Building Verification Components from Algebraic Principles

Lectures at Universitetet i Oslo 2019

Georg Struth

University of Sheffield

These Lectures

- building components for program correctness in Isabelle/HOL
  - ▹ program construction (by transformation/refinement)
  - ▹ program verification
- simple principled approach that separates control/data flow
  - ▹ abstract algebra for control
  - ▹ concrete semantics for data domains
- in detail: simple construction/verification components for
  - ▹ while programs
  - ▹ based on Hoare logic
  - ▹ Morgan's refinement calculus
  - ▹ predicate transformers

all components correct by construction

Principled Approach

algebra	intermediate semantics	concrete semantics
control flow	abstract data flow	concrete data flow
control flow logic	intermediate logic	verification tool

First Instance

KAT	relational KAT	relational KAT over store
control flow	abstract data flow	concrete data flow
propositional Hoare logic	relational verification conditions	verification conditions based on Hoare logic

Second Instance

KAD	relational KAD	relational KAD over store
control flow	abstract data flow	concrete data flow
transformers (PDL)	relational verification conditions	verification conditions based on transformers

Plan

lectures

1. algebraic foundations (Kleene algebras)
2. mathematical components for these algbras
3. construction/verification components for sequential programs
4. extension to hybrid programs (ongoing research)

exercises

depending on interest we could look at

- algebraic reasoning about programs
- verification examples

Algebraic Foundations

While-Programs

syntax (regular operations)

$+$	nondeterministic choice
$\cdot$	sequential composition
$*$	finite iteration
0	failure/abort
1	skip

abstract semantics

regular expressions $t ::= 0 \mid 1 \mid a \in \Sigma \mid t + t \mid t \cdot t \mid t^*$

Kleene algebra $(K, +, \cdot, 0, 1, ^*)$

Kleene algebra is algebra of regular expressions

Dioids

definition

a dioid (idempotent semiring) is a structure $(S, +, \cdot, 0, 1)$ where

- $(S, +, 0)$ is a semilattice with least element 0
- $(S, \cdot, 1)$ is a monoid
- multiplication distributes over addition
- zero is left/right annihilator

$$x + (y + z) = (x + y) + z \qquad x + y = y + x \qquad x + 0 = x \qquad x + x = x$$

$$x(yz) = (xy)z \qquad x1 = x \qquad 1x = x$$

$$x(y + z) = xy + xz \qquad (x + y)z = xz + yz$$

$$x0 = 0 \qquad 0x = 0$$

Dioids

natural order

- $(S, +)$ is semilattice with partial order $x \leq y \Leftrightarrow x + y = y$
- regular operations preserve order (e.g. $x \leq y \Rightarrow z + x \leq z + y$)
- 0 is least element

opposition

- map $(-)^\partial : S \to S$ swaps order of multiplication

$$0^\partial = 0 \quad 1^\partial = 1 \quad (x + y)^\partial = x^\partial + y^\partial \quad (x \cdot y)^\partial = y^\partial \cdot x^\partial$$

- $\partial[S]$ is again a dioid—the opposite dioid

Kleene Algebras

definition

a Kleene algebra is a dioid expanded by star operation that satisfies

$$\begin{aligned} 1 + xx^* &\leq x^* & \qquad z + xy \leq y &\Rightarrow x^*z \leq y \\ 1 + x^*x &\leq x^* & \qquad z + yx \leq y &\Rightarrow zx^* \leq y \end{aligned}$$

intuition

- x^*z is least solution of affine linear inequality $z + xy \leq y$
- zx^* is least solution of affine linear inequality $z + yx \leq y$

Models of Kleene Algebra

for programming

- binary relations form KAs
- our verification components are based on this model

for proofs

- (regular) languages form KAs
- regular expressions are ground terms in KA signature
- KAs are complete for regular expression equivalence
- variety of KA is decidable via automata (PSPACE-complete)

Language Kleene Algebras

let Σ^* denote free monoid with empty word ε over Σ

definition

a language is a subset of Σ^*

theorem (soundness)

- $(2^{\Sigma^*}, \cup, \cdot, {}^*, \emptyset, \{\varepsilon\})$ forms the full language KA over Σ, where

$$X \cdot Y = \{vw \mid v \in X \wedge w \in Y\}$$
$$X^* = \bigcup_{i \geq 0} X^i$$

  and $X^0 = \{\varepsilon\}, \quad X^{i+1} = XX^i$

- any subalgebra forms a language KA

Regular Language Kleene Algebras

definition

KA morphism $L : T_{\mathsf{KA}}(\Sigma) \to 2^{\Sigma^*}$ generates regular languages over Σ:

$$L\,0 = \emptyset \qquad L\,1 = \{\varepsilon\} \qquad L\,a = \{a\} \text{ for } \ a \in \Sigma$$

$$L\,(s + t) = L\,s \cup L\,t \qquad L\,(s \cdot t) = L\,s \cdot L\,t \qquad L\,(t^*) = (L\,t)^*$$

theorem (soundness)

- regular languages over Σ form KA
- in particular $\mathsf{KA} \vdash s = t \Rightarrow L\,s = L\,t$ for all $s, t \in T_{\mathsf{KA}}(\Sigma)$

Completeness of Kleene Algebra

theorem [Kozen]

$\mathsf{KA} \vdash s = t \Leftrightarrow L\,s = L\,t$ for all $s, t \in T_{\mathsf{KA}}(\Sigma)$

consequences

- regular languages over Σ are generated freely by Σ in variety of KA
- KA axiomatises equational theory of regular expressions
  (as induced by regular language identity)
- equational theory of KA decidable (by automata)

Relation Kleene Algebra

binary relation

subset of $A \times A$

$$R = \{(a, b) \mid a, b \in A\}$$

theorem (soundness)

- $(2^{A\times A}, \cup, \cdot, \emptyset, id, ^*)$ forms full relation Kleene algebra over A, where

$$id = \{(a, a) \mid a \in A\}$$

$$R \cdot S = \{(a, b) \mid \exists c.(a, c) \in R \wedge (c, b) \in S\}$$

$$R^* = \bigcup_{i\geq 0} R^i \quad \text{(reflexive transitive closure of } R\text{)}$$

- every subalgebra forms a relation Kleene algebra

Relation Kleene Algebra

theorem (completeness)

if $s = t$ holds in class of all relation KAs, then $\mathsf{KA} \vdash s = t$

consequence

- equational theory of relation KA is decidable via automata
- this makes KA interesting for program construction/verification

Beyond Equations

quasivariety of KA

undecidable (uniform word problem for semigroups)

quasivariety of regular expressions

KA does not work

- $x^2 = 1 \Rightarrow x = 1$ holds in language KA
- but not for relation $R = \{(0,1),(1,0)\}$, which is in KA (with $\{(0,0),(1,1)\}$, $\emptyset$, etc.)

program construction/verification requires
reasoning under assumptions

Kleene Algebras and Sequential Programs

program analysis

- reason about actions and propositions/states
- propositions can be tests or assertions

relational semantics

- relations model i/o-behaviour of programs on state spaces
- elements $p \leq 1$ represent sets of states/propositions
  - px yields all x-transitions that start from states in p
  - xp, by opposition, yields all x-transitions that end in states in p
- these element form boolean subalgebras
  (join is $+$, meet is $\cdot$, 0 is least and 1 greatest element)
- they can be used as tests or assertions in relational semantics

Kleene Algebras with Tests

abstraction

use KA for actions and BA (test algebra) for propositions

definition [Manes/Kozen]

two-sorted structure $(K, B, +, \cdot, \neg, 0, 1, ^*)$

- BA $(B, +, \cdot, \neg, 0, 1)$ embedded into K
- K models actions, B tests/assertions
- partial operation $\neg$ defined on subalgebra B

Models of KAT

relation KAT

- binary relations form KATs
  - test algebra formed by subsets of id
  - these subidentites are isomorphic to sets of states
- every relation KAT is isomorphic to relation KA
- hence equational theory of relation KAT is still PSPACE-complete

guarded string KAT

- essentially trace KAT in which propositions and actions alternate
- P formed by atoms of free BA generated by finite set G
- guarded strings (and traces) form words over enlarged alphabet
- this implies completeness of KAT for guarded regular languages

KAT and Imperative Programs

algebraic program semantics

while programs (no assignment):

$$\begin{aligned}
\textbf{abort} &= 0\\
\textbf{skip} &= 1\\
x;y &= xy\\
\textbf{if } p \textbf{ then } x \textbf{ else } y \textbf{ fi} &= px + \neg py\\
\textbf{while } p \textbf{ do } x \textbf{ od} &= (px)^*\neg p
\end{aligned}$$

Kleene Algebra with Isabelle

—demo—

Verification Component based on KAT

Outline

KAT	relational KAT	relational KAT with store
control flow	abstract data flow	concrete data flow
propositional Hoare logic	relational verification conditions	verification conditions from Hoare logic

Verification Component Outline

approach

1. use KAT as abstract algebraic semantics for while-programs
2. define validity of Hoare triples in KAT
3. derive rules of Hoare logic without assignment in KAT
4. derive assignment rule in relation KAT over program store
5. use Isabelle polymorphism to integrate arbitrary data domains
6. use KAT/Hore logic for verification condition generation
7. use domain-specific Isabelle components to verify programs

tool correct by construction

Hoare Triples in KAT

validity of Hoare triple

$$\vdash \{p\}\ x\ \{q\} \iff px\neg q = 0$$

intuition (partial correctness)

if program x is executed from state where p holds and if x terminates, then q must hold in state where x terminates

in relation KAT

$$\begin{aligned}
&\forall s, s'.\ (s, s') \notin px\neg q \\
&\Leftrightarrow \forall s, s'.\ \neg((s, s) \in p \wedge (s, s') \in x \wedge (s', s') \in \neg q) \\
&\Leftrightarrow \forall s, s'.\ ((s, s) \in p \wedge (s, s') \in x) \Rightarrow (s', s') \in q
\end{aligned}$$

Propositional Hoare Logic

propositional Hoare logic means Hoare logic without assignment rule

theorem [Kozen]

inference rules of PHL derivable in KAT

$$\begin{aligned}
&& \vdash \{p\}\ \textbf{skip}\ \{p\} \\
p \leq p' \wedge q' \leq q \wedge\ \vdash \{p'\}\ x\ \{q'\} &\ \Rightarrow\ & \vdash \{p\}\ x\ \{q\} \\
\vdash \{p\}\ x\ \{r\} \wedge\ \vdash \{r\}\ y\ \{q\} &\ \Rightarrow\ & \vdash \{p\}\ x; y\ \{q\} \\
\vdash \{pb\}\ x\ \{q\} \wedge\ \vdash \{p\neg b\}\ y\ \{q\} &\ \Rightarrow\ & \vdash \{p\}\ \textbf{if}\ b\ \textbf{then}\ x\ \textbf{else}\ y\ \textbf{fi}\ \{q\} \\
\vdash \{pb\}\ x\ \{p\} &\ \Rightarrow\ & \vdash \{p\}\ \textbf{while}\ b\ \textbf{do}\ x\ \textbf{od}\ \{\neg bp\}
\end{aligned}$$

Store and Assignments

simple store in Isabelle

- stores formalised as functions from variable to values
- generic for any type of data (KAT/relation KAT polymorphic)
- variables formalised as strings
- values can have any type

assignment

$$(v := e) \; = \; \{(s, \mathit{fun\_upd}\ s\ v\ (e\ s)) \mid s \in S\}$$

theorem

all inference rules of HL are derivable in relation KAT with store

$$\vdash \{Q[e/v]\}\ (v := e)\ \{Q\}$$

Verification Condition Generation

Hoare logic

- one structural rule per program construct
- can be programmed as hoare tactic in Isabelle
- blasts away entire control structure

derivable rules

$$p \leq p' \wedge \ \vdash \{p'\}\ x\ \{q\} \Rightarrow \vdash \{p\}\ x\ \{q\}$$

$$p \leq i \wedge \neg p i \leq q \wedge \ \vdash \{ib\}\ x\ \{i\} \Rightarrow \vdash \{p\}\ \textbf{while}\ b\ \textbf{inv}\ i\ \textbf{do}\ x\ \textbf{od}\ \{q\}$$

Verification Component with Isabelle

control flow

- Isabelle libraries for KAT include PHL
- hoare tactic generates verification conditions automatically from HL

data flow

- modelled generically in relation KAT (with store)
- shallow embedding of simple while-language
- analysed with Isabelle's provers
- functional data types often impersonate imperative data structures
- could use data refinement as justification. . .

— demo —

Refinement Component based on KAT

Refinement KAT

definition

refinement KAT is KAT expanded by specification statement $[\ ,\ ]$ and axiom

$$\vdash \{p\}\ x\ \{q\} \Leftrightarrow x \leq [p, q]$$

theorem

$(2^{A\times A}, B, \cup, \circ, [\_, \_],^*, \neg, \emptyset, id)$ forms rKAT with

$$[P, Q] = \bigcup\{R \subseteq A \times A \mid \vdash \{P\}\ R\ \{Q\}\}$$

Propositional Refinement Calculus

theorem

Morgan's propositional refinement laws are derivable in rKAT ($\sqsubseteq \; = \; \geq$)

$$\begin{aligned}
p \leq q \Rightarrow [p, q] &\sqsubseteq \textbf{skip} \\
p \leq p' \wedge q' \leq q \Rightarrow [p, q] &\sqsubseteq [p', q'] \\
[0, 1] &\sqsubseteq x \\
x &\sqsubseteq [1, 0] \\
[p, q] &\sqsubseteq [p, r]; [r, q] \\
[p, q] &\sqsubseteq \textbf{if } \; b \textbf{ then } [bp, q] \textbf{ else } [\neg bp, q] \textbf{ fi} \\
[p, \neg bp] &\sqsubseteq \textbf{while } b \textbf{ do } [bp, p] \textbf{ od}
\end{aligned}$$

no frame laws for local variables

Refinement Calculus

theorem

assignment laws derivable in relation rKAT

$$
\begin{aligned}
P \subseteq Q[e/'x] &\Rightarrow [P, Q] \sqsubseteq (v := e)\\
Q' \subseteq Q[e/'x] &\Rightarrow [P, Q] \sqsubseteq [P, Q'];(v := e)\\
P' \subseteq P[e/'x] &\Rightarrow [P, Q] \sqsubseteq (v := e);[P'; Q]
\end{aligned}
$$

— demo —

Verification Component based on Predicate Transformers

Outline

KAD	relational KAD	relational KAD over store
control flow	abstract data flow	concrete data flow
transformers (PDL)	relational verification conditions	verification conditions based on transformers

Verification Component Outline

approach

1. use KAD as abstract algebraic semantics for while-programs
2. define partial correctness specification in KAD
3. derive predicate transformer laws without assignment in KAD
4. derive assignment law in relation KAD over program store
5. use Isabelle polymorphism to integrate arbitrary data domains
6. use KAD predicate transformer laws for vcg
7. use domain-specific Isabelle components to verify programs

tool correct by construction

Adding Modalities

motivation

- many applications require different approach to actions/propositions
- systems dynamics by action on state space $K \to B \to B$
- computational logics (e.g. PDL) “use” KAs, but how precisely?

modal approach

- actions/propositions via relational (aka Kripke) frames
- modal operators via preimages/images $|x\rangle p$ / $\langle x|p$
- preimages/images via axioms for domain/codomain

State Transitions

in KAT

"terminating program x from store p goes to store q" expressed as

$$px \leq xq \quad \text{or equivalently} \quad px\neg q = 0$$

alternative

"q contains x-image of p"

how can we model relational (pre)images directly in semirings?

Adding Modalities

task

- abstract equational axioms for relational domain

$$d\,x = \{(p,p) \mid \exists q.\ (p,q) \in x\}$$

- algebraic definition of relational modalities
  - $\langle x|p = \mathrm{ran}\,(px)$ is image of p under x
  - $|x\rangle p = \mathrm{dom}\,(xp)$ by opposition, is preimage of p under x

two approaches

1. domain as map $K \to B$ in KAT
2. domain as endo $S \to S$ that induces B in dioid

Domain Semirings

domain semiring

semiring S with $d : S \to S$ that satisfies

$$x + d\,x \cdot x = d\,x \cdot x \qquad d\,(x \cdot y) = d\,(x \cdot d\,y) \qquad d\,(x + y) = d\,x + d\,y$$

$$d\,x + 1 = 1 \qquad d\,0 = 0$$

lemma

domain semirings are dioids

proposition

$d^2 = d$ (domain is retraction), so $x \in d[S] \Leftrightarrow d\,x = x$

Domain Algebra

theorem

$(d[S], +, \cdot, 0, 1)$ is bounded DL (d induces state space)

notation

- $(d[S], +, \cdot, 0, 1)$ is called domain algebra of S
- $p, q, r \ldots$ for domain elements

modalities

$$|x\rangle y = d\,(xy) \qquad\qquad \langle x|y = r\,(yx)$$

how can we obtain boolean state space?

Antidomain Semirings

antidomain semiring

semiring S with endo $a : S \to S$ that satisfies

$$a\,x \cdot x = 0 \qquad a\,(x \cdot y) \leq a\,(x \cdot a^2\,y) \qquad a^2\,x + a\,x = 1$$

remarks

- ∘ domain definable as $d = a^2$ (boolean complement)
- ∘ $a[S](= d[S])$ generated is maximal BA in $[0, 1]$
- ∘ simple axioms induce rich modal calculus...

diamonds again

$$|x\rangle y = d\,(xy) \qquad \langle x|y = r\,(yx)$$

Dualities for Modalities

$$|x\rangle p = d(xp) \qquad |x]p = a(xa(p))$$

$$\langle x|p = r(px) \qquad [x|p = ar(ar(p)x)$$

Dualities for Modalities

$$\begin{tikzcd}
{|x\rangle p \ = d(xp)} \arrow[d, leftrightarrow, "\text{opposition}"'] & {|x]p \ = a\,(xa(p))} \arrow[d, leftrightarrow, "\text{opposition}"] \\
{\langle x|p \ = r(px)} & {[x|p \ = ar(ar(p)x)}
\end{tikzcd}$$

Dualities for Modalities

$$\begin{tikzcd}
|x\rangle p \; = d(xp) \arrow[d, leftrightarrow, "\text{opposition}"'] & |x]p \; = \neg|x\rangle\neg p \arrow[d, leftrightarrow, "\text{opposition}"] \\
\langle x|p \; = r(px) & {[x|p} \; = \neg\langle x|\neg p
\end{tikzcd}$$

Dualities for Modalities

$$\begin{tikzcd}
|x\rangle p \arrow[r, leftrightarrow, "\text{De Morgan}"] \arrow[d, leftrightarrow, "\text{opposition}"'] & |x]p \arrow[d, leftrightarrow, "\text{opposition}"] \\
\langle x|p \arrow[r, leftrightarrow, "\text{De Morgan}"'] & [x|p
\end{tikzcd}$$

Dualities for Modalities

$$\begin{tikzcd}
|x\rangle p \arrow[r, leftrightarrow, "\text{De Morgan}"] \arrow[d, leftrightarrow, "\text{opposition}"'] & |x]p \arrow[d, leftrightarrow, "\text{opposition}"] \\
\langle x|p \arrow[r, leftrightarrow, "\text{De Morgan}"'] & [x|p
\end{tikzcd}$$

- conjugations

$$(|x\rangle p)q = 0 \Leftrightarrow p(\langle x|q) = 0 \qquad (|x]p)q = 0 \Leftrightarrow p([x|q) = 0$$

- adjunctions

$$\langle x|p \leq q \Leftrightarrow p \leq |x]q \qquad |x\rangle p \leq q \Leftrightarrow p \leq [x|q$$

Dualities for Modalities

demodalisation

$$
\begin{aligned}
|x\rangle p \le q \;&\Leftrightarrow\; \neg q x p \le 0 \;\Leftrightarrow\; p \le [x|q \\
\langle x| p \le q \;&\Leftrightarrow\; p x \neg q \le 0 \;\Leftrightarrow\; p \le |x] q
\end{aligned}
$$

properties

- conjugations/adjunctions as theorem generators
- dualities as theorem transformers

MKA and KAT

theorem

every KA with antidomain is a KAT (. . . but not conversely)

a Hoare logic is expressive if for each command x and postcondition q the weakest liberal precondition is definable

theorem

MKA is expressive for Hoare logic (. . . KAT isn't)

proof

- ○ $|x]q = wlp(x,q)$ for each command x and postcondition q
- ○ in relational model $|R]Q = \bigcup\{P \mid \{P\}\,R\,\{Q\}\}$

Control Elimination

partial correctness specification

$$p \leq |x]q$$

predicate transformer laws

- $|xy]q = |x]|y]q$
- $|\text{if } p \text{ then } x \text{ else } y]q = (\neg p + |x]q)(p + |y]q)$
- $p \leq i \wedge i\neg t \leq q \wedge it \leq |x]i \;\Rightarrow\; p \leq |\text{while } t \textbf{ inv } i \text{ do } x]q$

recursive wp/vc computation

Control Elimination

partial correctness specification

$$p \leq |x]q$$

predicate transformer laws

- $|xy]q = |x]|y]q$
- $|\text{if } p \text{ then } x \text{ else } y]q = (\neg p + |x]q)(p + |y]q)$
- $p \leq i \wedge i\neg t \leq q \wedge it \leq |x]i \;\Rightarrow\; p \leq |\text{while } t \ \textbf{inv} \ i \text{ do } x]q$

but what about assignment?

Data Integration

1. relational model
   - $(\mathrm{Rel}(X), \cup, ;, a, ar, \emptyset, Id, ^{*})$ forms relation MKA over X with
     - $a\,R = \{(a,a) \mid \neg\exists b.\ (a,b) \in R\}$
     - $ar\,R = \{(b,b) \mid \neg\exists a.\ (a,b) \in R\}$
   - subidentities $\{P \in \mathrm{Rel}(X) \mid P \subseteq Id\}$ form boolean subalgebra
2. relational store model
   - store as function $V \to E$ from variables to values
   - assignments defined by $(v := e) = \{(s, s[(e\,s)/v]) \mid s \in E^{V}\}$
   - wp law for assignments derivable: $|v := e]\lceil Q\rceil = \lceil \lambda s.\ Q\,s[(e\,s)/v]\rceil$

Modalities vs Predicate Transformers

relation $R \subseteq X \times Y$ gives rise to three transformers:

- state transformer $f_R : X \to 2^Y$ defined by

$$f_R\,x = \{y \mid (x, y) \in R\}$$

- conjunctive predicate transformer $|R] : 2^Y \to 2^X$ defined by

$$|R]P = \{x \mid f_R\,x \subseteq P\}$$

- disjunctive predicate transformer $\langle R| : 2^X \to 2^Y$ defined by

$$\langle R|\,p = \bigcup\{f_R\,x \mid x \in p\}$$

Isabelle Verification Component

—demo—

SP for Free

$$\begin{tikzcd}
{|x\rangle p} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & {|x]p = wlp(x,p)} \arrow[d, leftrightarrow] \\
{sp(x,p) = \langle x|p} \arrow[r, leftrightarrow] \arrow[ur, leftrightarrow, red] & {[x|p}
\end{tikzcd}$$

- adjunction $\langle x|p \leq q \Leftrightarrow p \leq |x]q$ dualises wp-laws
- Floyd-style assignment law works well in this setting
- useful for symbolic execution

Hoare Logic and Refinement

$$\begin{tikzcd}
\langle x|p \leq q \arrow[dr, leftrightarrow] & & p \leq |x]q \arrow[dl, leftrightarrow] \\
& px \leq xq \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
\{p\}x\{q\} & & x \leq [p, q]
\end{tikzcd}$$

Hoare Logic and Refinement

$$\begin{tikzcd}
\langle x|p \le q \arrow[dr, leftrightarrow] & & p \le |x]q \arrow[dl, leftrightarrow] \\
& px \le xq \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
\{p\}x\{q\} & & x \le [p,q]
\end{tikzcd}$$

- PHL rules derivable in MKA
- Morgan-style refinement calculus derivable in refinement MKA
- but MKA is refinement calculus
- assignment rules derivable in relational store model
- we link into KAT/rKAT components instead

—demo—

Literature

- algebra and HL
  - Kozen, *On Hoare Logic and Kleene Algebra with Tests*
  - Desharnais, Struth, *Internal Axioms for Domain Semirings*
  - Möller, Struth, *Algebras of Modal Operators and Partial Correctness*
- refinement calculi
  - Morgan, *Programming from Specifications*
  - Back, von Wright, *Refinement Calculus: A Systematic Introduction*
- Isabelle formalisations
  - Armstrong, Gomes, Struth, *Building Program Construction and Verification Tools from Algebraic Principles*
  - Struth, *Hoare Semigroups*

Conclusion

- principled approach to program correctness tools in Isabelle
  - use algebra at control flow layer
  - link with relation/predicate transformer semantics and store
  - derived Hoare logics or refinement calculi
- all algebras used have decidable fragments
- sequential program verification works smoothly
- concurrency verification (still) more tedious
- prototyping fast, simple, adaptable
- resulting tools lightweight