Lecture 1: Introduction

and analysis with Alloy
declarative modelling
micromodels of software
lightweight models

a foundation for robust, usable programs

• high cost of failure
• structure-determining
• hard to get right, or to check
• focus on risky aspects

• full automation
• partial models & analyses
• small & simple notations

Elements

lightweight models
what assurance costs
my work in marke/toberdorf context

but much less mature than ACL2, PVS, Statemate, etc

started this in 1994, and have had some successes

semantics in terms of sets, and SAT

only way to make things simple

role of mathematics

simulation, not just checking

like Harel, not Rushby & Moore

designed for experts, but not super-experts

deriving idioms due to Hoare, Woodcock et al

relational, not algebraic (cf. Tarlecki and Meseguer)

complementary to Harel & Pnueli

computation, not interaction
Features of Alloy

- Fully automatic
- Simulation & checking
- Analyzable
- Describes system as collection of constraints
- A full logic, with conjunction and negation
- Declarative
- With just a few powerful operators
- Express complex structure, static and dynamic
- Structural
There is no problem in computer science that cannot be solved by introducing another level of indirection, but that usually reveals new problems.

—David Wheeler

There is no problem in computer science that cannot be solved by introducing another level of indirection, but that usually reveals new problems.

Structural

Structural
why Less is more

Sys meets Prop: Sys ⊢ Prop

declarative description model is collection of properties

the more you say, the less happens

less constrained environment means greater freedom

why Less is more

Sys meets Prop: Sys ⊢ Prop

declarative: no separate language of properties

simplicity: no separate language of properties

partially: doesn’t require special constructs

incrementality: to say more, add a property

advantages

Less constrained system means implementation freedom

Sys meets Prop: Sys ⊢ Prop

declarative description model is collection of properties

the more you say, the less happens

less constrained environment means greater freedom
analyzable

unlike testing, billions cases/second
like testing, sound but not complete
concrete: generates samples & counterexamples
fully automatic, with no user intervention

Alloy's analysis

......

optimize for failing case: most of my examples will be wrong
catch errors early, develop confidence
simulate and check incrementally
tool-assisted modelling

but missed opportunity (and wishful thinking)

write-only models

analyzable
Alloy’s analysis can execute a model with right analysis technology but can have cake and eat it. Good arguments for both declarative XOR executable traditionally.

Traditionally declarative XOR executable? no ad hoc restrictions on logic without test cases forwards or backwards Alloy’s analysis can execute a model Small Tower of 6 Gears, Arthur Ganson
Given a numbering problem

(I like my Marktoberdorf notes)

A numbering problem can be solved by:

- Document with numbered paragraphs
- Style sheet that gives numbering rules for styles
- Document whose paragraphs are tagged with styles

Given a numbering problem...
a candidate solution

style sheet assigns to each style
> an initial value for numbering
> optionally, a parent

Part A: Introduction
A.1 Motivation
A.1.1 Why?
A.2 Overview
Part B: Conclusions
B.1 Unrelated Work
How to define acyclic:

```prolog
fun Acyclic [t] (r: t -> t) { no iden[t] & ^r }
```

Constrain parent relation to be acyclic:

```prolog
fact {Acyclic (parent)}
```

Ask for a sample:

```prolog
run Show
run Show ()
run Show ()
```

Declare styles & parent relation:

```prolog
sig Style {option: parent}
```

**Styles**
run Show
{some parent}
} () run Show
ask for a sample

{fact Style = NumberedStyle}
{sig NumberedStyle extends Style {init: Number}
add numbers to styles

{this != next}
{next: option Number
} sig Number
introduce numbers

numbers
for 2 but 1 Numbering
run ShowNumbering
{run ShowNumbering ()
{some num
ask for a sample

{num: Style —> Number
} sig Numbering
declare Numbering

numNumbering
fun Next (n, n': Numbering, s: Style) {n'.num =

if no n.num[s] then s.init else n.num[s].next
+ {d: s.parent, x: Number | x = n.num[d]}

}
16

next showing next

run Next for 3 but 2 Numbering

with initial value is numbered grandchild style s
run ShowNext for 3 but 2 Numbering

{ next (n,n',s) if some n.num = parent
}

run ShowNext (n,n': Numbering, s: Style)
Next \((n, n': \text{Numbering}, s: \text{Style})\) { 
  let 
  \(i = n.\text{num}[s] \rightarrow \text{some} i \rightarrow \text{next} n'.\text{num} = \{d: s.\text{parent}, x: Number | x = n.\text{num}[d]\} + s \rightarrow \text{if no n.\text{num}[s] then s.init else n.\text{num}[s].\text{next}\} \}

Fixing the operation
run ShowNext for 3 but 2 Numbering

```java
ShowNext (n,n': Numbering, s: Style) {
Next (n,n',s) &&
some n.num[s.par]
&&
some n.num[s]
}
```

guiding the simulation
checking a property

If style is not a parent, step is reversible

assert Reversible

\{ next(0, n', s) \Rightarrow next(n, n', s) \Rightarrow n_0.nm = n_1.nm \}

assert Reversible

if style is not a parent, step is reversible

checking a property
Trying again...

does this fix the problem?

\texttt{fact \{Injective (next)\}}

make numbering injective

...
counterexample

after numbering

n

adjacent style has no number afterwards
counterexample, ctd

before numbers

In and no adjacent style’s number before was irrelevant! so value eliminated, so value before was irrelevant!
masking

check ReversibleWhenLine

assert ReversibleWhenLine

check ReversibleWhenLine from a line again, assuming styles form a line

check ReversibleWhenLine
Counterexample, again.
checking a refactoring

are these equivalent?

```plaintext
fun Next1 (n,n': Numbering, s: Style) { n'.num = if n.num[s] = [d: s.parent | n.num[d] = n.num[s].next] then n.num[s] else n.num[s].next
        n'.num[d] = n.num[d]
        n'.num[s] = if no n.num[s] then s.init else n.num[s].next
        all d: s.parent | n.num[d] = n.num[d]
        all n,n': Numbering, s: Style | Next1(n,n',s) iff Next2(n,n',s)
}

assert Same:

ask the tool:

{ all n,n': Numbering, s: Style | Next1(n,n',s) iff Next2(n,n',s) }

fun Next2 (n,n': Numbering, s: Style) {
    all d: s.parent | n.num[d] = n.num[d]
    all n,n': Numbering, s: Style
    n'.num[d] = n.num[d]
    n'.num[s] = if no n.num[s] then s.init else n.num[s].next
    n'.num[s] = if no n.num[s] then s.init else n.num[s].next
    s <- s
    + {
        [p,d]: s.parent, x: Number, p: s.parent, x = n.num[p]
        n'.num = n.num
    }
}

fun Next2 (n,n': Numbering, s: Style)

are these equivalent?

checking a refactoring
```
incrementality

analyses prompted questions

encourages small models

avoids wasted time

to get key properties

constraint just enough

write a bit, analyze a bit

what happened
declarative vs. operational development.

Declarative:
- No behaviours;
satisfies no safety properties.

Operational:
- Satisfies all safety properties;
all behaviours;
What's been done?

- Chord peer-to-peer lookup (Whe)
- Intentional naming (Khursheed)
- Analyzing implemented systems
- Transaction cache (Tucker)
- UML meta model (Vaziri, from OCL)
- Unison file synchronizer (Nolte, from Pierce's math)
- Firewire Leader Election (me, from Vandersager's IOA)
- Microsoft COM (Sullivan, from Z)
- Analyzing existing models

Typically:

- Classic distributed algorithms (Shlyakhter, from SMV)
- Intentional naming (Khurshid)

200 lines of Alloy, 30-200 hours work
Intentional naming

Example: Intentional naming

Result of query is set of simple names

Query scheme

Intentional names are trees
31 results

what we did: analyzed claims made in paper: mostly untrue
analyzed algebraic properties: also untrue
e.g., add is monotonic
analyzed algebraic properties: also untrue
analyzed claims made in paper: mostly untrue

Initial analysis took 2 weeks and 100 lines of Alloy

Reflections

2000 lines of tests hadn't found bugs in a year
found all bugs in trees of 4 nodes or less -- approx 10 secs

developed new semantics & checked it
adapted model for fixes in code: also broken
eg, add is monotonic
analyzed algebraic properties: also untrue
analyzed claims made in paper: mostly untrue
what we did...

Results
challenge: get numbering right

- section has parent chapter and appendix
- multiple parents
- section and figure have parent chapter
- multiple children

fix the numbering mechanism to handle
What is a model?

- A representation of a system to be built
- To explain & evaluate existing system
- To explore design of system to be built
- More or less useful, not more or less correct [Fowler]

[useful to the extent that it answers questions [Ross]

Role of a model
why model?

plan to throw one away [Brooks]

› 100 line model or 100k lines of code?
› nasty surprises happen sooner

› not a good use of testing

› conceptual flaws get mired in code

› separation of concerns

designs with clear conceptual models easier to use and implement allow delegation & division of labour

not a good use of testing
lightweight formal methods

elements
› small & simple notations
› partial models & analyses
› full automation

focus on risky aspects
› hard to get right, or to check
› structure-determining
› high cost of failure