Lecture 4: a case study

and analysis with Alloy
declarative modeling of software
on research strategy
on research strategy

Jack Oliver

...strategist and tactician, but rather to the superior, not the most knowledgeable, nor the most gifted, nor the most skilled, ...
on research strategy

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 knowledgeable, but rather to the superior strategist and tactician. -- Jack Oliver

To make an important discovery, you must study an important problem. -- Peter Medawar
Know your secret weapon. -- Herb Simon

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

Know your secret weapon. -- Herb Simon

Know your secret weapon. -- Herb Simon

I have grown more and more aware that success in science comes not so much to the most gifted, not the most skillful, but rather to the superior strategist and tactician. -- Jack Oliver

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

Know your secret weapon. -- Herb Simon

I have grown more and more aware that success in science comes not so much to the most gifted, not the most skillful, but rather to the superior strategist and tactician. -- Jack Oliver

Know your secret weapon. -- Herb Simon

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

Know your secret weapon. -- Herb Simon
Know where you are
Session 20: Hints on Research Strategy

collection of aphorisms at

theory.lcs.mit.edu/~dmc/6898/lecture-notes.html
bakery algorithm
why this example?

bakery algorithm
bake\text{r}y algorithm

why this example?

curiosity -- I hadn’t done it before

bake\text{r}y algorithm
bakery algorithm

why this example?

- familiarity -- you can compare to Rushby
- curiosity -- I hadn't done it before

Familiarity -- you can compare to Rushby
Curiosity -- I hadn't done it before
Why this example?
why this example? › curiosity -- I hadn't done it before
› familiarity -- you can compare to Rushby
› illustrate aspects of Alloy modelling

bakep algorithm
aspects

- illustrate aspects of Alloy modelling
- familiarity -- you can compare to Rushby
- curiosity -- I hadn’t done it before

Why this example?

Bakery algorithm
no commitment to fixed topology in model itself

aspects

Illustrate aspects of Alloy modelling

Familiarity -- you can compare to Rushby

Curiosity -- I hadn't done it before

Why this example?

Bakery Algorithm
bakery algorithm

why this example?
- curiosity -- I hadn’t done it before
- familiarity -- you can compare to Rushby
- illustrate aspects of Alloy modeling
- can easily encode traces in the logic

no commitment to fixed topology in model itself

aspects
bakey algorithm

- both invariant reasoning & trace analysis
- can easily encode traces in the logic
- no commitment to fixed topology in the model itself

illustrate aspects of Alloy modeling

- familiarity -- you can compare to Rushby
- curiosity -- I hadn’t done it before
- why this example?
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 modelling
- Familiarity -- you can compare to Rushby's Buchberger
- Curiosity -- I hadn't done it before

why this example?
nothing's easy, but everything's possible
general observations

Jackson on Alloy

Rushby on PVS

everything's easy, but nothing's possible

nothing's easy, but everything's possible
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
{part idle, trying, critical: set Process
  ticket: Process ->? Ticket,
}

}\ sig State

}\ sig Ticket

}\ sig Process

open std ord

module bakery

sigatures
{ part idle, trying, critical: set Process

  ticket: Process ->? Ticket,

  set State { ticket: Process } 

  when you’re being served
  holds no ticket: hand in ticket
  process in critical phase

} sig Ticket

} sig Process

open std/ord

module bakery

signatures
safety condition
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],
        next = Ord[Ticket].next | {p in s.trying otherTickets in s.ticket[p].^nextp in s.trying
        p in s.trying
    } |
        next = Ord[Ticket].next |
        let otherTickets = s.ticket[Process-p]' |
    Trans (s, s': State, p: Process) 
}
transition relation

fun Trans (s, s': State, p: Process) {
    let 
    otherTickets = s.ticket[p].^nextp
    p in s.trying

    s'.ticket[p] 
    s'.critical &&

    next = Ord[Ticket].next [p is in critical & no s.ticket[p]]

    let otherTickets = s.ticket[process-p] 

    ...}

precondition: p is in trying phase and all other tickets follow its ticket
fun Trans (s, s': State, p: Process) {
    let otherTickets = s.ticket[p].next | {p in s.tryingotherTickets
    in s.ticket[p].^nextp
    in s'.critical &&
    no s'.ticket[p]}
or …
}

| next = Ord[Ticket].next
let otherTickets = s.ticket[Process-p].
| next |

Transition Relation

precondition:
no ticket
after, and holds
p is in critical phase
postcondition:
p in s.critical && no s.ticket[p]

p in s.trying && no s.ticket[p]
p in s.trying

p is in critical phase after, and holds
no ticket
other cases
Trans(s, s': State, p: Process) {
    let otherTickets = s.ticket[p] & otherTickets, next
    or p \in s'.trying
          s'.ticket[p] = Ord[Ticket].next | …
    or p \in s'.idle
          {p \in s.critical
              s'.ticket[p] = s.ticket[p]
              s'.ticket[p] = s'.ticket[p]}
    or ...
    | otherTickets = s.ticket[p], next = Ord[Ticket].next
    let otherTickets = s.ticket[Process-p], next = Ord[Ticket].next
}

fun Trans(s', s: State, p: Process)
Frame condition
frame condition

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

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

This NoChange function returns `true` if the ticket of process `p` remains unchanged between states `s` and `s'`. Therefore, we can frame condition:

frame condition
initial condition
fun Init (s::State) {
  safe (s)
}

Initial condition
putting things together
Putting things together

```plaintext
fun Interleaving ()
{
    { }
    { }
        all x : Process - p
          Nocchange(s,s',x)
        (trans (s,s',p))
    } some p : Process
    | all s : State - Ord[State].last, s' : Ord[State].next[s]
        Init (Ord[State].first)
    }
```
putting things together
allowing simultaneous actions
allowing simultaneous actions

fun Simultaneity() =
  \{ 
    all p : Process | Trans (s,s',p) or NoChange(s,s',p)
    | all s : State - Ord[State].last, s' : Ord[State].next[s]
      Init (Ord[State].first)
  \}
Checking a conjecture
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?
how much assurance?
2 processes ... seems reasonable.

check InterleavingSafe for 4 but 2 processes.

How much assurance?
InterleavingSafe

for 4 Process \[ Process \leq \begin{array}{c}
\text{Analysis within bounded scope:} \\
\text{how much assurance?}
\end{array}
\]

\[ \text{check InterleavingSafe for 4 Process} \]

\[ \text{not considering all states may miss bugs} \]

\[ \text{Running out of tickets is a poor approximation} \]

\[ \text{4 tickets? 4 states? \ldots not at all reasonable} \]

\[ \text{We've learned something about a real scenario} \]

\[ \text{2 processes \ldots seems reasonable} \]
when is a trace long enough?
when is a trace long enough?

but how long? i.e., what is scope of state?

for safety properties, check all traces
when is a trace long enough?

enough to consider only traces \( \leq k \)

if all states reached in path \( \leq k \)

idea: bound the diameter

but how long? i.e., what is scope of State?

for safety properties, check all traces

when is a trace long enough?
when is a trace long enough? 

strategy

enough to consider only traces \( \leq k \)

if all states reached in path \( \leq k \)

idea: bound the diameter

tighter bounds possible: e.g., no shortcuts

if none, then \( k \) is a bound

ask for loopless trace of length \( k+1 \)

but how long? i.e., what is scope of states?
When is a trace long enough?

but can express conditions directly
like bounded model checking

tighter bounds possible: e.g., no shortcuts

if none, then k is a bound

ask for loopless trace of length k+1

Strategy

enough to consider only traces ≤ k

if all states reached in path ≤ k

Idea: bound the diameter

but how long? i.e., what is scope of state?
when is a trace long enough?

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 \( k \) if all states reached in path \( k \)

idea: bound the diameter

max loopless = 1
diameter = 1

but how long? ie, what is scope of State?

for safety properties, check all traces

but can express conditions directly

like bounded model checking
when is a trace long enough?

strategy

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 $k+1$

(diameter = 1

max loopless = 5

diameter = 1

max loopless = 4

diameter = 1

max loopless = 3

diameter = 1

max loopless = 1

diameter = 1

max loopless = 1

for safety properties, check all traces

enough to consider only traces $< k$

if all states reached in path $< k$

idea: bound the diameter

but how long? ie, what is scope of State?

When is a trace long enough?
Finding the diameter
Finding the diameter

run Norepetitions for 3 but 2 Process, 8 State
{
    s'.idle = s.idle & s.critical = s'.critical
    s.ticket = s'.ticket
}

fun Equiv (s, s': State) {
    no disj s.idle & Equiv (s, s')
    Interleaving ()
}

fun Norepetitions ()

can we fix the tickets in the same way?
but know that we never run out of tickets
 bounded the ticket scope for fast analysis
 what we want to do

Can we fix the tickets in the same way?
can we fix the tickets in the same way?

- 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
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
defining the order

introduce process ordering as a new field

StateWithOrder

extends State { precedes: Process -> Process }

{ all p, p': Process | p->p' in precedes iff ticket[p'] in (ord(Ticket).next)(ticket[p]) }

fact { ticket[p] in (ord(Ticket).next)(ticket[p]) }

all p, p': Process

precedes: Process -> Process

StateWithOrder extends State

introduce process ordering as a new field

defining the order
defining state equivalence
defining state equivalence

define equivalence modulo ordering

```javascript
define state equivalence

defining state equivalence
```
defining state equivalence

define equivalence modulo ordering

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

fun NoRepetitionsUnderOrdered () {
  Interleaving ()
  (no disj s,s' : State | EquivProcessOrder (s,s'))
}

define no repetition constraint

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

define equivalence modulo ordering

defining state equivalence
finding the bounds
Finding the bounds

run NonRepetitiousUnderordered for 7 but 3 Process, 13 State

and a diameter

finding the bounds
finding the bounds

\[
\text{check EnoughTickets} \text{ for 7 but 3 Process, 12 State}
\]

\{
\text{Interleaving () => Ord[Ticket] last in State.ticket [Process]}
\}

\assert EnoughTickets

check that tickets not all used

\run NonRepetitiousUnderordered for 7 but 3 Process, 13 State

find a diameter

Finding the bounds
Finding the bounds

so now we know

checking that tickets not all used

assert EnoughTicketsI

for 7 but 3 Process

run NonRepetitiousUnderordered for 7 but 3 Process

Find a diameter
getting full coverage
Finally, we check this.

Getting full coverage.
getting full coverage

Finally, we check this

\[ \text{InterleavingSafe for 7 but 3 Process, 12 State} \]

we have a 'proof' for 3 processes

if no counterexample

\[ \text{InterleavingSafe for 7 but 3 Process, 12 State} \]
what we did
what we did

unbounded model of bakery

• no fixed number of processes or tickets
what we did

unbounded model of bakery
  › no fixed number of processes or tickets

analysis in small finite scope
  › may miss counterexamples
what we did

established diameter

may miss counterexamples

in small finite scope

no fixed number of processes or tickets

unbounded model of bakery

What we did
what we did
summary of Alloy
A simple language: \[\text{signature for structuring: global relations}\]
\[\text{relational first-order logic}\]
\[\text{set of constraints}\]

Summary of Alloy
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

summary of Alloy
summary of Alloy

A simple language
• relational first-order logic
• signatures for structuring: global relations
• 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

used for teaching in ~15 universities

used for teaching in ~15 universities

used for teaching in ~15 universities

used for teaching in ~15 universities

used for teaching in ~15 universities
challenges: better analysis
choosing symmetry predicates?
eliminating irrelevant constraints?
exploiting equationalities?
improving analyses

challenges: better analyses
challenges: better analysis

- decision procedure for subset?
- data independence: scope of 3 enough?
- mitigating effects of scope
- choosing symmetry predicates?
- eliminating irrelevant constraints?
- exploiting equality
- improving analysis
challenges: better analysis

• 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?

exploring equality

improving analysis

challenges: better analysis
challenges: applications
challenges: applications

- counterexample is trace
  \[ \text{pre}(s) \land \text{p}(s, s'_0, s'_1, \ldots, s') \land \text{post}(s, s') \]
- check the conjecture \( \text{p}(s, s'_0, s'_1, \ldots, s') \)
- extract formula from procedure
- finding bugs in code
challenges: applications

- finding bugs in code
- extract formula from procedure
- check the conjecture
- counterexample is trace

build veneers on Alloy

eg, role-based access control
eg, semantic web design
eg, API, or as macro language

pre(s, p(s,s0,s1,⋯,s′″′) => post(s,s′″′))

eg, role-based access control
eg, semantic web design

challengess: applications
challenges: case studies
challenges: case studies

• model CVS at multiple levels
• source code control

• is it correct?
challenges: case studies

- check theorems of Unified Theory?
- check consistency of UML metamodeling
- is it correct?
- model CVS at multiple levels
- source code control
challenges: case studies

- reverse path forwarding, eg
- 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