Know your secret weapon. -- Herb Simon

To make an important discovery, you must study an important problem. -- Peter Medawar

I have grown more and more aware that success in science … comes not so much to the most gifted, nor the most skillful, nor the most superior strategist and tactician, but rather to the superior research strategy …

-- Jack Oliver
age of discovery

Know where you are
collection of aphorisms at

Session 20: Hints on Research Strategy

theory.lcs.mit.edu/~dnj/6898/lecture-notes.html
can mitigate effects of finite bounds
both invariant reasoning & trace analysis
can easily encode traces in the logic
no commitment to fixed topology in model itself

Illustrate aspects of Alloy modeling
Familiarity -- you can compare to Rushby
Curiosity -- I hadn’t done it before

Why this example?

Bakery algorithm
more is possible than you might have guessed

It's not always so easy

not quite...

everything's easy, but nothing's possible

Jackson on Alloy

nothing's easy, but everything's possible

Rushby on PVS

general observations
module bakery

open std/ord

sig signatures

part idle, trying, critical: set Process

ticket: Process ->? Ticket,

} State

sig Ticket

{} Process

{} sig

process in critical phase holds no ticket: hand in ticket

when you're being served
at most one process is in the critical phase:

**safety condition**
transition relation

fun Trans (s, s': State, p: Process) {
let otherTickets = s.ticket[p].otherTickets
let next = Ord[s.ticket[p].next]
let tickets = s.ticket[p]

if p in s.trying && no s'.ticket[p] then
  postcondition: p is in critical phase
else
  postcondition: p is in trying phase and all other tickets follow its ticket

precondition:
no ticket after, and holds p is in critical phase

...
other cases

fun Trans (s: State, p: Process) {
  let
    otherTickets = s.0d[Process][ticket].next
  or
    s.0d[Process][ticket] = s.0d[Process][ticket].next
    p in s.idle
  or
    s.0d[Process][ticket] = s.0d[Process][ticket].next
    p in s.critical
  or
    ...
  |
  otherTickets = s.otherTickets
  next = Ord[Ticket].next
  }


define a condition saying that a process $p$ doesn't change:

```typescript
fun NoChange (s, s': State, p: Process) {
    in s.critical = p in s'.critical
    in s.trying = p in s'.trying
    in s.idle = p in s'.idle
    s.ticket[p] = s'.ticket[p]
}
```

Frame condition
fun Init (s: State) {
    Safe (s)
}
Fun Interleaving ()

putting things together

the set State

a total order on

Instantiation imposes

use of ordering:

Interleaving () {Init (Ord[State].first)

all s: State - Ord[State].last, s': Ord[State].next[s] |

some p: Process {Trans (s,s',p)

all x: Process - p Nocchange(s,x)

Trans (s,s',p)

} some p: Process

all s: State - Ord[State].last, s': Ord[State].first

Init (Ord[State].first)

} ()

Simultaneity ()

\[
\begin{align*}
\text{fun Simultaneity} & \rightarrow \\
\text{allowing simultaneous actions} & \\
\end{align*}
\]
checking a conjecture

assert InterleavingSafe {Interleaving () =>
  all s: State | Safe (s)
} Check InterleavingSafe for 4 but 2 Process

checking a conjecture
counterexamples...
how much assurance?

check InterleavingSafety for 4 but 2 Process

analyses within bounded scope:

4 tickets? 4 states? ... not at all reasonable
2 processes ... seems reasonable

not considering all states may miss bugs

running out of tickets is a poor approximation

we've learned something about a real scenario
when is a trace long enough?

but can express conditions directly like bounded model checking

• tighter bounds possible: eg, no shortcuts
  • if none, then $k$ is a bound
  • ask for loopless trace of length $k+1$

strategy

• enough to consider only traces $\leq k$
  • if all states reached in path $\leq k$

idea: bound the diameter

max loopless = 1
max loopless = 5

$\text{diameter} = 1$

$\text{diameter} = 1$

$\text{diameter} = 1$

$\text{diameter} = 1$

$\text{diameter} = 1$
Finding the diameter

run NoRepetitionsI for 3 but 2 Process, 8 State

{ s.idle = s'.idle & s.critical = s'.critical
  s.ticket = s'.ticket
} run Equiv (s, s'.State)

fun Equiv (s, s'.State) { no dist s, s'.State | Equiv (s, s')
  Interleaving ()
} run NoRepetitionsI

fun Equiv (s, s'.State) {
can we fix the tickets in the same way?

and show not all tickets are used

so find diameter with respect to ticket ordering

ticket allocations with same process order are equivalent

a better idea

ensure enough tickets for longest trace

find diameter of machine

one idea

but know that we never run out of tickets

bound the ticket scope for fast analysis

what we want to do
defining the order
introduce process ordering as a new field

StateWithOrder
extends State {
    
        all p, p': Process | p->p'
    in
        ticket[p'] in (ord(Ticket).next)(ticket[p])
        iff
            ticket[p'] in ^ Ord[Ticket].next)(ticket[p])

        p->p' in precedes iff
            all p, p': Process
                all p' : Process
                    precedes: Process <- Process

} sig StateWithOrder extends State

{ State = StateWithOrder }

introduce process ordering as a new field

defining the order
defining state equivalence

define equivalence modulo ordering

fun EquivProcessOrder (s, s')
{
    s.precedes = s'.precedes
    s.idle = s'.idle && s.critical = s'.critical
}

define no repetition constraint

run NoRepetitionOrdered ()

define no repetition constraint

{ }

fun EquivProcessOrder (s, s')
{
    s.idle = s'.idle && s.critical = s'.critical
    s.precedes = s'.precedes
}

define equivalence modulo ordering

defining state equivalence
Finding the bounds

For 3 processes, 12 states and 7 tickets is fully general.

So now we know

```plaintext
check EnoughTicketsI for 7 but 3 Process, 12 State
assert EnoughTicketsI
check that tickets not all used
run NoRepetitionsUnderordered for 7 but 3 Process, 13 State
```

Find a diameter
Getting full coverage

Finally, we check this.

If no counterexample

Check InterleavingSafe for 7 but 3 processes. 12 State

We have a proof for 3 processes.
all scenarios for 3 processes

full analysis for bounded topology

established diameter

may miss counterexamples

analysis in small finite scope

no fixed number of processes or tickets

unbounded model of bakery

What we did
Summary of Alloy

- A simple language
- Relational first-order logic
- Signatures for structuring: Global relations
- Used for teaching in ~15 universities
- A variety of case studies
- Applications

- Tool reduces Alloy to SAT
- User provides scope, distinct from model
  Simulation & checking are instance-finding
  An effective analysis

- Description is set of constraints
- Signatures for structuring: Global relations
- Relational first-order logic
- A simple language
challenges: better analyses

• tool might show which constraints used
  false => property
  might have shown
  what when no instances are found?
  analyzing inconsistency

  • decision procedure for subset?
  data independence: scope of 3 enough?
  mitigating effects of scope

  • choosing symmetry predicates?
  eliminating irrelevant constraints?
  exploiting entailments?
  improving analyses

challenges: better analyses
challenges: applications

- Finding bugs in code
- Extracting formula from procedure
- Checking the conjecture
- Building veneers on Alloy

Example counterexample is trace:

\[ \text{pre(s)} \stackrel{\text{post(s)}}{\Rightarrow} \text{post(s, s')} \]

- Semantic web design
- Role-based access control
- On API or as macro language
challenges: case studies

reverse path forwarding, etc.

dynamic topology algorithms

check theorems of Unified Theory?
check consistency of UML metamodel
meta modelling

is it correct?
model CVS at multiple levels
source code control
thank you!
dnj@mit.edu