Kleene Algebra with Tests and the Static Analysis of Programs

Dexter Kozen

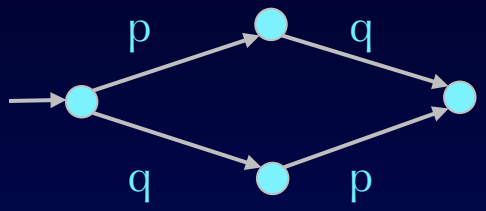
Cornell University

Kleene Algebra (KA)

is the algebra of regular expressions

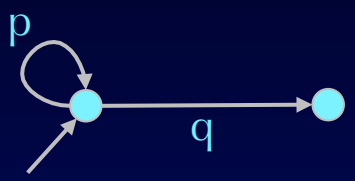
pq + qp

{pq, qp}



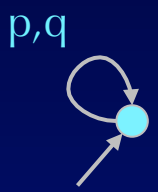
p*q

{q, pq, p^2q, p^3q, …}



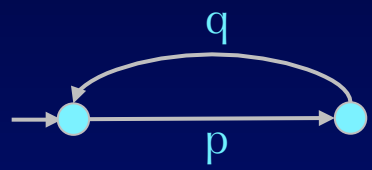
(p + q)* = (p*q)*p*

{all strings over p,q}



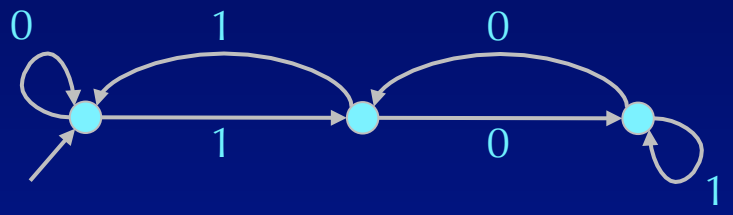
(pq)*p = p(qp)*

{p, pqp, pqpqp, …}



(0 + 1(01*0)*1)*

{multiples of 3 in binary}



Axioms of KA

- K is an idempotent semiring under $+$, $\cdot$, 0, 1

$$(p + q) + r = p + (q + r) \qquad (pq)r = p(qr)$$
$$p + q = q + p \qquad p1 = 1p = p$$
$$p + p = p \qquad p0 = 0p = 0$$
$$p + 0 = p$$

$$p(q + r) = pq + pr$$
$$(p + q)r = pr + qr$$

- p^*q = least x such that $q + px \leq x$
- qp^* = least x such that $q + xp \leq x$

$$x \leq y \overset{\text{def}}{\leftrightarrow} x + y = y$$

Models

- standard model
  - regular sets of strings over a finite alphabet
- models useful in program semantics
  - binary relations
  - traces (computation paths)
- other more esoteric interpretations
  - min,+ algebra for shortest path problems
  - convex sets used in computational geometry

Completeness and Complexity

- Deductively complete for regular sets of strings and binary relations [K 94]
- Equational theory decidable in PSPACE [Meyer +Stockmeyer 74]
- Hoare theory (Horn theory with premises p = 0) decidable in PSPACE [Cohen 93]

Matrices over a KA

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} + \begin{pmatrix} e & f \\ g & h \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} a+e & b+f \\ c+g & d+h \end{pmatrix}$$

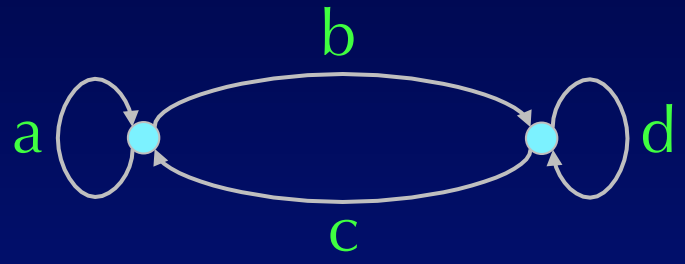
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

$$0 \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad 1 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* \stackrel{\text{def}}{=} \begin{pmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^* \\ (d+ca^*b)^*ca^* & (d+ca^*b)^* \end{pmatrix}$$

Matrices over a KA

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* \overset{\text{def}}{=} \begin{pmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^* \\ (d+ca^*b)^*ca^* & (d+ca^*b)^* \end{pmatrix}$$



Matrices over a KA

- Representation of finite automata
- Construction of regular expressions
- Solution of linear equations
- Connectivity and shortest path algorithms

Kleene Algebra with Tests (KAT)

$(K, B, +, \cdot, ^*, \bar{}, 0, 1)$

- $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra
- $(B, +, \cdot, \bar{}, 0, 1)$ is a Boolean algebra
- $(B, +, \cdot, 0, 1) \subseteq (K, +, \cdot, 0, 1)$

Completeness and Complexity

- Deductively complete for regular sets of guarded strings and relations [K+Smith 96]
- Equational theory decidable in PSPACE [K+S 96, Cohen+K+S 97]
- Hoare theory (Horn theory with premises $p = 0$) decidable in PSPACE [K+S 96]

Modeling Programs

[Fischer + Ladner 74]

if b then p else q $\equiv$ $bp + \overline{b}q$

while b do p $\equiv$ $(bp)^*\overline{b}$

Hoare Logic [C. A. R. Hoare 69]

{b}p{c}partial correctness assertion

"If b holds in the current state, and if p is executed starting in the current state, then if p halts, c will be true of the halting state."

KAT subsumes Hoare Logic

$\{b\}p\{c\}$ equivalent to $bp \leq pc$

$$\frac{\{b_1\}p_1\{c_1\},\ \{b_2\}p_2\{c_2\},\ \ldots,\ \{b_n\}p_n\{c_n\}}{\{b\}p\{c\}}$$

equivalent to the Horn formula

$$b_1p_1 \leq p_1c_1 \ \wedge\ \ldots\ \wedge\ b_np_n \leq p_nc_n \ \rightarrow\ bp \leq pc$$

Rules of Hoare Logic

$$\{b[x/e]\}\ x:=e\ \{b\}$$ assignment rule

$$\frac{\{b\}p\{c\},\ \ \{c\}q\{d\}}{\{b\}pq\{d\}}$$ composition rule

$$\frac{\{bc\}p\{d\},\ \{\overline{b}c\}q\{d\}}{\{c\}\text{if } b \text{ then } p \text{ else } q\{d\}}$$ conditional rule

$$\frac{\{bc\}p\{c\}}{\{c\}\text{while } b \text{ do } p\{\overline{b}c\}}$$ while rule

$$\frac{b' \rightarrow b,\ \{b\}p\{c\},\ c \rightarrow c'}{\{b'\}p\{c'\}}$$ weakening rule

Theorem

All Hoare rules are theorems of KAT

Completeness Theorem [K 99]

All relationally valid rules of the form

$$\frac{\{b_1\}p_1\{c_1\},\ \ldots,\ \{b_n\}p_n\{c_n\}}{\{b\}p\{c\}}$$

are derivable in KAT (not so for HL)

Counterexample

$$\frac{\{c\}\text{if } b \text{ then } p \text{ else } p\{c\}}{\{c\}p\{c\}}$$

is trivially unprovable in Hoare Logic, but

easily provable in KAT

Applications of KAT

- Compiler optimization
- Static analysis & verification
- Complexity of substructural logics
- Unique ordered BDDs
  - a special case of Myhill-Nerode theorem for automata on guarded strings

Reasoning under Assumptions

Self-evident premises about the atomic or local behavior of individual instructions and tests

correctness of global code restructuring

$$p_1 = q_1 \ \wedge \ p_2 = q_2 \ \wedge \ \ldots \ \wedge \ p_n = q_n \ \rightarrow \ p = q$$

Static Analysis

- Derivation of information about the execution state at various points in a program at compile time or load time, prior to execution
- Approaches
  - type inference
  - dataflow analysis
  - abstract interpretation
  - set constraints
- Applications
  - code optimization
  - verification

Software Model Checking with SLAM

Thomas Ball
Testing, Verification and Measurement

Sriram K. Rajamani
Software Productivity Tools

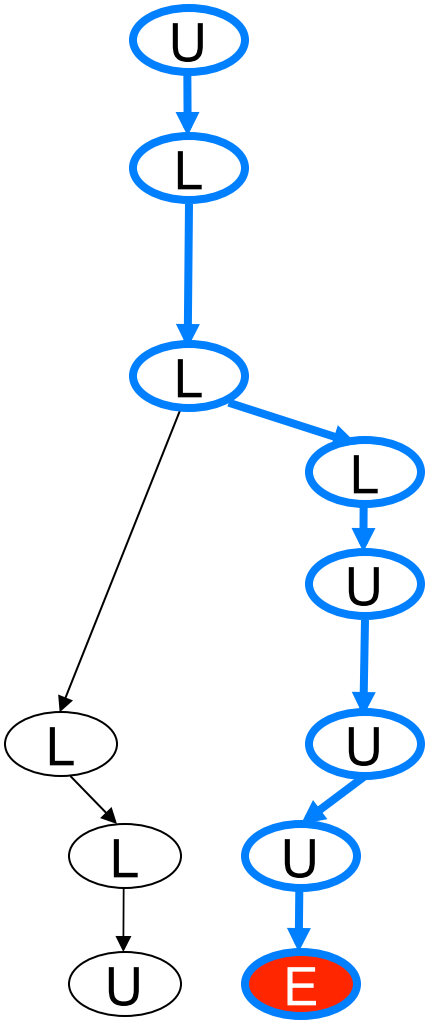
Microsoft Research

http://research.microsoft.com/slam/

Example

b : (nPacketsOld == nPackets)

Add new predicate to boolean program (c2bp)



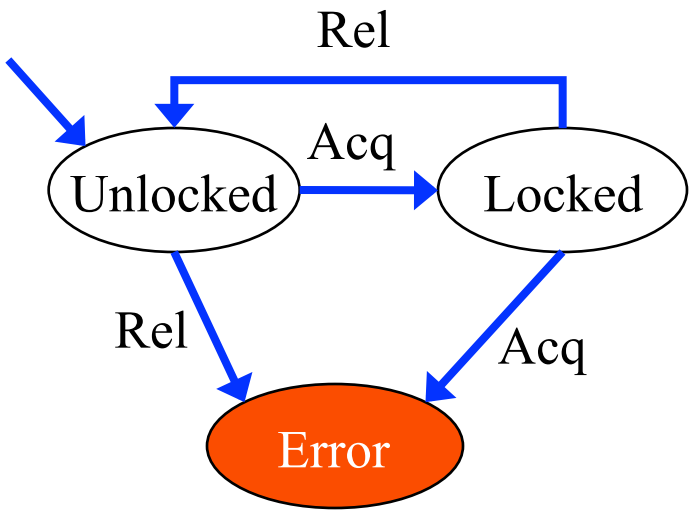

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets; b = true;

   if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++; b = b ? false : *;
   }
} while (nPackets != nPacketsOld);   !b

KeReleaseSpinLock();
```

State Machine for Locking



Locking Rule in SLIC

```
state {
  enum {Locked,Unlocked}
    s = Unlocked;
}

KeAcquireSpinLock.entry {
  if (s==Locked) abort;
  else s = Locked;
}

KeReleaseSpinLock.entry {
  if (s==Unlocked) abort;
  else s = Unlocked;
}
```

```
do {
   KeAcquireSpinLock();
   nPacketsOld = nPackets;
   if (request) {
     request = request->Next;
     KeReleaseSpinLock();
     nPackets++;
   }
 } while (nPackets != nPacketsOld);
 KeReleaseSpinLock();
```

```
do {
   kA;
   n;
   if (R) {
      u;
      kR;
      m;
   }
 } while (~B);
 kR;
```

```
do {
   kA;
   n;
   if (R) {
     u;
     kR;
     m;
   }
 } while (~B);
 kR;
```

```
do p while C
   =  p(Cp)*~C

if C then p
   = Cp + ~C
```

```
do {
   kA;
   n;
   if (R) {
     u;
     kR;
     m;
   }
 } while (~B);
 kR;

 kA n (R u kR m + ~R)
     (~B kA n (R u kR m + ~R))* B kR
```

```
do p while C
   =  p (C p)* ~C

if C then p
   = C p + ~C
```

Preconditions for safe execution

operation	precondition
`kA` (acquire)	`~A` (unlocked)
`kR` (release)	`A` (locked)

Global precondition: initially unlocked

Original program

```
~A kA n (R u kR m + ~R)
    (~B kA n (R u kR m + ~R))* B kR
```

Annotated program

```
~A ~A kA n (R u A kR m + ~R)
    (~B ~A kA n (R u A kR m + ~R))* B A kR
```

```
kA = kA A
kR = kR ~A
B m = B m ~B

n = n B

A n = n A
A u = u A
A m = m A
B u = u B
B kR = kR B
```

– acquiring the lock acquires it

– releasing the lock releases it

– if two integer variables are equal and we increment one, then they are no longer equal

– setting one variable equal to another makes them equal

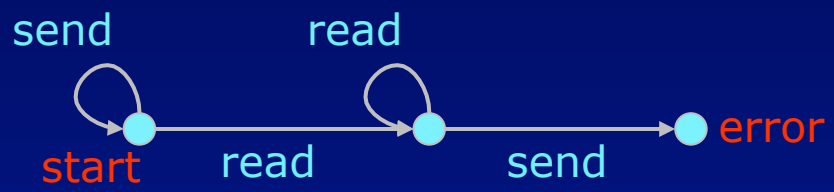
– commutativity conditions

```
kA = kA A
→ kR = kR ~A
→ B m = B m ~B
→ n = n B
→ A n = n A
→ A u = u A
→ A m = m A
→ B u = u B
→ B kR = kR B

→     ~A kA n (R u kR m + ~R)
          (~B kA n (R u kR m + ~R)) * B kR
                          =
      ~A ~A kA n (R u A kR m + ~R)
          (~B ~A kA n (R u A kR m + ~R)) * B A kR
```

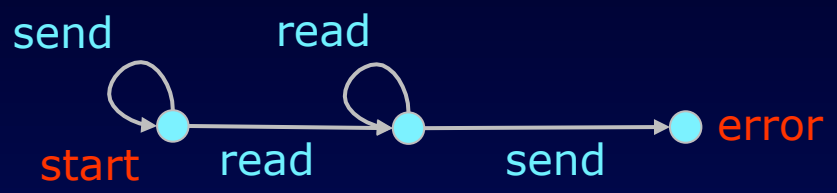
Security Automata [Schneider 98]

- General mechanism for enforcement of security policies at runtime
- Policy is specified by a finite automaton *M*
- Code is instrumented to call *M* before all critical operations (ones that could change state of *M*)
- *M* aborts the computation if the operation causes a transition to an error state
- Can handle a robust class of policies



Example

no send after read



- Code is instrumented to call *M* before each read and send instruction
- Purely a runtime mechanism

Security Automata and KAT

Idea: use KAT to propagate state information statically to eliminate unnecessary calls to *M*

- Assertion U for each state of *M*
- Premises $UV = 0$ for $U \neq V$
- Premise $Up \leq pV$ for atomic transition

$$U \xrightarrow{p} V$$

- Premise $Up \leq pU$ if p is noncritical
- Safe states $Safe_p$ for critical p – those not leading to an error state under p

Eliminating Runtime Checks

- `Pre(r)` = prefixes of `r` (defined inductively)
- `Pre(r)'` = substitute $\texttt{Safe}_{\texttt{p}}\texttt{;p}$ for `p` in `Pre(r)`

Soundness Theorem

`r` is safe with respect to M in any model if

$$\text{KAT} \vdash \mathsf{A} \rightarrow \texttt{S;Pre(r)} = \texttt{S;Pre(r)'}$$

where

- A are the premises `Up` $\leq$ `pV` and `UV` = `0`
- `S` is the start state of M

Eliminating Runtime Checks

Weak Completeness Theorem

If KAT cannot prove

$$A \rightarrow \texttt{S;Pre(r)} = \texttt{S;Pre(r)'}$$

then `r` is unsafe in some relational model

More Generally...

Let L be an upper semilattice such that all ascending chains are finite (ACC). Elements are called types or abstract values.

Associated with each atomic action p is a strict monotone map $f_p : L \rightarrow L$ called its transfer function.

Take premises $Xp \leq p\ f_p(X)$ and propagate information as before (ACC needed for *).

$Safe_p$ = dom f_p.

For details, please see

- D. Kozen. Kleene Algebra with Tests and the Static Analysis of Programs. TR 2003-1915, Computer Science, Cornell, November 2003

Other papers on applications of KAT in static analysis and compiler optimization

- L. Kot and D. Kozen. Second-Order Abstract Interpretation via Kleene Algebra. Technical Report 2004-1971, Computer Science, Cornell, December 2004
- L. Kot and D. Kozen. Kleene Algebra and Bytecode Verification. Proc. Bytecode '05, April 2005, 201-215
- D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. Proc. CL2000, July 2000, 568-582

Thank you!