap \Phi \cdot \top \;\; \} \\
& (*) \cdot \langle [\![q]\!], [\![r]\!] \rangle \cdot \Phi \\
= & \qquad \{ \;\; \text{splits: } \langle R, S \rangle \cdot \Phi = \langle R, S \cdot \Phi \rangle \equiv \Phi \text{ coreflexive [Oliveira, 2007]} \;\; \} \\
& (*) \cdot \langle [\![q]\!] \cdot \Phi, [\![r]\!] \rangle \\
= & \qquad \{ \;\; \text{right-conditions} \;\; \} \\
& (*) \cdot \langle \top \cdot \Phi \cap [\![q]\!], [\![r]\!] \rangle \\
= & \qquad \{ \;\; \top \cdot \Phi := p \text{ ; definitions} \;\; \} \\
& [\![(p \wedge q) * r]\!]
\end{array}
$$

Closing

- High-valued programmers are heavy users of logic: which entails the need for earlier introduction and explicit use of logic in middle and high school
- but a heavy use of logic entails the need for more concise ways of expression and notations amenable to formal, systematic manipulation.

[With symbols] when controversies arise, there will be no more necessity for disputation between two philosophers than between two accountants. Nothing will be needed but that they should take pen and paper, sit down with their calculators, and say 'Let us calculate'.

G. W. Leibniz (1646-1716)