Building Verification Tools with Isabelle

Lectures at Midlands Graduate School 2015

Georg Struth

University of Sheffield

These Lectures

- building tools for program correctness in Isabelle/HOL
  - program construction (by transformation/refinement)
  - program verification
- principled approach that separates control/data flow
  - algebra for control
  - set theory for data/memory domain
- in detail: simple construction/verification tools for
  - sequential programs
  - local reasoning with separation logic
  - concurrent programs with rely-guarantee method

all tools 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

Instance

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

Plan

lectures

1. algebraic foundations (from semirings to quantales)
2. brief introduction to Isabelle
3. construction/verification tools for sequential programs
4. extensions to separation logic/rely-guarantee

exercises

depending on interest we could look at

- algebraic reasoning about programs
- interactive proofs with Isabelle
- verification examples

Algebraic Foundations

– Lecture I –

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 (see Roy's lectures)

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

- $\partial(S)$ is again a dioid—the opposite dioid

Dioids

free dioids

- are isomorphic to sets of words (languages)
- distributivity laws yield normal forms
  - $(x + y)z = xz + yz$ yields trees
  - $x(y + z) = xy + xz$ pushes $+$-nodes towards root

free algebras/objects are discussed in detail in Roy's lectures

Kleene Algebras

definition

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

$$1 + xx^* \leq x^* \qquad z + xy \leq y \Rightarrow x^*z \leq y$$

$$1 + x^*x \leq x^* \qquad z + yx \leq y \Rightarrow zx^* \leq y$$

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 tools 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]

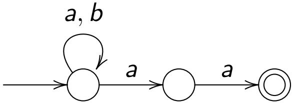
$\mathsf{KA} \vdash s = t \Leftrightarrow L(s) = L(t)$ for all $s, t \in T_{\mathsf{KA}}(\Sigma)$

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)

Completeness Proof

automata as matrices



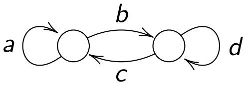
$$\left[\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} a+b & a & 0 \\ 0 & 0 & a \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right]$$

Completeness Proof

theorem

- let $M_n(K)$ denote $n \times n$ matrices over Kleene algebra K
- let Z_n and I_n be $n \times n$ zero and identity matrix
- then $(M_n(K), +, \cdot, ^*, Z_n, I_n)$ forms Kleene algebra

star



$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad M^* = \begin{pmatrix} f^* & f^* b d^* \\ d^* c f^* & d^* + d^* c f^* b d^* \end{pmatrix}$$

for $f = a + bd^*c$
partition larger matrices into submatrices with squares along diagonal

Example

in previous example split

$$\left(\begin{array}{c|cc} a+b & a & 0 \\ \hline 0 & 0 & a \\ 0 & 0 & 0 \end{array}\right)$$

and first compute

$$\begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix}^*$$

now $f = 0 + 0 = 0$, thus solution is

$$\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$$

now compute f for the 3×3-matrix

$$f = (a+b) + \begin{pmatrix} a & 0 \end{pmatrix} \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = a + b$$

Example

for the other parts of the star matrix we obtain

$$(a+b)^* \begin{pmatrix} a & 0 \end{pmatrix} \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} = (a+b)^* \begin{pmatrix} a & aa \end{pmatrix} = \begin{pmatrix} (a+b)^* a & (a+b)^* aa \end{pmatrix}$$

$$\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \end{pmatrix} (a+b)^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} (a+b)^* \begin{pmatrix} a & 0 \end{pmatrix} \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} = \cdots = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$$

this yields

$$\begin{pmatrix} a+b & a & 0 \\ 0 & 0 & a \\ 0 & 0 & 0 \end{pmatrix}^* = \begin{pmatrix} (a+b)^* & (a+b)^* a & (a+b)^* aa \\ 0 & 1 & a \\ 0 & 0 & 1 \end{pmatrix}$$

Completeness Proof

acceptance

automaton $[i, M, f]$ accepts language $L(i^T M^* f)$

example

$$
\begin{aligned}
&\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a+b & a & 0 \\ 0 & 0 & a \\ 0 & 0 & 0 \end{pmatrix}^* \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
&\quad = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} (a+b)^* & (a+b)^* a & (a+b)^* aa \\ 0 & 1 & a \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
&\quad = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} (a+b)^* aa \\ a \\ 1 \end{pmatrix} \\
&\quad = (a+b)^* aa
\end{aligned}
$$

Completeness Proof

lemma

every M can be written, for boolean matrices J and M_a, as

$$M = J + \sum_{a \in \Sigma} a \cdot M_a$$

remark

this yields characteristic matrix equation

lemma

every t satisfies $t = i^T M^* f$ for some automaton of this form
(replay one half of Kleene's theorem at matrix level)

Completeness Proof

theorem

the following constructions are theorems of Kleene algebra

1. for every $[i_1, M_1, f_1]$ exists ε-free $[i_2, M_2, f_2]$ with
   $i_1^T M_1^* f_1 = i_2^T M_2^* f_2$ (by simple matrix algebra)
2. for every such $[i_2, M_2, f_2]$ exists deterministic $[i_3, M_3, f_3]$ with
   $i_2^T M_2^* f_2 = i_3^T M_3^* f_3$ (by simulating subset construction)
3. for every such $[i_3, M_3, f_3]$ exists minimal $[i_4, M_4, f_4]$ with
   $i_3^T M_3^* f_3 = i_4^T M_4^* f_4$ (by dividing by Myhill-Nerode relation)

remark

(2) and (3) use coercion matrices to map $XM_i = M_{i+1}X$

Completeness Proof

theorem (completeness)

$$L(s) = L(t) \Rightarrow \mathsf{KA} \vdash s = t$$

proof

- let s and t denote the same regular language
- let $[i_s, M_s, f_s]$ and $[i_t, M_t, f_t]$ be minimal DFAs that satisfy

$$s = i_s^T M_s^* f_s \qquad t = i_t^T M_t^* f_t$$

- then they are isomorphic, so there is permutation matrix P with

$$M_s = P^T M_t P \qquad i_s = P^T i_t \qquad f_s = P^T f_t$$

- thus, in KA,

$$s = i_s^T M_s^* f_s = (P^T i_t)^T (P^T M_t P)^* (P^T f_t) = \cdots = t$$

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

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

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

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

- every subalgebra forms a relation Kleene algebra

Relation Kleene Algebra

theorem (completeness)

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

proof

Cayley map $c(L) = \{(x, xy) \mid x \in \Sigma^*, y \in L\}$ shows that relation/language KAs have same equational theory

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 form KA (with $\{(0,0),(1,1)\}$, $\emptyset$, etc.)

program construction/verification requires reasoning under assumptions

Path Kleene Algebras

paths in digraphs

- finite sequences of states from digraph $G = (V, E)$ related by edges
- empty path ε

path product

glue paths on initial/final state

$$\sigma.p \cdot p.\sigma' = \sigma.p.\sigma' \qquad \sigma.p \cdot q.\sigma' \quad \text{undefined}$$

powerset lifting

- $P_1 \cdot P_2 = \{\pi_1 \cdot \pi_2 \mid \pi_1 \in P_1, \pi_2 \in P_2, \pi_1 \cdot \pi_2 \text{ defined}\}$
- other operations as usual

theorem

(suitable) sets of paths form path Kleene algebras

Trace Kleene Algebras

traces

alternating sequence $p_0 a_0 p_1 a_1 p_2 \dots p_{n-2} a_{n-1} p_{n-1} \in (P \cdot A)^* \cdot P$

trace product

$\sigma.p \cdot p.\sigma' = \sigma.p.\sigma'$ $\sigma.p \cdot q.\sigma'$ undefined

powerset lifting

- $T_1 \cdot T_2 = \{\tau_1 \cdot \tau_2 \mid \tau_1 \in T_1, \tau_2 \in T_2, \tau_1 \cdot \tau_2 \text{ defined}\}$
- other operations as usual

theorem

(suitable) sets of traces form trace Kleene algebras

Relationship Between Models

relationship

essentially by forgetting structure in trace algebras

- path/language Kleene algebras forget actions/propositions
- relation Kleene algebras forget everything between endpoints

theorem

- (equational) properties are inherited by (relations), paths, languages
- equational theories are all the same

Other Models

other models

- matrices (as we have seen)
- formal power series (as we will see)
- tropical (min-plus) semiring $(N_\infty, \min, +, \infty, 0, {}^*)$ forms Kleene algebra if $n^* = 0$ for all $n \in N_\infty$

tropical semirings

- applications in graph algorithms, combinatorial optimisation, internet routing
- this would require another lecture series...
- max-plus semiring cannot be expanded to Kleene algebra

we have implemented all these models in Isabelle

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
- 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 (without assignment):

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

Quantales

definition

- quantale is structure $(Q, \leq, \cdot)$ where
  - $(Q, \leq)$ is complete lattice
  - $(Q, \cdot)$ is semigroup
  - $x \cdot \sum_{i \in I} y_i = \sum_{i \in I}(x \cdot y_i)$ and $(\sum_{i \in I} x_i) \cdot y = \sum_{i \in I}(x_i \cdot y)$
- quantale is unital if its semigroup reduct is a monoid

properties

- every (unital) quantale is a KAT with $x^* = \sum_{i \geq 0} x^i$
- least fixpoints of continuous functions exist and can be iterated

Quantales

definition

a Galois connection between posets $(A, \leq_A)$ and $(B, \leq_B)$ is a pair of functions $f : A \to B$ (lower adjoint), $g : B \to A$ (upper adjoint) such that

$$f\ x \leq_B y \Leftrightarrow x \leq_A g\ y$$

theorem

every continuous function between quantales has an upper adjoint

example

- $\lambda y.x \cdot y$ is continuous; it has upper adjoint $\lambda y.x \to y$
- $\lambda y.y \cdot x$ is continuous; it has upper adjoint $\lambda y.y \leftarrow x$
- KA with both of these residuals is called action algebra

adjunctions are studied in detail in Roy's lectures

Uniform Model Construction

observation

- many interesting models are (power set) lifting of simple structure
  - $X \cdot Y = \{x \cdot y \mid x \in X \wedge y \in Y\}$
  - $R \circ S = \{(a, b) \mid \exists c.\ (a, c) \in R \wedge (c, b) \in S\}$

convolution

for partial semigroup S, quantale Q and $f, g : S \to Q$,

$$(f \otimes g)\ x = \sum_{x = y \cdot z} f\ y \odot g\ z$$

theorem

- for $(S, \cdot)$ partial SG and $(Q, \leq, \odot)$ quantale, $(Q^S, \leq, \otimes)$ is quantale
- infs/sups lift pointwise, $\odot$ by convolution

Uniform Model Construction

languages

$(\Sigma^*, \cdot)$ and $(\mathbb{B}, \leq, \sqcap)$ lift to language quantale with language product

$$(X \cdot Y)\ w = \sum_{w = u \cdot v} X\ u \sqcap Y\ v$$

relations

partial SG $(S^2, \cdot)$ with

$$(a, b) \cdot (c, d) = \begin{cases} (a, d) & \text{if } b = c \\ \perp & \text{otherwise} \end{cases}$$

lifts to relational quantale with relational composition

$$(R \cdot S)\ (a, b) = \sum_{(a,b) = (a,c) \cdot (c,b)} R\ (a, c) \sqcap S\ (c, b)$$

Uniform Model Construction

theorem

for alphabet Σ and Kleene algebra K the function space K^{Σ^*} forms a Kleene algebra

remarks

- functions $f : \Sigma^* \to K$ are called (formal/rational) power series in formal language theory
- hence power series into KAs form KAs
- the previous slides generalise this construction

Literature

- on KA
  - Conway, *Regular Algebras and Finite Machines*
  - Kozen, *A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events*
- on KAT
  - Kozen, *Kleene Algebra with Tests*
  - Kozen, *On Hoare Logic and Kleene Algebra with Tests*
- misc
  - Rosenthal, *Quantales and Their Applications*
  - Gondran, Minoux, *Graphs, Dioids and Semirings*
  - Berstel Reutenauer, *Rational Series and Their Languages*
  - Ésik, Kuich, *Modern Automata Theory*
  - Dongol, Hayes, Struth, *Convolution, Separation and Concurrency*
  - Armstrong, Struth, Weber, *Programming and Automating Mathematics in the Tarski-Kleene Hierarchy*

Exercises

1. prove the following facts in Kleene algebra
   (a) $(x + y)^* = x^*(yx^*)^*$,
   (b) $xy \leq yz \Rightarrow x^*y \leq yz^*$,
   (c) $x^* = (x^{n+1})^* \sum_{i=0}^{n} x^i$ for all $n \in \mathbb{N}$.
2. show that for every regular expression t there is an automaton $[i, M, f]$ such that $t = i^T M^* f$
3. prove that the following expressions are equivalent in KAT.
   (a) $px \leq xq$,
   (b) $px\neg q = 0$,
   (c) $px = pxq$.
4. show that the star unfold and induction laws of KA hold in every quantale.
5. prove that every continuous function between complete lattices has an upper adjoint.

A Brief Introduction to Isabelle

– Lecture II –

What is Isabelle/HOL?

Isabelle is

- a generic theorem proving environment
  - a formal specification language for mathematical theories
  - an interactive theorem prover based on a logical calculus
- developed at the universities of Cambridge, München, Paris-Sud
- about 25 years old
- used by computer scientists and mathematicans world wide

Isabelle is

- a joy because it sometimes makes mathematics easy
- a pain because it sometimes makes mathematics hard

What is Isabelle/HOL?

specific characteristics

- Isabelle is an LCF-style theorem prover
- written in the functional programming language ML
- it has a small logical core and is therefore trustworthy
- it has also stood the test of time
- the user owns the means of production
- Isabelle assists users in formalising proofs
- but aims at high level of proof automation

What is Isabelle/HOL?

HOL

- Isabelle offers different logics for theorem proving
- the most popular one is Isabelle/HOL
- it is based on classical typed higher-order logic
- it supports reasoning with sets, inductive sets, recursive functions

almost everything you can write as a mathematician
you can write in Isabelle

What is Isabelle/HOL?

. . . almost everything:

- partially defined objects can be difficult to implement
  - partial functions, matrices, categories, allegories. . .
- objects that are not recursively defined as well
  - graphs, automata, networks, . . .

What is Isabelle/HOL?

workflow

- two user interfaces
  - Isabelle jEdit
  - Proof General (probably outdated)
- four modes of proof
  - interactive with natural deduction rules
  - automated with built-in provers, simplifiers, tactics
  - automated with external first-order theorem provers: sledgehammer
  - interactive with proof-scripting language Isar
- counterexample generators: nitpick/quickcheck
- type classes/locales allow building mathematical hierarchies
- large libraries of mathematical components have been implemented
- excellent documentation helps users

What is Isabelle/HOL?

users

- main applications in program verification/correctness
- increasing interest by mathematicians

alternatives

- Coq offers some advantages for programming mathematics
- Agda is popular with type theorists
- Mizar provides large mathematical libraries
- HOL is quite similar to Isabelle

This Lecture

overview

1. Isabelle's natural deduction system
2. programming and proving with numbers and lists
3. formalising mathematical structures and hierarchies
4. formalising KA and KAT

—Demo—

Some Isabelle Classes

```
class dioid = semiring +
  assumes add_idem: "x + x = x"
```

```
subclass (in dioid) join_semilattice
  by unfold_locales (auto simp add: add.commute add.left_commute)
```

- defines dioid as idempotent semiring
- shows that every dioid is a join semilattice
- proof obligations are dictated by Isabelle

Some Isabelle Classes

```
lemma star_cosim: z · x ≤ y · z ⟶ z · x* ≤ y* · z
proof
  assume z · x ≤ y · z
  hence y* · z · x ≤ y* · y · z
    by (metis mult_isol mult_assoc)
  also have ... ≤ y* · z
    by (metis mult_isor star_1r)
  finally have z + y* · z · x ≤ y* · z
    by (metis add_lub_var mult_1_left mult_isor star_ref)
  thus ?thesis
    by (metis star_inductr)
qed
```

- textbook style proof obtained with Isar
- individual lines verified with sledgehammer
- sledgehammer proofs reconstructed with metis

Some Isabelle Classes

```
interpretation rel_kleene_algebra: kleene_algebra
  (op ∪) (op O) Id {} (op ⊆) (op ⊂) rtrancl
  ⟨ proof ⟩

class dioid_tests = dioid + comp_op +
  assumes test_one: "n n 1 = 1"
  and test_mult: "n n (n n x · n n y) = n n y · n n x"
  and test_mult_comp: "n x · n n x = 0"
  and test_de_morgan: "n x + n y = n (n n x · n n y)"

abbreviation test_operator :: "'a ⇒ 'a" ("t_" [100] 101)
  where "t x ≡ n (n x")

class kat = kleene_algebra + dioid_tests
```

Conclusion

- we made the first steps with Isabelle/HOL
- sometimes we needed to program mathematical objects
- sometimes the automated provers were surprisingly good
- sometimes we struggled to prove simple facts

Literature

- papers
  - Isabelle documentation
  - Armstrong, Struth, Weber, *Programming and Automating Mathematics in the Tarski-Kleene Hierarchy*
  - Armstrong, Gomes, Struth, *Algebras for Program Correctness in Isabelle/HOL*
  - Armstrong, Gomes, Struth, *Algebraic Principles for Rely-Guarantee Style Concurrency Verification Tools*
- mathematical components
  - Armstrong, Struth, Weber, *Kleene Algebra*, AFP
  - Armstrong, Gomes, Struth, *Kleene Algebra with Tests and Demonic Refinement Algebra*, AFP

Exercises

use Isabelle for the following tasks

1. prove or refute the following facts by using natural deduction
   (a) $(p \wedge q) \rightarrow r \dashv\vdash p \rightarrow (q \rightarrow r)$,
   (b) $\exists x.\forall y.P(x, y) \vdash \forall y.\exists x.P(x, y)$,
   (c) $\forall y.\exists x.P(x, y) \vdash \exists x.\forall y.P(x, y)$.
2. prove the following facts about Kleene algebra.
   (a) $(x + y)^* = x^*(yx^*)^*$,
   (b) $xy \leq yz \Rightarrow x^*y \leq yz^*$,
   (c) $x^* = (x^{n+1})^* \sum_{i=0}^{n} x^i$.
3. prove that the following expressions are equivalent in KAT.
   (a) $px \leq xq$,
   (b) $px\neg q = 0$,
   (c) $px = pxq$.

Construction/Verification Tools for Sequential Programs

– Lecture III –

Verification Tool 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 Tool 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 extended by store
5. use Isabelle polymorphism to integrate arbitrary data domains
6. write tactic to transform KAT/Hore logic into verification conditions
7. verify programs

tool correct by construction

Hoare Triples in KAT

validity of Hoare triple

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

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

$$
\begin{array}{rcl}
 & & \vdash \{p\}\ \mathsf{skip}\ \{p\} \\
p \le p' \wedge q' \le q \wedge \vdash \{p'\}\ x\ \{q'\} & \Rightarrow & \vdash \{p\}\ x\ \{q\} \\
\vdash \{p\}\ x\ \{r\} \wedge \vdash \{r\}\ y\ \{q\} & \Rightarrow & \vdash \{p\}\ x; y\ \{q\} \\
\vdash \{pb\}\ x\ \{q\} \wedge \vdash \{p\neg b\}\ y\ \{q\} & \Rightarrow & \vdash \{p\}\ \mathbf{if}\ b\ \mathbf{then}\ x\ \mathbf{else}\ y\ \mathbf{fi}\ \{q\} \\
\vdash \{pb\}\ x\ \{p\} & \Rightarrow & \vdash \{p\}\ \mathbf{while}\ b\ \mathbf{do}\ x\ \mathbf{od}\ \{\neg bp\}
\end{array}
$$

Store and Assignments

store in Isabelle

- store S implemented as record of program variables
- generic for any type of data (KAT/relation KAT polymorphic)
- each variable has retrieve/update function
- state s is element of store

assignment

$$('x := e) \ = \ \{(s, x\_update\ s\ e) \mid s \in S\}$$

theorem

all inference rules of HL derivable in relation KAT with store

$$P \subseteq Q[e/'x] \ \Rightarrow \ \vdash \{P\}\ ('x := e)\ \{Q\}$$

Verification Condition Generation

Hoare logic

- one structural rule per program construct
- programmed as hoare tactic in Isabelle
- usually 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 Tool

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 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

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

Simple Refinement Calculus

theorem

Morgan's refinement laws 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

Simple Refinement Calculus

theorem

assignment laws derivable in relation rKAT

$$P \subseteq Q[e/'x] \Rightarrow [P,Q] \sqsubseteq ('x := e)$$
$$Q' \subseteq Q[e/'x] \Rightarrow [P,Q] \sqsubseteq [P,Q']; ('x := e)$$
$$P' \subseteq P[e/'x] \Rightarrow [P,Q] \sqsubseteq ('x := e); [P'; Q]$$

— demo —

Verification/Refinement Tool

conclusion

- refinement tool built in one afternoon from verification tool
- other refinement rules easily derivable
- more construction/verification examples in libraries

Programs with Recursion

definition

a test quantale is a quantale that is also a test semiring

theorem

for isotone function $f : Q \to Q$ on test quantale Q, the inference rules of PHL and the following recursion rule are derivable

$$(\forall x \in Q.\ \vdash \{p\}\ x\ \{q\} \Rightarrow \vdash \{p\}\ f\ x\ \{q\}) \Rightarrow \vdash \{p\}\ \mu f\ \{q\}$$

proof

use Knaster-Tarski theorem

Programs with Recursion

programming syntax

$$\textbf{letrec } f \textbf{ in } S \textbf{ end} = \mu(\lambda f.\ S)$$

example

```
letrec Fac in
   if ‘x = 0 then
      ‘y := 1
   else
      ‘x := ‘x - 1;
      Fac;
      ‘x := ‘x + 1;
      ‘y := ‘y·‘x
   fi
end
```

Programs with Recursion

remarks

- additional consequence rule (Kleymann's rule) is needed for relative completeness
- this allows verification of parameterless recursive programs
- mutual recursion is currently not implemented
- quantales also support powerful rules for program construction
  - refinement calculi with recursion laws can be derived
  - transformation techniques (e.g. fixpoint fusion) are available
  - transition from imperative to functional programming requires types

— demo —

Predicate Transformer Algebras

predicate transformers

- associate state transformer $f_R : A \to 2^B$ with relation $R \subseteq A \times B$

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

- lift to predicate transformer $|R] : 2^B \to 2^A$

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

- lift to $\langle R|P = \bigcup\{f_R\ x \mid x \in P\}$ of type $2^A \to 2^B$

intuition

- $\langle R|P$ / $|R\rangle P$ model image/preimage of X under P
- boxes are De Morgan duals of diamonds: $|R]P = \neg|R\rangle\neg P$

Predicate Transformer Algebras

modal KA

- KAT plus functions $d : K \to B$, $r : K \to B$ satisfying 3 axioms each
- define modal operators/predicate transformers

$$\langle x|p = r(px) \qquad |x\rangle p = d(xp) \qquad [x|p = \neg\langle x|\neg p \qquad |x]p = \neg|x\rangle\neg p$$

- define Hoare triple

$$\vdash \{p\}\ x\ \{q\} \ \Leftrightarrow\ \langle x|p \leq q \ \Leftrightarrow\ p \leq |x]q$$

- assignment statement
  - as before $\quad ('x := e) = \lambda x.\ (x\_update\ s\ e)$
  - lifted to pt as $\quad |('x := e)]$
- HL/refinement laws once more derivable

Domain Semirings

definition

domain semiring is dioid S expanded by $d : S \to S$ that satisfies

$$x + d(x)x = d(x)x \qquad d(xy) = d(xd(y)) \qquad d(x+y) = d(x) + d(y)$$
$$d(x) + 1 = 1 \qquad d(0) = 0$$

theorem

domain algebra $(d(S), +, \cdot, 0, 1)$ is bounded DL

modalities

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

yields distributive lattice with operators (range r opposite of d)

Antidomain Semirings

definition

antidomain semiring is semiring S expanded by $' : S \to S$ that satisfies

$$x'x = 0 \qquad (xy)' \leq (xy'')' \qquad x'' + x' = 1$$

domain algebra

- domain definable as $d(x) = x''$ (Boolean complement)
- subalgebra $d(S)$ is maximal BA in $[0, 1]$

consequence

general way of defining modal logics

- we have $|x\rangle 0 = 0$ and $|x\rangle(p + q) = |x\rangle p + |x\rangle q$
- this yields BAOs [JónssonTarski]
- every BAO can be obtained that way

Modalities, Symmetries, Dualities

demodalisation

$$|x\rangle p \leq q \;\Leftrightarrow\; \neg q x p \leq 0 \qquad \langle x| p \leq q \;\Leftrightarrow\; p x \neg q \leq 0$$

dualities

- de Morgan: $|x]p = \neg|x\rangle\neg p \qquad [x|p = \neg\langle x|\neg p$
- opposition: $\langle x|,\ [x| \Leftrightarrow |x\rangle,\ |x]$

symmetries

- conjugation: $(|x\rangle p)q = 0 \;\Leftrightarrow\; p(\langle x|q) = 0$
- Galois connection: $|x\rangle p \leq q \;\Leftrightarrow\; p \leq [x|q$

Models

trace model

$$p_0 a_0 p_1 a_1 p_2 \dots p_{n-2} a_{n-1} p_{n-1}, \quad p_i \in P, a_i \in A$$

theorem

- powerset algebra $2^{(P,A)^*}$ forms (full trace) MKA where

$$|T\rangle Q = \{p \mid p.\sigma.q \in T \text{ and } q \in Q\}$$

- subalgebras form trace MKAs

other models

- path, language, relation MKAs can again be obtained by forgetting
- in relation MKAs, sets are subidentities

MKA and Hoare Logic

fundamental equation of Hoare logic

syntax and semantics related by Galois connection

$$\vdash \{p\}x\{q\} \;\Leftrightarrow\; \langle x|p \le q \;\Leftrightarrow\; p \le |x]q \;\Leftrightarrow\; p \le wlp(x,q)$$

consequence

- $wlp(x,q) = \sum\{p \mid \{p\}x\{q\}\}$ in quantale
- hence $wlp(x,p)$ is indeed weakest liberal precondition
- wlp/predicate transformer semantics is simply calculus of MKA (e.g. $\vdash \{wlp(x,q)\}x\{q\}$ is cancellation law of Galois connection)

theorem

inference rules of PHL derivable in MKA

MKA and Hoare Logic

consequence

- simple algebraic approach to predicate transformers
- strong relationship to PDL/BAO
- simple equational soundness/completeness proofs for Hoare logic
- valid in variety of models
- easy to implement (in Isabelle)

Decidability of PHL

Hoare formulas

quasi-identities in modal Kleene algebra

$$\langle x_1|p_1 \leq q_1, \ldots, \langle x_n|p_n \leq q_n \Rightarrow \langle a_0|p_0 \leq q_0$$

PSPACE decision procedure

1. demodalisation: rewrite as equivalent quasi-identity in KAT

$$p_1 x_1 \neg q_1 \leq 0, \ldots, p_n x_n \neg q_n \leq 0 \Rightarrow p_0 x_0 \neg q_0 \leq 0$$

2. hypothesis elimination: reduce to equivalent identity $s' \leq t'$
3. apply decision procedure for equational theory of KAT

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 implementations
  - Guttmann, Struth, Weber, *Automating Algebraic Methods in Isabelle*
  - Armstrong, Gomes, Struth, *Algebras for Program Correctness in Isabelle/HOL*
  - Armstrong, Gomes, Struth, *Lightweight Program Construction and Verification Tools in Isabelle/HOL*
  - Armstrong, Gomes, Struth, *Building Program Construction and Verification Tools from Algebraic Principles*

Exercises

1. derive the inference rules of PHL in KAT
2. derive Morgan's refinement laws in rKAT
3. derive the recursion rule in a test quantale
4. derive the inference rules of PHL in MKA

Extensions

– Lecture IV –

Local Reasoning with Separation Logic

MKA	predicate transformers over assertion quantale	predicate tansformers over store/heap
control flow	abstract data flow	concrete data flow
propositional Hoare logic with frame rule	wlp-based verification conditions	Hoare logic with frame rule

Algebra of Separation Logic

assertion algebra

- so far: assertions form BA in KAT/quantale
- now: include separating conjunction

algebra of programs

- so far: relational semantics for KAT
- now: predicate transformer semantics/MKA much simpler

what is separating conjunction?

Assertion Quantale of Separation Logic

separating conjunction

for resource semigroup $(S, \cdot)$ and quantale $\mathbb{B}$

$$p * q = \{y \cdot z \in S \mid y \in p \wedge z \in q\} \qquad (p * q)\ x = \sum_{x = y \cdot z} p\ y \sqcap q\ z$$

assertion algebra

$(\mathbb{B}^S, \leq, *)$ forms commutative quantale

instances

- multisets form free commutative SGs
- heaplets $h : A \to B$ form partial commutative SGs with

$$h_1 \cdot h_2 = \begin{cases} h_1 \cup h_2 & \text{if } d(h_1) \cap d(h_2) = \emptyset \\ \bot & \text{otherwise} \end{cases}$$

Assertion Quantale of Separation Logic

extended separating conjunction

for store S, heap H and quantale $\mathbb{B}$ define

$$(p * q)\ s\ h = \sum_{h=h_1 \cdot h_2} p\ s\ h_1 \sqcap q\ s\ h_2$$

extended lifting

let S be set, H partial commutative SG and Q commutative quantale, then $Q^{S \times H}$ is commutative quantale

Predicate Transformers over Assertions Quantale

definition

pt $|x]$ is local if $(|x]p) * q \leq |x](p * q)$ (iff $|x] * |1] \leq |x]$)

theorem

the following function spaces form MKAs/distributive quantales

- conjunctive pts over assertion quantales
- local conjunctive pts over assertion quantales
  - multiplication of pts is function composition
  - $*$ is not lifted

theorem

- each PHL rule equivalent to some MKA/quantale identity
- frame rule derivable iff pts local

$$\vdash \{p\}\ x\ \{q\} \Rightarrow \vdash \{p * r\}\ x\ \{q * r\}$$

Verification/Refinement Tool

- store/heap model from Isabelle libraries
- assignment/mutation rules derivable in model
- specification statement $[p, q] = \sum\{|x] \mid p \leq |x]q\}$ in pt quantale
- specific refinement law $[p * r, q * r] \sqsubseteq [p, q]$ derivable
- additional refinement laws for mutation

tool tested e.g. on linked list reversal

Verification/Refinement Tool

conclusion

- algebraic foundations of separation logic
- inspired by abstract separation logic/BI quantale
- uses formal power series/modal algebra
- supports verification/refinement
- PHL as quantale fragment used for vc generation
- MKA yields dynamic separation algebra similar to PDL

Rely-Guarantee Based Concurrency Verification

rgKA	trace rgKA	trace rgKA over store
control flow with r-g interface	abstract data flow with interference	concrete data flow with interference
propositional calculus	transition trace verification conditions	r-g calculus

Concurrency Verificaton

intuition

- one central issue is interference
- another is compositionality
- interference interferes with compositionality
- traces/languages with interleaving form standard model

rely-guarantee method

- compositional Hoare logic for shared variable concurrency
- based on interleaving semantics
- relies describe effect of environment on process
- guarantees describe effect of process on environment

what is the algebra of rely-guarantee?

Rely-Guarantee Kleene Algebra

definition

bi-Kleene algebra is structure $(K, +, \cdot, ||, 0, 1, ^*, ^{(*)})$ where

- $(K, +, \cdot, 0, 1, ^*)$ is KA
- $(K, +, ||, 0, 1, ^{(*)})$ is commutative KA

models

- shuffle (regular) languages
- series-parallel rational pomset languages

Rely-Guarantee Kleene Algebra

definition

rely-guarantee KA is structure $(K, I, +, \sqcap, \cdot, \|, 0, 1,^*)$ where

- $(K, +, \sqcap)$ is distributive lattice
- $(K, +, \cdot, \|, 0, 1,^*)$ is bi-KA (no concurrent star)
- $I \subseteq K$ is set of interference constraints with axioms

$$r\|r \leq r \qquad r \leq r\|r' \qquad r\|xy = (r\|x)(r\|y) \qquad r\|x^+ \leq (r\|x)^+$$

consequences

$$1 \leq r \qquad r^* = rr = r = r\|r \qquad r\|x^+ = (r\|x)^+$$

Rely-Guarantee Calculus

Hoare triples (Tarlecki-style)

$$\vdash \{x\}\ y\ \{z\} \Leftrightarrow xy \leq z$$

Jones quintuples

$$r, g\ \vdash \{p\}\ x\ \{q\}\ \Leftrightarrow\ p(r\|x) \leq q \wedge x \leq g$$

theorem

rules of propositional rg-calculus derivable in rgKA, e.g.

$$\frac{r_1, g_1\ \vdash \{p_1\}\ x\ \{q_1\}\quad g_1 \leq r_2\quad r_2, g_2\ \vdash \{p_2\}\ y\ \{q_2\}\quad g_2 \leq r_1}{r_1 \sqcap r_2, g_1||g_2\ \vdash \{p_1 \sqcap p_2\}\ x||y\ \{q_1 \sqcap q_2\}}$$

Breaking Compositionality

basic rgKA too compositional

- $x = y \Rightarrow x \| z = y \| z$ fails if x and y have different atomicity
- map $\pi : K \to K$ prevents that $\pi(x) = \pi(y) \Rightarrow \pi(x||z) = \pi(y||z)$
- redefine Jones quintuples

$$r, g \ \vdash \{p\}\ x\ \{q\} \Leftrightarrow \pi(p(r||x)) \leq \pi(q) \wedge x \leq q$$

theorem

rules of propositional rg-calculus still derivable

Finite Trace Model

transition/Aczel traces

- traces in $\mathrm{Tra}(\Sigma)$ built from pairs of letters from Σ

$$(\sigma_1, \sigma'_1)(\sigma_2, \sigma'_2)(\sigma_3, \sigma'_3) \ldots (\sigma_n, \sigma'_n)$$

- process controls (σ_i, σ'_i), environment/rely controls $\sigma'_i)(\sigma_{i+1}$
- trace consistent if $\sigma'_i = \sigma_{i+1}$

interleaving breaks consistency

$(a, b)(b, c)$	$(a, b)(c, c)$
$(a, b)(b, c)(b, c)$	$(a, b)(b, c)(c, c)$

Finite Trace Model

π in trace model

- let $\mathsf{Trc}(\Sigma)$ be set of consistent Σ-traces
- then $\pi = \lambda X.\ X \cap \mathsf{Trc}(\Sigma)$

interference constraints

- r-gs represented as binary relations R
- lifted to singleton language $\langle R \rangle$ in obvious way
- set of interference constraints $I = \{r \mid \exists R.\ r = \langle R \rangle^*\}$

theorem

$(\mathsf{Trc}(\Sigma), I, \cup, \cdot, ||, \emptyset, \{\varepsilon\},^*, \pi)$ forms rgKA

Verification Tool

additional concepts for programs/assertions

- tests $P \subseteq \langle id \rangle$
- $P \cdot Q = P \cap Q$ requires closure conditions (stuttering, mumbling)

tool tested e.g. on findp example

Verification of FINDP

$f_A := \text{len}(\text{array});$

$f_B := \text{len}(\text{array});$

$$\left(\begin{array}{l} i_A = 0 \\ \text{while } i_A < f_A \wedge i_A < f_B \ \{ \\ \qquad \text{if } P(\text{array}[i_A]) \ \{ \\ \qquad\qquad f_A := i_A \\ \qquad \} \text{ else } \{ \\ \qquad\qquad i_A := i_A + 2 \\ \qquad \} \\ \} \end{array} \middle\| \begin{array}{l} i_B = 1 \\ \text{while } i_B < f_A \wedge i_B < f_B \ \{ \\ \qquad \text{if } P(\text{array}[i_B]) \ \{ \\ \qquad\qquad f_B := i_B \\ \qquad \} \text{ else } \{ \\ \qquad\qquad i_B := i_B + 2 \\ \qquad \} \\ \} \end{array}\right);$$

$f = \min(f_A, f_B)$

Verification of FINDP

postcondition

either we find least element or terminate at end of array

$$\pi(FINDP) \leq \pi(end(leastP(f)) + end(f = |array|))$$

relies/guarantees

- A guarantees that it only changes f_A in B
- B guarantees that it only changes f_B in A
- these serve respectively as relies for the other process

verification

- very tedious
- needs tuning/optimisation

Verification Tool

conclusion

- algebraic principles of rg-reasoning based on bi-KA
- transition trace model obtained from language KA
- performance needs to be optimised
- current work on infinite traces and cyclic rg-conditions

Literature

- separation logic
  - Hoare et al., *On Locality and the Exchange Law for Concurrent Processes*
  - Dongol, Hayes, Struth, *Convolution, Separation and Concurrency*
  - Dongol, Gomes, Struth, *A Program Construction and Verification Tool for Separation Logic*
  - Gomes, Struth, *Dynamic Separation Algebra*
- rely-guarantee
  - Hoare, Möller, Struth, Wehrman, *Concurrent Kleene Algebra and its Foundations*
  - Armstrong, Gomes, Struth, *Algebraic Principles for Rely-Guarantee Style Concurrency Verification Tools*
  - Laurence, Struth, *Completeness Theorems for Bi-Kleene Algebras and Series-Parallel Rational Pomset Languages*

General Conclusion

- principled approach to program correctness tools in Isabelle
  - use algebra at control flow layer
  - link with relation/trace/pt semantics and store
  - derive 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

Formalisation in Isabelle

Archive of Formal Proofs

- http://afp.sourceforge.net
- own contributions
  - Kleene algebra (> 100p)
  - relation algebra (> 30p)
  - Kleene algebra with tests and demonic refinement algebra (> 50p)
  - regular algebras (> 50p)
  - modal Kleene algebra (Summer 2015)
  - quantales (Summer 2015)

tool web sites

- staffwww.dcs.shef.ac.uk/people/V.Gomes/refinement
- github.com/vborgesfer/sep-logic
- github.com/Alasdair/FM2014