a puzzle

why doesn’t pencil & paper design work?
The first principle is that you must not fool yourself, and you are the easiest person to fool.

Richard P. Feynman
early formalization
early formalization

formalize design
early formalization

formalize design

$Z$, $VDM$, $B$, ...
early formalization

formalize design

$Z, VDM, B, ...$

exercise design
early formalization

formalize design
Z, VDM, B, ...

exercise design
write proofs
early formalization

**formal methods**

formalize design

Z, VDM, B, ...

exercise design

write proofs
early formalization

formal methods

formalize design
Z, VDM, B, ...

not code

exercise design
write proofs

very painful
early formalization

formal methods
- formalize design
  - $Z$, VDM, B, ...
  - not code
- exercise design
  - write proofs
  - very painful

agile methods
early formalization

**formal methods**
- formalize design
  - Z, VDM, B, ...
  - *not code*
- exercise design
  - write proofs
  - very painful

**agile methods**
- formalize design
early formalization

**formal methods**
- formalize design
  - Z, VDM, B, ...
  - *not code*
- exercise design
  - write proofs
  - very painful

**agile methods**
- formalize design
  - Java, ML, Ruby, C, ...
early formalization

**formal methods**
- formalize design
  - Z, VDM, B, ...
  - not code
- exercise design
  - write proofs
  - very painful

**agile methods**
- formalize design
  - Java, ML, Ruby, C, ...
- exercise design
<table>
<thead>
<tr>
<th>formal methods</th>
<th>agile methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalize design</td>
<td>formalize design</td>
</tr>
<tr>
<td>Z, VDM, B, ...</td>
<td>Java, ML, Ruby, C,</td>
</tr>
<tr>
<td>not code</td>
<td>...</td>
</tr>
<tr>
<td>exercise design</td>
<td>exercise design</td>
</tr>
<tr>
<td>write proofs</td>
<td>write tests</td>
</tr>
<tr>
<td>very painful</td>
<td></td>
</tr>
</tbody>
</table>
early formalization

**formal methods**
- formalize design
  - Z, VDM, B, ...
  - *not code*
- exercise design
  - write proofs
  - *very painful*

**agile methods**
- formalize design
  - Java, ML, Ruby, C, ...
  - *verbose*
- exercise design
  - write tests
  - *painful
  - poor coverage*
a dream?

succinct, expressive models
animation & property checking
high coverage without proofs
no unit tests to write
code synthesized automatically
a heresy

Small Tower of 6 Gears, Arthur Ganson
a heresy

what matters most?

Small Tower of 6 Gears, Arthur Ganson
a heresy

what matters most?

finding subtle bugs?

Small Tower of 6 Gears, Arthur Ganson
a heresy

what matters most?
finding subtle bugs?
instant feedback

Small Tower of 6 Gears, Arthur Ganson
a heresy

what matters most?
finding subtle bugs?
instant feedback

Small Tower of 6 Gears, Arthur Ganson
transatlantic alloy

motivation

structural description + automation
transatlantic alloy

motivation

structural description + automation

Oxford, home of Z
transatlantic alloy

motivation

structural description + automation

Oxford, home of Z

Pittsburgh, home of SMV
# how we got here

<table>
<thead>
<tr>
<th>version</th>
<th>language</th>
<th>analysis</th>
<th>sample case study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nitpick (1995)</td>
<td>relational calculus subset of Z</td>
<td>relation enumeration</td>
<td>IPv6 routing</td>
</tr>
<tr>
<td>Alloy 1 (1999)</td>
<td>+ navigation exps quantifiers</td>
<td>WalkSAT, DP</td>
<td>intentional naming</td>
</tr>
<tr>
<td>Alloy 2 (2001)</td>
<td>+ relational ops higher arity</td>
<td>Chaff, Berkmin symmetry, sharing</td>
<td>key management, Unison filesync</td>
</tr>
<tr>
<td>Alloy 4 (2007)</td>
<td>+ meta, sequences, arithmetic</td>
<td>sparse matrices better sharing</td>
<td>flash file systems, code analyses</td>
</tr>
</tbody>
</table>
example
simplified voting

voters

ballots

boxes

tally

Alice
Bob
Carol

125
63
168

What properties establish this requirement?
sig Candidate {}
sig Voter { ballot: lone Ballot }
sig Ballot { choice: lone Candidate}
sig Box { cast: set Ballot }

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int
}

all c: Candidate | count[c] = #(ballots & choice.c)
animation
animation

run {}
animation

instance

Tally

ballots

Ballot

run {}
run { }
run { some cast }
animation

instance

run {}
run {}
animation

instance

run {}

run { some cast }

... show next
run {}
... show next
Adding facts

```
sig Candidate {}
sig Voter { ballot: lone Ballot }
sig Ballot { choice: lone Candidate }
sig Box { cast: set Ballot }
one sig Tally { ballots: set Ballot, count: Candidate -> one Int }
  { all c: Candidate | count[c] = #(ballots & choice.c) }
```

```
# All voters cast ballots
fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

# No box stuffing
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

# No tally stuffing
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

# No ballots lost
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```
fact AllVotersCastBallots {
    all v: Voter | some x: Box | v.ballot in x.cast
}

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
adding facts

sig Candidate {}

sig Voter { ballot: lone Ballot }

sig Ballot { choice: lone Candidate }

sig Box { cast: set Ballot }

one sig Tally { ballots: set Ballot, count: Candidate -> one Int }
  { all c: Candidate | count[c] = #(ballots & choice.c) }

fact AllVotersCastBallots {
  all v: Voter | some x: Box | v.ballot in x.cast
}

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

run { }
adding facts

**sig** Candidate {}
**sig** Voter { ballot: lone Ballot }
**sig** Ballot { choice: lone Candidate}
**sig** Box { cast: set Ballot }

**one sig** Tally { ballots: set Ballot, count: Candidate -> one Int }  
{ all c: Candidate | count[c] = #(ballots & choice.c) }

**fact** AllVotersCastBallots {  
  all v: Voter | some x: Box | v.ballot in x.cast
}

**fact** NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }
**fact** NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }
**fact** NoBallotsLost { all x: Box | x.cast in Tally.ballots }

run { }
checking an assertion

sig Candidate {}

sig Voter { ballot: lone Ballot }

sig Ballot { choice: lone Candidate }

sig Box { cast: set Ballot }

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int }

{ all c: Candidate | count[c] = #(ballots & choice.c) }

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBox Stuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTally Stuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallots Lost { all x: Box | x.cast in Tally.ballots }
checking an assertion

sig Candidate {}
sig Voter { ballot: lone Ballot }
sig Ballot { choice: lone Candidate}
sig Box { cast: set Ballot }

one sig Tally {
    ballots: set Ballot,
    count: Candidate -> one Int
}
{
    all c: Candidate | count[c] = #(ballots & choice.c)
}

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

pred CountingCorrect { all c: Candidate | Tally.count[c] = #ballot.choice.c }
check CountingCorrect
sig Candidate {}
sig Voter { ballot: lone Ballot }
sig Ballot { choice: lone Candidate}
sig Box { cast: set Ballot }

one sig Tally {
    ballots: set Ballot,
    count: Candidate -> one Int
}
{
    all c: Candidate | count[c] = #(ballots & choice.c)
}

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

pred CountingCorrect { all c: Candidate | Tally.count[c] = #ballot.choice.c }

check CountingCorrect
**fixing & rechecking**

```prolog
sig Candidate {}

sig Voter { ballot: lone Ballot }

sig Ballot { choice: lone Candidate }

sig Box { cast: set Ballot }

one sig Tally {
    ballots: set Ballot,
    count: Candidate -> one Int
}

{ all c: Candidate | count[c] = #(ballots & choice.c) }

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```
fixing & rechecking

```
sig Candidate {}
sig Voter { ballot: lone Ballot }
sig Ballot { choice: lone Candidate}
sig Box { cast: set Ballot }
one sig Tally {
    ballots: set Ballot,
    count: Candidate -> one Int }
{
    all c: Candidate | count[c] = #(ballots & choice.c)
}

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }
check CountingCorrect for 5
```
fixing & rechecking

```plaintext
sig Candidate {}

sig Voter { ballot: Ballot }

sig Ballot { choice: Candidate }

sig Box { cast: set Ballot }

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int
}

{ all c: Candidate | count[c] = #(ballots & choice.c) }

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
```
 Executing "Check CountingCorrect for 5"
Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=2 Symmetry=20
3693 vars. 235 primary vars. 8238 clauses. 180ms.
No counterexample found. Assertion may be valid. 92ms.

**fixed & rechecking**

```
sig Candidate
sig Voter
sig Ballot
sig Box

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int } 
{
  all c: Candidate | count[c] = #(ballots & choice.c)
}

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot in x.cast }
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
```
fixing & rechecking

sig Candidate {};
sig Voter { ballot: Ballot };
sig Ballot { choice: Candidate };
sig Box { cast: set Ballot };

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int
}
{
  all c: Candidate | count[c] = #(ballots & choice.c)
}

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
Executing "Check CountingCorrect for 5"
Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=2 Symmetry=20 3693 vars. 235 primary vars. 8238 clauses. 180ms.
No counterexample found. Assertion may be valid. 92ms.

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
**fixing & rechecking**

```plaintext
sig Candidate {
}
sig Voter {
  ballot: Ballot
}
sig Ballot {
  choice: Candidate
}
sig Box {
  cast: Ballot
}

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int
}

{ all c: Candidate | count[c] = #(ballots & choice.c) }

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot in x.cast }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
```
fixing & rechecking

```
one sig Tally {
    ballots: set Ballot,
    count: Candidate -> one Int }
{
    all c: Candidate | count[c] = #(ballots & choice.c)
}

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
```
fixing & rechecking

```
sig Candidate {}
sig Voter { ballot: Ballot }
sig Ballot { choice: Candidate }
sig Box { cast: set Ballot }

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int
}

{ all c: Candidate | count[c] = #(ballots & choice.c) }

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot in x.cast }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
```

Total space: $2^{235}$
fixing & rechecking

```
Executing "Check CountingCorrect for 5"
Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=2 Symmetry=20
3693 vars, 235 primary vars, 8238 clauses. 180ms.
No counterexample found. Assertion may be valid. 92ms.
```

```
sig Candidate {}
sig Voter { ballot: Ballot }
sig Ballot { choice: Candidate }
sig Box { cast: set Ballot }

one sig Tally {
  ballots: set Ballot,
  count: Candidate -> one Int
}

all c: Candidate | count[c] = #(ballots & choice.c)

fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }

fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = b }

fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.cast }

fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }

fact NoSharedBallots { all b: Ballot | lone ballot.b }

check CountingCorrect for 5
```
what makes it work?
five ideas

analysis ideas
analysis as solving
small scope analysis
solving by SAT

language ideas
generalizing dot
funky idioms
idea: analysis = solving
idea: analysis = solving

does system S have property P?

\[ S(beh) \land \neg P(beh) \]
idea: analysis = solving

does system S have property P?
\[ S(beh) \land \neg P(beh) \]

is operation op deterministic?
\[ op(s,s') \land op(s,s'') \land s' \neq s'' \]
idea: analysis = solving

does system S have property P?
\[ S(\text{beh}) \land \neg P(\text{beh}) \]

is operation op deterministic?
\[ op(s,s') \land op(s,s'') \land s' \neq s'' \]

does method m meet pre, post?
\[ m(s,s') \land pre(s) \land \neg post(s,s') \]
idea: analysis = solving

does system S have property P?
\[ S(\text{beh}) \land \neg P(\text{beh}) \]

is operation \( op \) deterministic?
\[ op(s,s') \land op(s,s'') \land s' \neq s'' \]

does method \( m \) meet \( pre, post \)?
\[ m(s,s') \land pre(s) \land \neg post(s,s') \]

is \( op' \) a valid refactoring of \( op \)?
\[ op'(s,s') \land \neg op(s,s') \]
sound analysis?
traditional view
traditional view

analysis should be ‘sound’

consider all cases

ensures no bugs missed

is a cast-iron guarantee
traditional view

analysis should be ‘sound’
consider all cases
ensures no bugs missed
is a cast-iron guarantee

but sound ⇒ spurious errors
⇒ annoyance
⇒ bugs missed
traditional view

analysis should be ‘sound’
    consider all cases
    ensures no bugs missed
    is a cast-iron guarantee

but sound ⇒ spurious errors
    ⇒ annoyance
    ⇒ bugs missed

and not a guarantee anyway?
    even theorem proving isn’t watertight
    inconsistent axioms may undermine proof
idea: small scope analysis
idea: small scope analysis

random testing: a few big tests
idea: small scope analysis

random testing: a few big tests

small scope analysis: ‘all small tests’
idea: generalizing dot $x.f$
idea: generalizing dot

\( x \cdot f \)

- Voter
  - ballot
- Box
  - cast
- Ballot
  - choice
- Candidate
idea: generalizing dot

\[ x \cdot f \]\n
\[ v.b\text{ballot} \]
idea: generalizing dot

\( x \cdot f \)

\( v.ballot \)

\( v.ballot.choice \)

Diagram:
- **Ballot**
  - ballot
  - choice
- **Voter**
- **Box**
  - cast
- **Candidate**
idea: generalizing dot

\[ x.f \]

- \[ v.\text{ballot} \]
- \[ v.\text{ballot.choice} \]
- \[ b.\text{cast} \]
idea: generalizing dot

\[ x.f \]

\[ \text{v.ballot} \]
\[ \text{v.ballot.choice} \]
\[ \text{b.cast} \]
\[ \text{b.cast.choice} \]

Diagram:
- Voter
  - ballot
- Ballot
  - cast
  - choice
- Candidate
idea: generalizing dot

\[ x.f \]

- v.ballot
- v.ballot.choice
- b.cast
- b.cast.choice
- choice.c

- Ballot
  - Voter
  - Box
    - ballot
    - cast
    - choice

Candidate
idea: generalizing dot

\[
x \cdot f
\]

\[
v.b\text{allot}
v.b\text{allot.choice}
b.c\text{ast}
b.c\text{ast.choice}
choice.c
ballot.choice.c
\]

- Voter
- Ballot
- Candidate
- Box

ballot

choice

cast
a definition

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
a definition

generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]

v.ballot.choice
a definition

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
a definition

generalized join

\[ a.b = \{ (a_0,\ldots, a_{n-1}, b_1,\ldots, b_m) \mid (a_0,\ldots, a_n) \in a \land (a_n, b_1,\ldots, b_m) \in b \} \]
A definition

Generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]

<table>
<thead>
<tr>
<th>V0</th>
<th>V0</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>B1</td>
<td></td>
</tr>
</tbody>
</table>
a definition

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
a definition

generalized join

$$a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \}$$
a definition

generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) | (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]
A definition

Generalized join

\[ a \cdot b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
a definition

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
a definition

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
right associating

generalized join

\[ a \cdot b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]
right associating

generalized join

\[ a.b = \{ (a_0,\ldots,a_{n-1},b_1,\ldots,b_m) \mid (a_0,\ldots,a_n) \in a \land (a_n, b_1,\ldots,b_m) \in b \} \]
right associating

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]

\[ \text{ballot} \cdot \text{choice} \]

<table>
<thead>
<tr>
<th>V0</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>B2</td>
</tr>
<tr>
<td>V2</td>
<td>B1</td>
</tr>
</tbody>
</table>
right associating

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
right associating

generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]

<table>
<thead>
<tr>
<th>V0</th>
<th>B0</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>B1</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>B0</th>
<th>C0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>B2</td>
<td>C0</td>
</tr>
</tbody>
</table>
right associating

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
right associating

generalized join

\[ a.b = \{ (a_0,\ldots, a_{n-1}, b_1,\ldots, b_m) \mid (a_0,\ldots, a_n) \in a \land (a_n, b_1,\ldots, b_m) \in b \} \]

```
V0 C0
V1 C0
V2 C1
```
right associating

generalized join

$$a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \}$$
right associating

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
right associating

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
right associating

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
navigating back

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
navigating back

generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]
navigating back

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
navigating back

generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]

ballot.choice.c

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>C0</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>B2</td>
<td>C0</td>
</tr>
</tbody>
</table>
navigating back

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
navigating back

generalized join

\[ a \cdot b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
navigating back

generalized join

\[
a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \}\n\]
navigating back

generalized join

\[ a.b = \{ (a_0, ..., a_{n-1}, b_1, ..., b_m) \mid (a_0, ..., a_n) \in a \land (a_n, b_1, ..., b_m) \in b \} \]
navigating back

generalized join

\[ a.b = \{ (a_0, \ldots, a_{n-1}, b_1, \ldots, b_m) \mid (a_0, \ldots, a_n) \in a \land (a_n, b_1, \ldots, b_m) \in b \} \]
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]
idea: solving by SAT

v \cdot \text{ballot} = b
idea: solving by SAT

\[ v \cdot ballot = b \]

<table>
<thead>
<tr>
<th>( V0 )</th>
<th>( V1 )</th>
<th>( V2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]

<table>
<thead>
<tr>
<th>( V0 )</th>
<th>( V1 )</th>
<th>( V2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]

<table>
<thead>
<tr>
<th>$v_0$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0</td>
</tr>
<tr>
<td>$v_2$</td>
<td>0</td>
</tr>
</tbody>
</table>
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]

| \(V0\) | 1 |
| \(V1\) | 0 |
| \(V2\) | 0 |
| \(B0\) | 1 |
| \(B1\) | 0 |
| \(B2\) | 0 |

[Table with values]
idea: solving by SAT

v \cdot \text{ballot} = b

<table>
<thead>
<tr>
<th>V0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>0</td>
</tr>
<tr>
<td>V2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>0</td>
</tr>
</tbody>
</table>
idea: solving by SAT

V. ballot = β
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]
idea: solving by SAT

v \cdot \text{ballot} = b
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]
idea: solving by SAT

\[ \mathbf{v} \cdot \mathbf{ballot} = \mathbf{b} \]
idea: solving by SAT

\[ \mathbf{v} \cdot \text{ballot} = b \]
idea: solving by SAT

v \cdot \text{ballot} = b
idea: solving by SAT

\[ v \cdot \text{ballot} = b \]

\[ b_0 = v_0 \land f_{00} \lor v_1 \land f_{10} \lor v_2 \land f_{20} \]
\[ b_1 = v_0 \land f_{01} \lor v_1 \land f_{11} \lor v_2 \land f_{21} \]
\[ b_2 = v_0 \land f_{02} \lor v_1 \land f_{12} \lor v_2 \land f_{22} \]
idea: funky idioms
idea: funky idioms

alloy is just a logic

with subtypes, navigation, etc

but no states or mutation
idea: funky idioms

alloy is just a logic

with subtypes, navigation, etc

but no states or mutation

for dynamic models

many ways to represent behaviour
idea: funky idioms

alloy is just a logic

with subtypes, navigation, etc

but no states or mutation

for dynamic models

many ways to represent behaviour

a surprise

least conventional most useful
funky idioms: stateful objects

sig Candidate {}

sig Voter { intent: Candidate }

sig Ballot { choice: Candidate }

sig Tally { ballots: set Ballot, count: Candidate -> one Int }

sig Precinct {
    registered: set Voter,  
    cast: Ballot -> Time,  
    voted: Voter -> Time
}

funky idioms: stateful objects

```plaintext
sig Candidate {}
sig Voter { intent: Candidate }
sig Ballot { choice: Candidate }
sig Tally { ballots: set Ballot, count: Candidate -> one Int }
sig Precinct {
    registered: set Voter,
    cast: Ballot -> Time,
    voted: Voter -> Time
}
```

cast: a ternary relation
**funky idioms: stateful objects**

```
sig Candidate {}

sig Voter { intent: Candidate }

sig Ballot { choice: Candidate }

sig Tally { ballots: set Ballot, count: Candidate -> one Int }

sig Precinct {
    registered: set Voter,
    cast: Ballot -> Time,
    voted: Voter -> Time
}
```

cast: a ternary relation

cast: Precinct -> Ballot -> Time

p.cast: Ballot -> Time

cast.t: Precinct -> Ballot
funky idioms: stateful objects

\textbf{sig} Candidate {}
\textbf{sig} Voter { intent: Candidate }
\textbf{sig} Ballot { choice: Candidate }
\textbf{sig} Tally { ballots: set Ballot, count: Candidate \rightarrow \textbf{one} Int }
\textbf{sig} Precinct {
  registered: set Voter,
  cast: Ballot \rightarrow Time,
  voted: Voter \rightarrow Time
}

\textit{cast: a ternary relation}
\textit{cast: Precinct \rightarrow Ballot \rightarrow Time}
\textit{p.cast: Ballot \rightarrow Time}
\textit{cast.t: Precinct \rightarrow Ballot}

\textit{cast: projected onto Time}
open util/ordering[Time]

sig Time {}

abstract sig Event {before, after: Time}

pred Event.follows (that: Event) { this.before = that.after }

fact {
    all t: Time - last | one before.t
    all e: Event | e.after = e.before.next
}

-- one event starting in every time instant
-- events execute in time order
**funky idioms: events**

```plaintext
open util/ordering[Time]
sig Time {}
abstract sig Event {before, after: Time}
pred Event.follows (that: Event) { this.before = that.after }
fact {
    all t: Time - last | one before.t
    all e: Event | e.after = e.before.next
} -- one event starting in every time instant
                             -- events execute in time order
```

*a generic model of events*
fun*ky idioms: events

open util/ordering[Time]
sig Time {}
abstract sig Event {before, after: Time}
pred Event.follows (that: Event) { this.before = that.after }
fact {
  all t: Time - last | one before.t
  all e: Event | e.after = e.before.next
}

sig Cast extends Event { voter: Voter, ballot: Ballot, precinct: Precinct } {
  ballot.choice = voter.intent
  voter in precinct.registered - precinct.voted.before
  voted.after = voted.before + precinct -> voter
  cast.after = cast.before + precinct -> ballot
}
	sig Count extends Event { tally: Tally } {
  tally.ballots = Precinct.cast.before
  -- ballots tallied are those cast in all precincts
}
funky idioms: dynamics

\[
\text{fact} \{ \\
\quad \text{no cast.first} \quad \text{-- at first, ballot boxes are empty} \\
\quad \text{all } v: \text{Voter} \mid \text{some } c: \text{Cast} \mid c.\text{voter} = v \quad \text{-- all voters cast ballots} \\
\quad \text{no } ca: \text{Cast, co: Count} \mid ca.\text{follows}[co] \quad \text{-- no casting after counting} \\
\} 
\]
funky idioms: dynamics

fact {
    no cast.first -- at first, ballot boxes are empty
    all v: Voter | some c: Cast | c.voter = v -- all voters cast ballots
    no ca: Cast, co: Count | ca.follows [co] -- no casting after counting
}

check { all e: Count | all c: Candidate | e.tally.count[c] = #intent.c } for 4 -- tally matches voter intent
Voter casts Ballot0 in Precinct0
Voter casts Ballot0 in Precinct0

Voter casts Ballot1 in Precinct1
Voter casts Ballot0 in Precinct0

Voter casts Ballot1 in Precinct1
what else?

language
  implicit subtyping
sequences, arithmetic
  meta features

analysis
  coverage by unsat core
  partial instances
  sharing, symmetry, etc
navigations are succinct
navigations are succinct

\textit{tally count for candidate c is} # intent.c
navigations are succinct

tally count for candidate $c$ is $\# \text{intent} . c$

in $Z$

$| \text{intent} \sim (\{c\}) |$
Navigations are succinct

tally count for candidate c is \# intent.c

in \( Z \)

\[ | \text{intent} \sim \{ \{ c \} \} | \]

in SQL

select count(*) from voter where intent = c
navigations are succinct

tally count for candidate $c$ is $\# \text{intent}_c$

in $\mathbb{Z}$
\[
| \text{intent} \sim (\{c\}) |
\]

in SQL
\[
\text{select count(*) from voter where intent = c}
\]

in Java
\[
aagh!
\]
some applications
design analyses

Mondex [Ramanandro, 2006]  
*reduced Z to Alloy & checked*

flash file systems [Kang, 2008]  
*designs checked against POSIX*

Chord [Zave, 2009]  
*a peer to peer protocol*

typically a few billion cases and ...  
*kloc of Alloy*  
*months of modeling*  
*minutes of analysis*
a typical story

Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance.

*Ion Stoica et al. Chord: A Scalable Peer to Peer Lookup Service for Internet Applications, SIGCOMM 2001 (also TON, 2003)*
Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance.

Ion Stoica et al. Chord: A Scalable Peer to Peer Lookup Service for Internet Applications, SIGCOMM 2001 (also TON, 2003)

Modeling and analysis have shown that the Chord routing protocol is not correct according to its specification. Furthermore, not one of the six logical properties claimed as invariant is invariantly maintained by the protocol.

Pamela Zave. Invariant-Based Verification of Routing Protocols: The Case of Chord, 2009
code analysis studies

graph libraries (Taghdiri, 2006–2008)
  Open JGraph, Quartz graph API

KOA Remote Voting System (Dennis, 2007)
  found 19 methods not meeting specs

Burr Proton Therapy Center (Seater, 2008)
  checked various dependability arguments
small scope hypothesis

analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]
small scope hypothesis

analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]
small scope hypothesis

analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]
small scope hypothesis

analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]
code analysis: jforge plugin

plugin implemented by Kuat Yessenov; underlying Forge analyzer by Greg Dennis
code synthesis

WebAlloy [Chang 2009] data model + policy in Alloy generate entire site gateway enforces policy

results
3 real websites in ~100 lines of Alloy each
declarative configuration
declarative configuration

‘We don’t need hackers to break systems because they’re falling apart by themselves’
Peter Neumann

bad configurations responsible for 80% of USAF vulnerabilities
NSA
declarative configuration

ConfigAssure
[Telcordia, 2008]
use Kodkod to find network configuration that meets rules & goals
declarative configuration

ConfigAssure [Telcordia, 2008]
use Kodkod to find network configuration that meets rules & goals
declarative configuration

ConfigAssure
[Telcordia, 2008]
use Kodkod to find network configuration that meets rules & goals
some other uses of Alloy

bytecode security checking
  Reynolds

mixed programming
  Rayside et al

product-line test generation
  Uzuncaova et al

memory model analysis
  Torlak

security policy analysis
  Krishnamurthi & Fisler, Hassan

ontology analysis
  Dong

architectural styles
  Kim & Garlan, Sullivan

HOL to Alloy
  Blanchette

UML to Alloy
  Dingel & Zito, Bordbar, Roquette

multiobjective optimization
  Rayside & Estler
a new direction
kemper arena, kansas city, 2007
kemper arena, 1979
what happened?

Levy & Salvadori, Why Buildings Fall Down
failure = flawed success story
AECL fault tree analysis (1983) did not include software

\[ P(\text{computer selects wrong energy}) = 10^{-11} \]

Leveson & Turner (1993)

race conditions, lack of interlocks, etc
A Direct Path to Dependable Software, CACM, March 2009

wordle thanks to Jonathan Feinberg, IBM Research, Cambridge
dependability arguments

solve the real problem
not just software: operators & peripherals too
top-to-bottom, end-to-end

be realistic
not ‘full correctness’: prioritize requirements

don’t be a masochist
use a safe language
design to localize critical properties
a dependence diagram

diagram by Eunsuk Kang
reflections
the future of modelling
the future of modelling

it will happen, really

domain-specific approaches

Rails is a modelling language
the future of modelling

it will happen, really

*domain-specific approaches*

*Rails is a modelling language*

animation now, analysis later

*exploration*, *test case generation*

*then maybe checking*
the future of modelling

it will happen, really

domain-specific approaches

Rails is a modelling language

animation now, analysis later

exploration, test case generation

then maybe checking

relational modelling

non-transparent ORMs must go
what I learnt from this project
what I learnt from this project

what worked

*drive research by case studies*
*robust tools: x10 cost, but worth it*
*semantics is a verb: helps keep it simple*
what I learnt from this project

what worked

*drive research by case studies*
*robust tools: x10 cost, but worth it*
*semantics is a verb: helps keep it simple*

what didn’t work

*language design is a hard sell*
*software engineering is a really hard sell*
*easy to get new idea funded once popular*
what I learnt from this project

what worked

drive research by case studies

robust tools: x10 cost, but worth it

semantics is a verb: helps keep it simple

what didn’t work

language design is a hard sell

software engineering is a really hard sell

easy to get new idea funded once popular

a mistake?

keeping Alloy general & pure

limited impact, except in teaching
for more info

http://sdg.csail.mit.edu

other fun things my group does

http://alloy.mit.edu

community, downloads, tutorial

http://softwareabstractions.org

sample chapters, models
acknowledgments

content of this talk especially
Alloy 4: Felix Chang
Kodkod engine: Emina Torlak
code analysis: Greg Dennis, Mana Taghdiri, Mandana Vaziri
Alloy 3: Ilya Shlyakhter, Manu Sridharan

not shown in photos: Sarfraz Khurshid, Mandana Vaziri, Manu Sridharan, Ilya Shlyakhter