Constructing the Views Framework (yes, again! ☺)

Stephan van Staden

Outline

- The Views framework
- The motivation for constructing it again
- Formal languages
- Constructing the program logic
- Constructing operational calculi
- Soundness
- Conclusion

The Views framework (1)

Unifies several compositional program logics for reasoning about concurrent programs

- Concurrent separation logic
- Concurrent abstract predicates
- Rely-guarantee
- Owicki-Gries

Views are abstract versions of the assertions of a program logic

- They can be composed and satisfy certain laws
- They are mapped to sets of states

The Views framework (2)

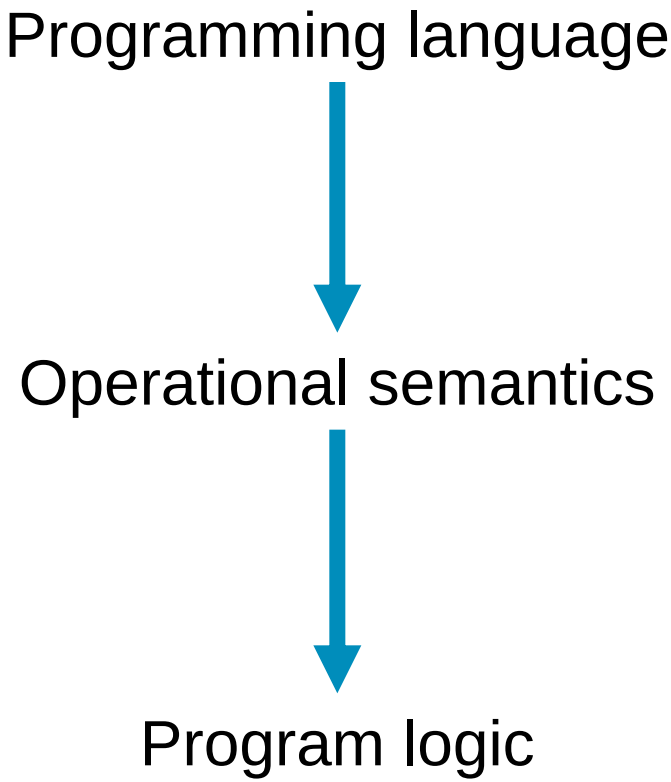
The abstract properties of views justify the soundness of inference rules

- E.g. the “frame rule” and “concurrency rule”

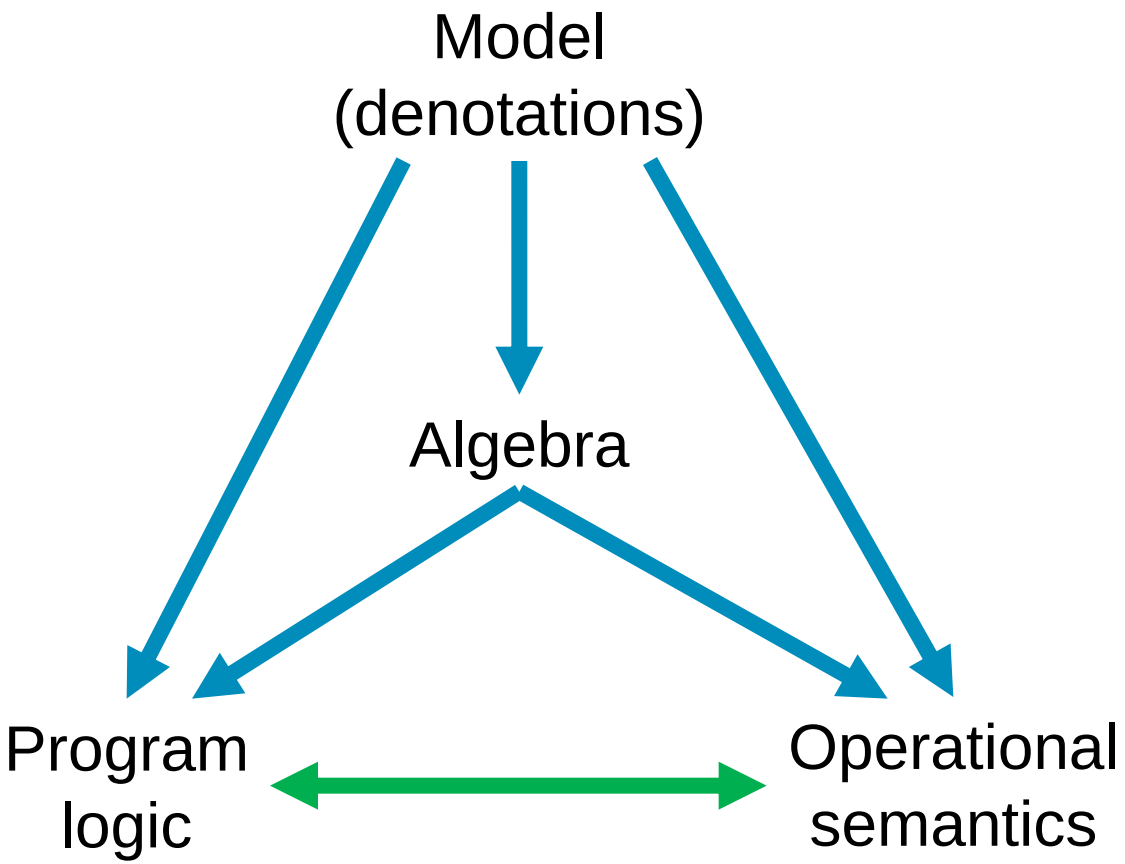
Program logics use different instantiations of views. Their inference rules look rather different, BUT deep down the reasoning is the same

In this sense, the views framework captures the essence of these seemingly different techniques in a unified formalism - imo a beautiful result!

Its metatheory in the POPL'13 paper



But I wanted to show it differently…



A complementary view of Views Framework

More semantic and simpler in a sense:

- No fixed syntax for programs: treat them as semantic objects (formal languages over state pairs)
- All judgements have direct definitions; all inference rules are theorems:
  - Views program logic is constructed from Hoare logic in a stepwise fashion. Completely decoupled from operational rules
  - Operational judgements also defined directly. Rules are derived and not postulated
- Soundness is independent of the choice of operational rules; views logic is sound because Hoare logic is
- Proofs do not inspect syntax or derivations

Formal languages (1)

Operators / notions:

– skip	the language {[]}	does nothing
– ;	language concatenation	sequencing
– ||	language shuffle	concurrency
– ∪	language union	nondet choice
– ⊆	language inclusion	refinement

Formal languages (2)

We mostly consider formal languages over pairs of states (i.e. the alphabet is $\Sigma \times \Sigma$)

- a word is called a *trace*
- an *atom* is a language whose traces have length 1
- a trace is *consistent* when the states between adjacent pairs are equal, e.g. $[(\sigma,\sigma_1),(\sigma_1,\sigma_2),(\sigma_2,\sigma')]$
- Incon is the set of all inconsistent traces
- end(σ) is the set of all consistent traces that end in state σ
- end(S) is the set of all consistent traces that end in some state in S

Constructing the program logic

Stepwise, from first principles:

- Hoare logic
- Basic views calculus
- Framing calculus
- Full views calculus

Hoare logic

$S \{P\} S' \equiv end(S) ; P \subseteq end(S') \cup Incon$

Direct semantic definition. Rules are theorems:

$S \{skip\} S$

Proof: $end(S);skip = end(S) \subseteq (end(S) \cup Incon)$

$S \{P\} S' \;\&\; S' \{Q\} S'' \Rightarrow S \{P;Q\} S''$

Proof: $end(S);(P;Q) \subseteq (end(S);P);Q \subseteq (end(S') \cup Incon);Q$
$\subseteq (end(S');Q \cup Incon;Q) \subseteq (end(S');Q \cup Incon)$
$\subseteq (end(S'') \cup Incon \cup Incon) = end(S'') \cup Incon$

Basic views calculus

Assume a set *Views*

Each view v is mapped to a set of states ⌊v⌋

The basic views calculus uses views for assertions:
v <P> v' ≡ ⌊v⌋ {P} ⌊v'⌋

Rules of the basic calculus follow immediately from those of Hoare logic

E.g. v <P> v' & v' <Q> v'' ⇒ v <P;Q> v''

Framing calculus

Views can be combined with ✱

✱ is associative and commutative

The framing calculus requires “frame preservation”:
v [P] v’ ≡ v <P> v’ & ∀v’’. v✱v’’ <P> v’✱v’’

Stronger judgement: v [P] v’ ⇒ v <P> v’

New rule: v [P] v’ ⇒ v✱v’’ [P] v’✱v’’
Proof: By the associativity of ✱ and elementary logic

Full views calculus (1)

For compositional reasoning about concurrency, the *intermediate steps* should also preserve views

- programs can't interfere to invalidate each other's views

To this end, the full views calculus reasons about commands = formal languages over atoms

$\{v\}\ C\ \{v'\} \quad \equiv \quad \forall as \in C.\ \ v\ \#as\#\ v',$ where

$v\ \#[]\#\ v' \quad \equiv \quad v\ [skip]\ v'$

$v\ \#a{:}as\#\ v' \quad \equiv \quad \exists v''.\ v\ [a]\ v''\ \ \&\ \ v''\ \#as\#\ v'$

Stronger judgement than framing calculus

Full views calculus (2)

New rule: $\{v_1\}\ C_1\ \{v_1'\}\ \ \&\ \ \{v_2\}\ C_2\ \{v_2'\}\ \ \Rightarrow$
$\{v_1 * v_2\}\ C_1 || C_2\ \{v_1' * v_2'\}$

Proof: The frame and sequence rules of the framing calculus and the commutativity of $*$ imply

$v_1\ \#as_1\#\ v_1'\ \ \&\ \ v_2\ \#as_2\#\ v_2'\ \ \&\ \ as \in as_1 \otimes as_2\ \ \Rightarrow$
$(v_1 * v_2)\ \#as\#\ (v_1' * v_2')$

Corollary: $\{v\}\ C\ \{v'\}\ \ \Rightarrow\ \ \{v * v''\}\ C\ \{v' * v''\}$

Proof: Apply the concurrency rule to $\{v\}C\{v'\}$ and $\{v''\}skip\{v''\}$. The result follows by $C||skip = C$.

Constructing operational calculi (1)

Operational calculi help to discover executions

Not special or somehow fundamental here

Define each operational judgment directly and prove that inference rules are valid (no postulation!)

Big-step operational judgement:
$\langle P, \sigma \rangle \rightarrow \sigma' \equiv \exists t \in end(\sigma), t' \in end(\sigma'). \{t\};P \supseteq \{t'\}$

Example theorems: 1) $\langle skip, \sigma \rangle \rightarrow \sigma$

2) $\langle P, \sigma \rangle \rightarrow \sigma' \;\&\; \langle Q, \sigma' \rangle \rightarrow \sigma'' \Rightarrow \langle P;Q, \sigma \rangle \rightarrow \sigma'$

Constructing operational calculi (2)

Small-step operational judgement:

$\langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \quad \equiv$

$\exists Q \in \text{Actions}. \quad P \supseteq Q;P' \quad \& \quad \langle Q, \sigma \rangle \rightarrow \sigma'$

Stronger: $\langle P, \sigma \rangle \rightarrow^* \langle \text{skip}, \sigma' \rangle \quad \Rightarrow \quad \langle P, \sigma \rangle \rightarrow \sigma'$

Example theorems:

- $\langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \quad \Rightarrow \quad \langle P \| Q, \sigma \rangle \rightarrow \langle P' \| Q, \sigma' \rangle$
- $\langle P, \sigma \rangle \rightarrow \langle \text{skip}, \sigma' \rangle \quad \Rightarrow \quad \langle P \| Q, \sigma \rangle \rightarrow \langle Q, \sigma' \rangle$
- $\langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \quad \Rightarrow \quad \langle P;Q, \sigma \rangle \rightarrow \langle P';Q, \sigma' \rangle$

Partial correctness

The construction of the program logics never referred to operational rules. Nonetheless:
$S \{P\} S' \Leftrightarrow (\forall \sigma \in S.\ \forall \sigma'.\ \langle P, \sigma\rangle \rightarrow \sigma' \Rightarrow \sigma' \in S')$

$S \{P\} S' \Rightarrow (\forall \sigma \in S.\ \forall \sigma'.\ \langle P, \sigma\rangle \rightarrow^* \langle skip, \sigma'\rangle \Rightarrow \sigma' \in S')$

The other program logic judgements are stronger, and hence also correct w.r.t. execution!

No coinduction, no mention of particular rules, no inspection of the program syntax

Summary

Explained the foundations of the Views Framework in a different way

- semantic: programs are not syntactic objects; they are modelled as sets of traces
- all the laws of CKA are valid
- incremental development of calculi from first principles
- program logic and operational semantics are decoupled
- partial correctness holds - reduced to the soundness of Hoare logic

Complements the POPL treatment

Final comments

That it could be explained in this way adds to the credit of the Views Framework

- elegant and general

Similar ideas could be used in the future to construct new program logics

- prototype them in a lightweight semantic setting
- use basic logics as a foundation for advanced ones

Is it practical? To which extent can generic semantic settings help to construct/explain program logics? E.g. weak memory, message passing, …