On Automated Program Construction and Verification

Rudolf Berghammer

Christian-Albrechts-University of Kiel
Germany

Georg Struth

University of Sheffield
United Kingdom

War on Error

two approaches: (program correctness)

- synthesis/construction (Dijkstra/Gries)
- verification (Floyd/Hoare)

imperative programs:

- pre/postconditions, invariants, termination measures

challenge: from programming science to program engineering

- modelling languages: simple, expressive
- tool support: invisible, automatic

Our Approach

relational modelling languages (à la Alloy)	domain specific algebras (relation algebra . . . Kleene algebra) + ATP systems/model search (Prover9, Mace4, RelView)

aim: automated support for Dijkstra/Gries method

- intuitive calculational reasoning
- modelling game of proof/refutation
- algebras/proofs at dark side of interface

Three Case Studies

automated correctness proofs of classical algorithms

1. Warshall's algorithm (synthesis)
2. reachability in digraphs (verification)
3. Szpilrajn's algorithm (synthesis/verification)

Relational Modelling Language

binary relation: $x \subseteq A \times A$

operations:

$$x + y \qquad x * y \qquad x' \qquad 0 \qquad U$$

$$x; y = \{(a, b) : \exists c.(a, c) \in x \wedge (c, b) \in y\}$$

$$x^{\wedge} = \{(b, a) : (a, b) \in x\}$$

$$1 = \{(a, a) : a \in A\}$$

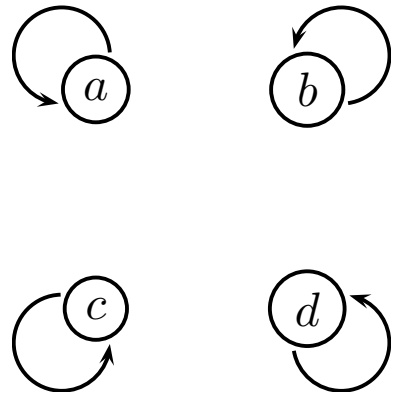
$$\mathsf{rtc}(x) = \sum_{i \geq 0} x^i \qquad \mathsf{tc}(x) = \sum_{i \geq 1} x^i$$

$$d(x) = \{a \in A : \exists b \in A.(a, b) \in x\}$$

Relations, Matrices, Graphs

finite relations: matrices/graphs

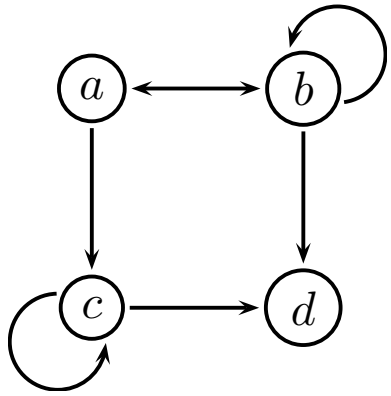
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Relations, Matrices, Graphs

finite relations: matrices/graphs

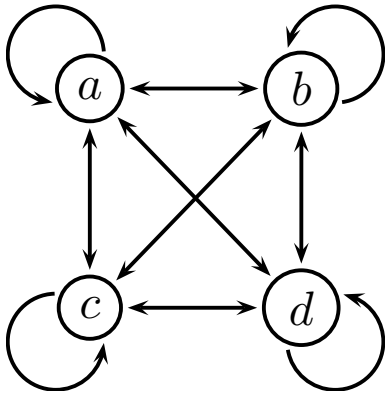
$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



Relations, Matrices, Graphs

finite relations: matrices/graphs

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



Relations, Matrices, Graphs

relational operations reflected in matrix linear algebra

sets modelled as

- vectors (row constant matrices) or
- subidentities

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

Domain-Specific Algebras

relation algebra: $(R, +, *, ', 0, U, ;, 1, \hat{})$

```
x+y=y+x  &  x+(y+z)=(x+y)+z  &  x=(x'+y')'+(x'+y)'.
x;(y;z)=(x;y);z  &  x;1=x  &  (x+y);z=(x;z)+(y;z).
(x^)^=x  &  (x+y)^=x^+y^  &  (x;y)^=y^;x^  &  x^;(x;y)'+y'=y'.
```

(reflexive) transitive closure: (à la Ng/Tarski)

```
1+x;rtc(x)=rtc(x)  &  z+x;y<=y -> rtc(x);z<=y.
1+rtc(x);x=rtc(x)  &  z+y;x<=y -> z;rtc(x)<=y.
tc(x)=x;rtc(x).
```

remark: executable code for Prover9/Mace4

Domain-Specific Algebras

idempotent semiring: $(S, +, ;, 0, 1)$

```
x+y=y+x  &  x+(y+z)=(x+y)+z  &  x+0=x  &  x+x=x.
x;(y;z)=(x;y);z  &  x;1=x  &  1;x=x  &  x;0=0  &  0;x=0.
x;(y+z)=x;y+x;z  &  (x+y);z=x;z+y;z.
x<=y <-> x+y=y.
```

Kleene algebra: idempotent semiring + rtc-axioms

sets and points:

```
a(x);x=0  &  a(x;a(a(y)))=a(x;y)  &  a(a(x))+a(x)=1 & d(x)=a(a(x)).
set(x) <-> d(x)=x.
x<=U.
rctangle(x) <-> x;(U;x)=x.
wpoint(x) <-> set(x) & rctangle(x).
```

Synthesis vs Verification

proof obligations: (simple while-programs)

1. initialisation establishes invariant when precondition is true
2. loop executions preserve invariant when guard is true
3. invariant establishes postcondition when guard is false

synthesis: "program and correctness proof developed hand-in-hand"

- develop invariant as modification of postcondition
- incrementally establish proof obligations, derive guard/assignments

verification:

- add assertions (pre/postcondition, invariant) to code
- generate/analyse proof obligations automatically

Warshall's Algorithm: Initial Specification

spec: given finite binary relation x, find program with relational variable y that stores transitive closure of x after execution

goal: instantiate template

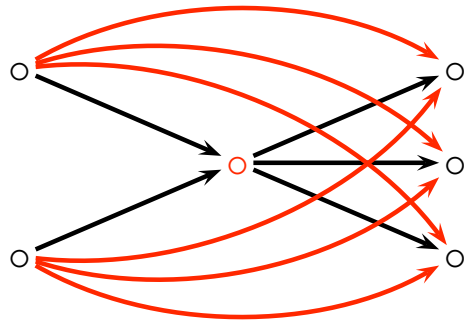
```
... y:=x ...
while ... do
  ... y:=? ... od
```

pre/postcondition: (evident from spec)

```
pre(x) <-> x=x.
post(x,y) <-> y=tc(x).
```

task: use proof obligations to synthesise initialisation, guard, body

Developing the Invariant



invariant: `inv(x,y,v) <-> (set(v) -> y=rtc(x;v);x).`

Initialisation and Guard

initialisation: $v := 0$

guard: $v \neq d(x)$

justification: in KA with domain by Prover9

```
pre(x) -> inv(x,x,0).                  %no time
inv(x,y,v) & v=d(x) -> post(x,y).      %no time
```

Termination and Synthesis of Loop

task: use preservation of invariant to find assignments

- $v := v + p$ (increment set v by point p)
- $y := y + f(y, p)$ (increment y by tc of y with p)

algorithm:

```
y,v:=x,0
while v!=d(x) do
  p:=point(v')            %choose fresh point from compl of v
  y,v:=y+f(y,p),v+p od
```

Termination and Synthesis of Loop

next: determine $y + f(y, p)$

idea: y before/after assignment

- $y = \mathsf{rtc}(x; v); x$
- $y = \mathsf{rtc}(x; (v + p)); x = \mathsf{rtc}(x; v + x; p); x \stackrel{?}{=} y + f(x, y, p)$

question: can we refine rtc of sum into sum of rtcs?

Termination and Synthesis of Loop

refinement law: for KA with maximal element and rectangle y

$$\mathsf{rtc}(x+y) = \mathsf{rtc}(x) + \mathsf{rtc}(x); y; \mathsf{rtc}(x)$$

consequence: for point p and $y = \mathsf{rtc}(x; v); x$

$$\mathsf{rtc}(x; (v+p)); x = y + y; p; y$$

therefore: `wpoint(w) & inv(x,y,v) & y!=d(x) -> inv(x,y+y;(w;y),v+w).`

Correctness of Warshall's Algorithm

theorem: Warshall's algorithm is (partially) correct:

```
y,v:=x,0
while v!=d(x) do
  p:=point(v')
  y,v:=y+y;p;y,v+p od
```

discussion: correctness by construction

- fully automated proof with Prover9
- uses KA axioms + 2 independent hypotheses
- Mace4 indicates when these are needed
- finding them can be learned
- reasoning essentially inductive (beyond FOL)

Outlook: Verification of Reachability Algorithm

task: compute set w of vertices that are reachable in digraph y from set of vertices v

algorithm: (naive)

```
{pre(y,v) <-> true}
w:=v
while -(y^;w<=w) do
  {inv(y,v,w) <-> v<=w & w<=rtc(y^);v}
  w:=w+y^;w od
{post(y,v,w) <-> w=rtc(y^);v}
```

idea: assume imaginary tool that translates assertions and generates proof obligations

Outlook: Verification of Reachability Algorithm

assertions: (in KA, $y^\wedge$ replaced by x)

```
%pre(x,v) <-> x=x.
guard(x,v,w) <-> -(x;w<=w).
post(x,v,w) <-> w=rtc(x);v.
inv(x,v,w) <-> v<=w & w<=rtc(x);v.
```

proof obligations:

```
inv(x,v,w) & -guard(x,v,w) -> post(x,v,w).
inv(x,v,v).
inv(x,v,w) & guard(x,v,w) -> inv(x,v,w+x;w).
```

correctness proof: very quick/short with Prover9

in paper: verification of refined algorithm (no guard recomputations)

Discussion

contribution: automated program construction via algebra + ATP

theory engineering: calculational style ideal for automation

- here: relations as data structures
- in general: domain-specific algebras can model control flow
  (Hoare logic, refinement calculus, process algebras, concurrency,. . .)

perspective: lightweight formal methods with heavyweight automation

- analyse further algorithms
- combine with SMT/ITP
- integrate into program analysis tools

complete code: `www.dcs.shef.ac.uk/~georg/ka`

Shut Up and Calculate!

[David Mermin]