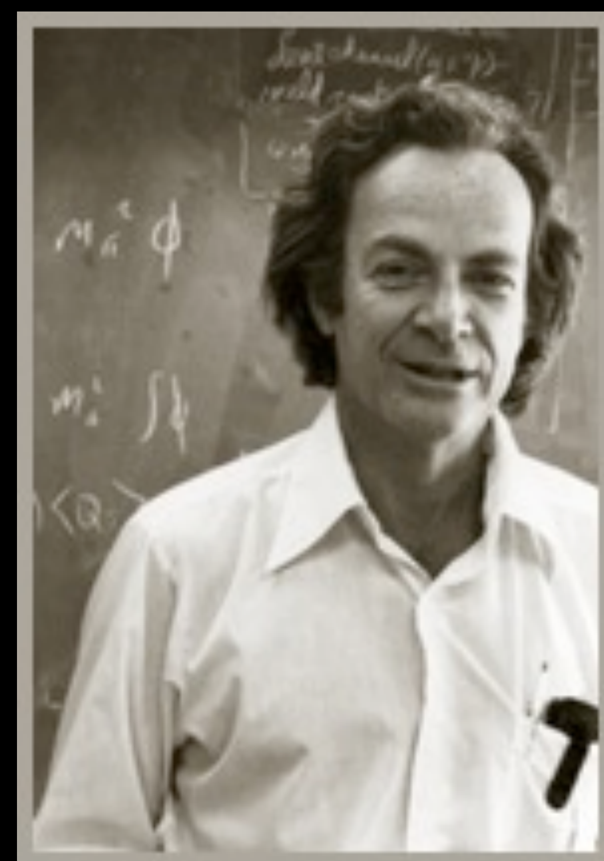
Alloy & Applications

University of Wisconsin, Madison · November 4, 2009

Daniel Jackson, MIT

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	*agile methods*
formalize design *Z, VDM, B, ...* *not code* exercise design *write proofs* *very painful*	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	*agile methods*
formalize design	formalize design
Z, VDM, B, ...	*Java, ML, Ruby, C, ...*
not code	
exercise design	exercise design
write proofs	
very painful	

early formalization

formal methods	*agile methods*
formalize design	formalize design
Z, VDM, B, ...	*Java, ML, Ruby, C, ...*
not code	
exercise design	exercise design
write proofs	*write tests*
very painful	

early formalization

formal methods	*agile methods*
formalize design	formalize design
Z, VDM, B, ...	*Java, ML, Ruby, C, ...*
not code	*verbose*
exercise design	exercise design
write proofs	*write tests*
very painful	*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

transantlantic 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

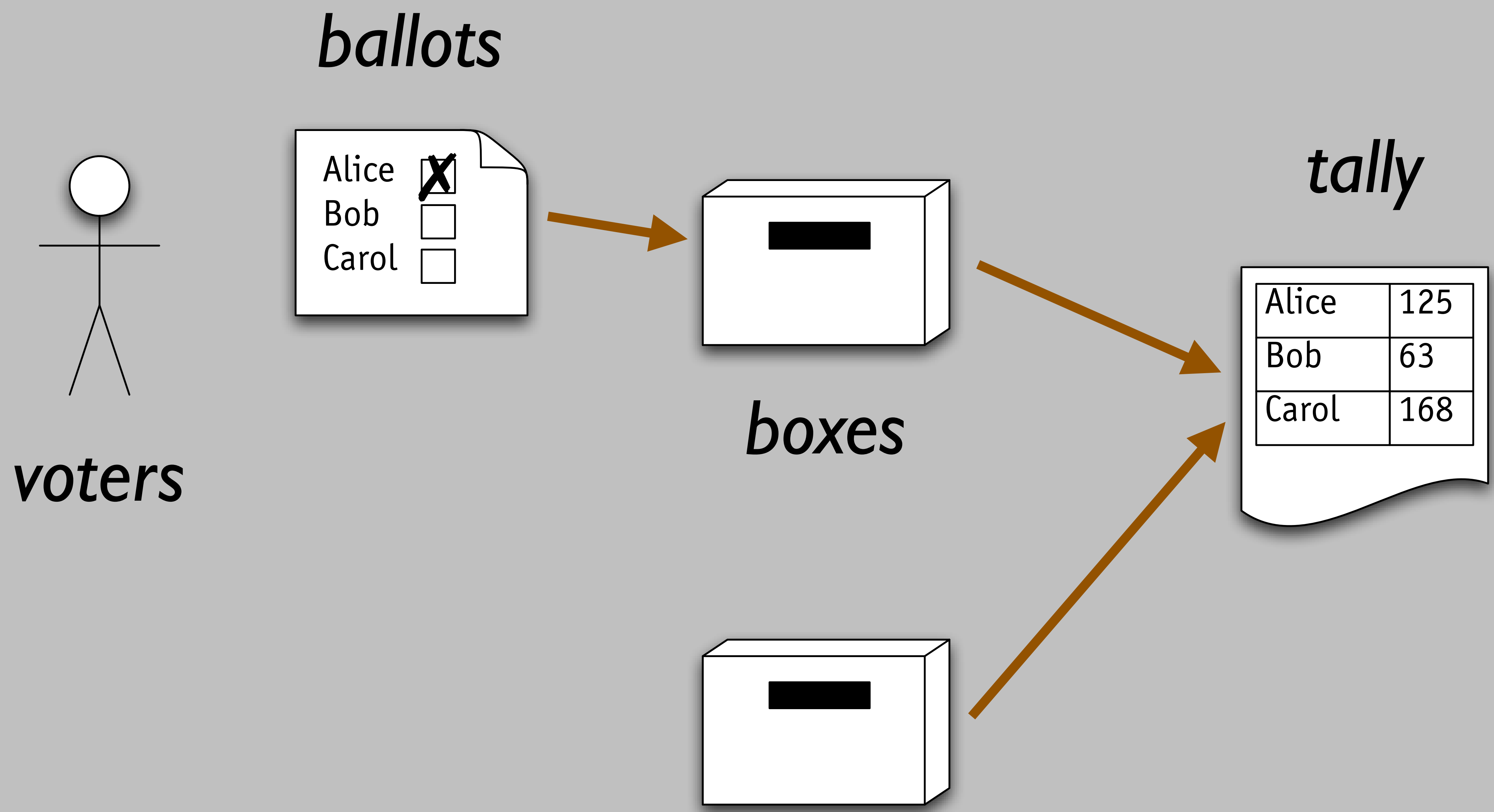
Pittsburgh, home of SMV

how we got here

version	*language*	*analysis*	*sample case study*
Nitpick (1995)	relational calculus subset of Z	relation enumeration	IPv6 routing
Alloy 1 (1999)	+ navigation exps quantifiers	WalkSAT, DP	intentional naming
Alloy 2 (2001)	+ relational ops higher arity	Chaff, Berkmin symmetry, sharing	key management, Unison filesync
Alloy 3 (2004)	+ subtyping, overloading	atomization (bad)	Mondex electronic purse
Alloy 4 (2007)	+ meta, sequences, arithmetic	sparse matrices better sharing	flash file systems, code analyses

example

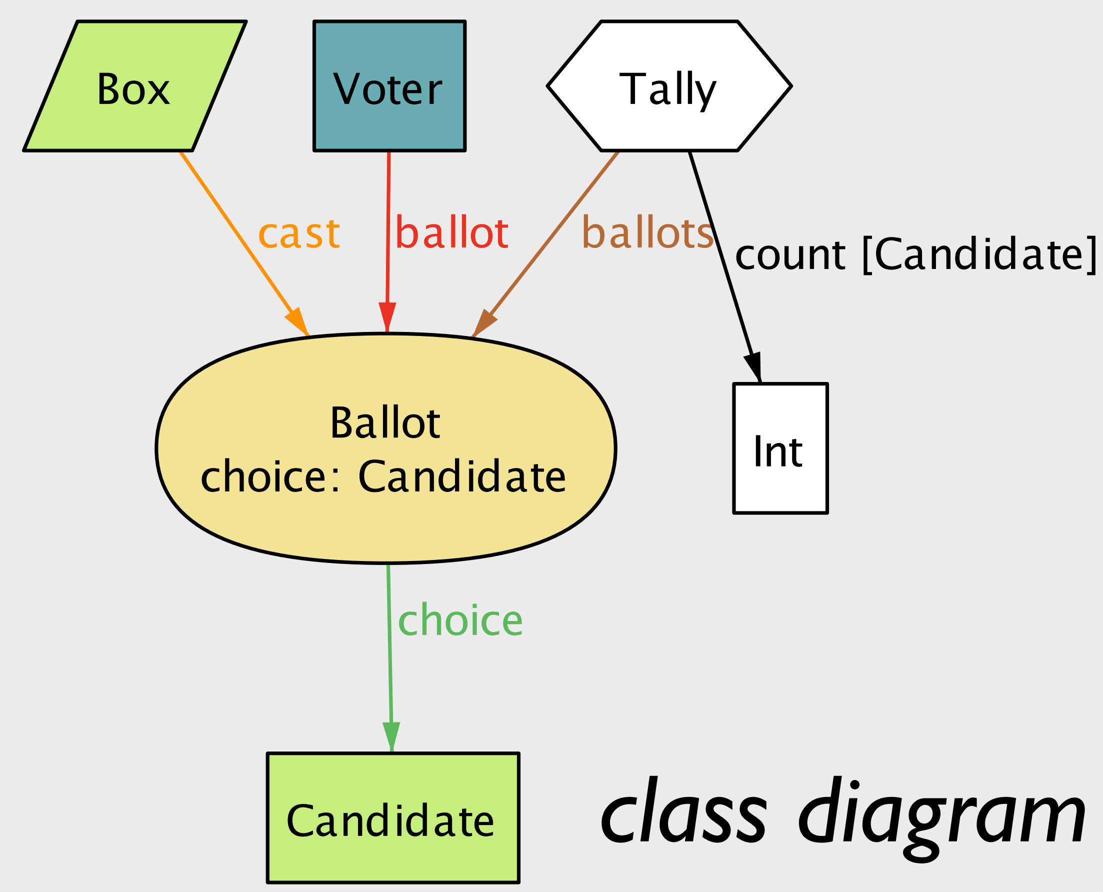
simplified voting



tally should correspond to voter's choices
what properties establish this requirement?

basic structure

```
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)
        }
```



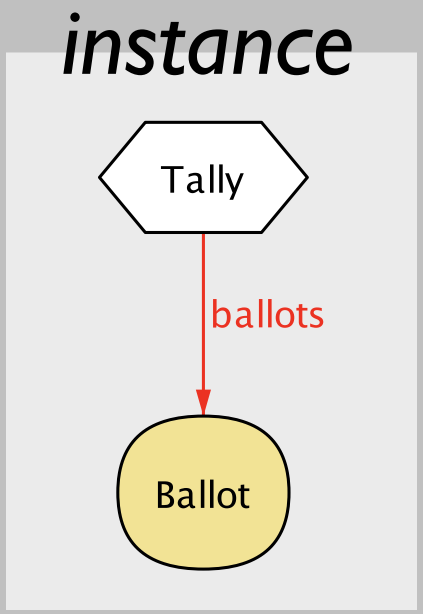
class diagram

animation

animation

run { }

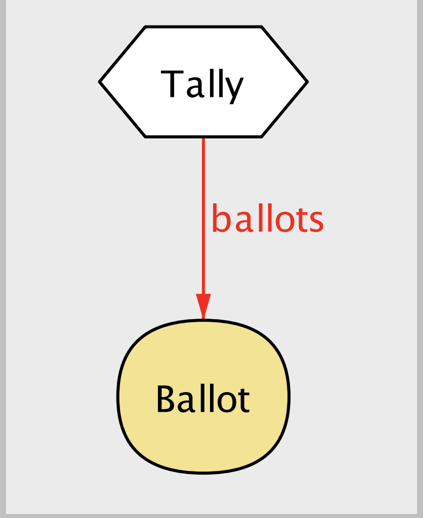
animation



run { }

animation

instance

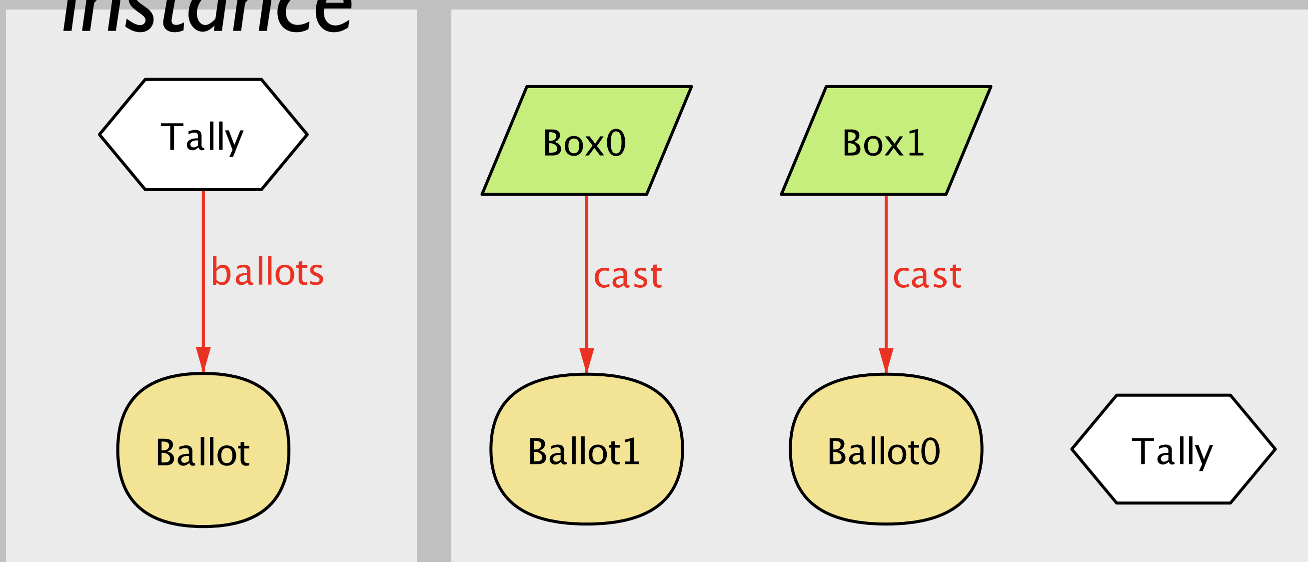


run { }

run { **some** cast }

animation

instance

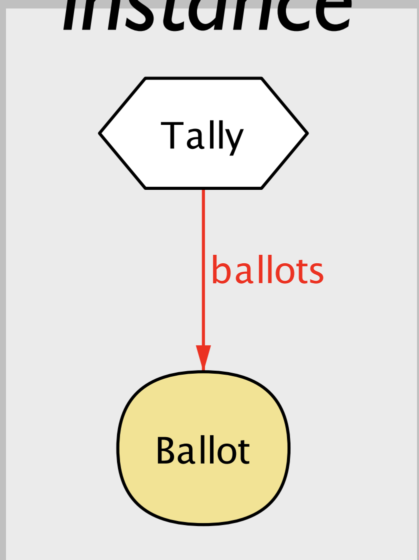


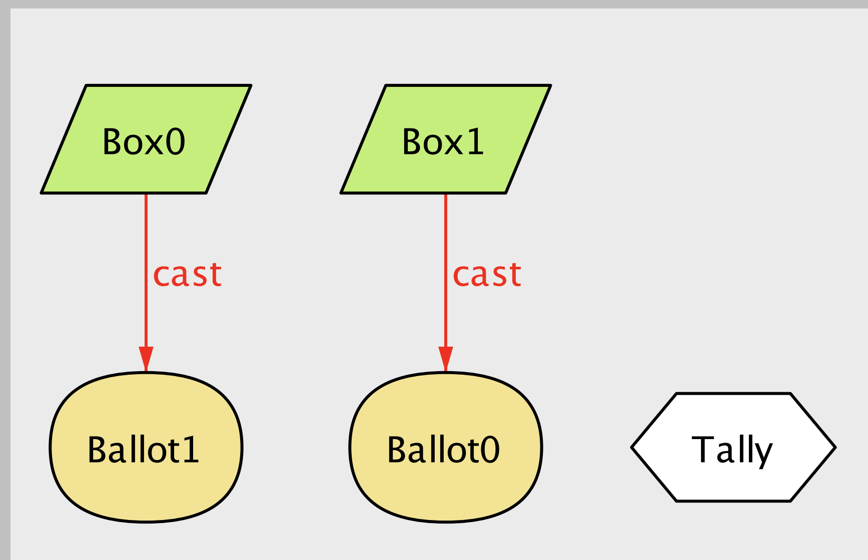
run { }

run { **some** cast }

animation

instance





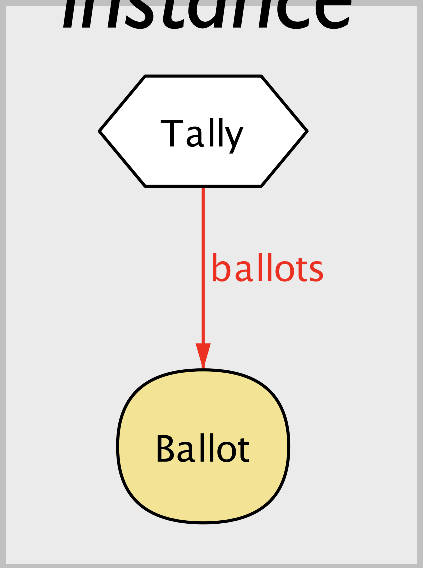
run { }

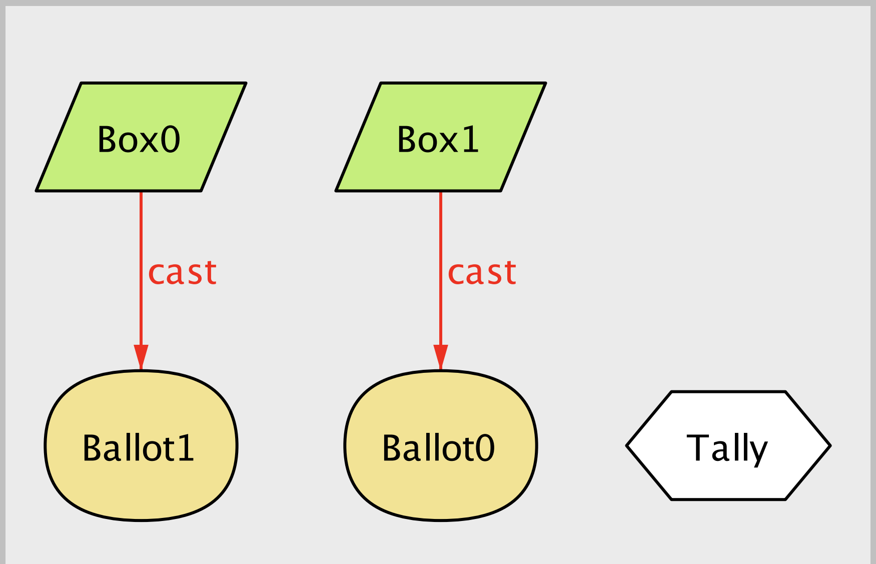
run { **some** cast }

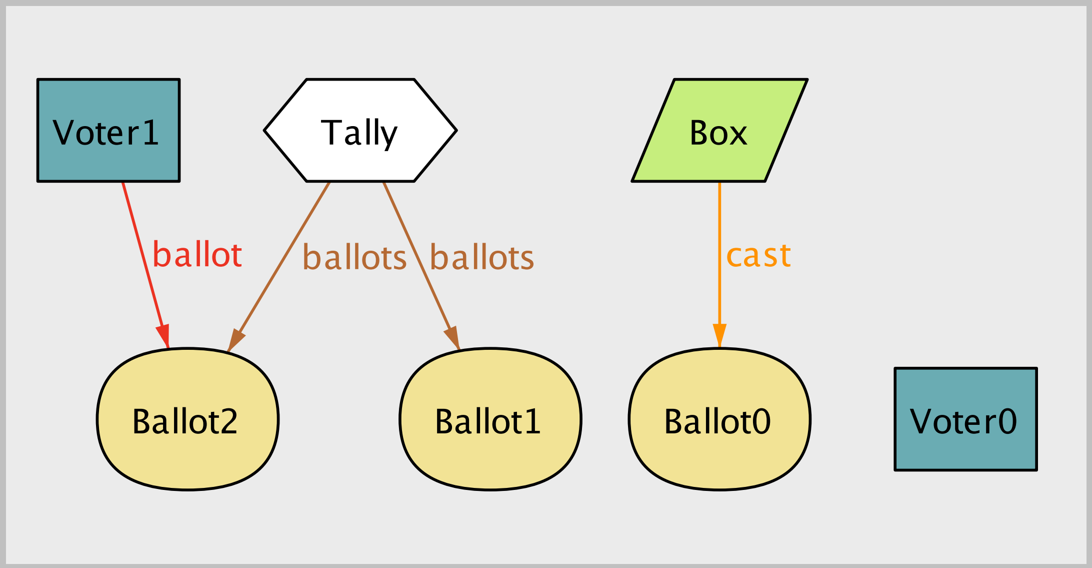
… *show next*

animation

instance







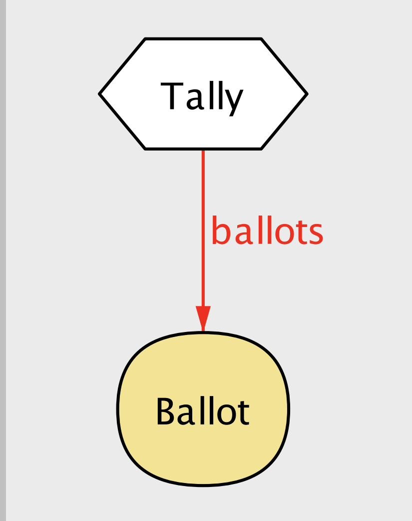
run { }

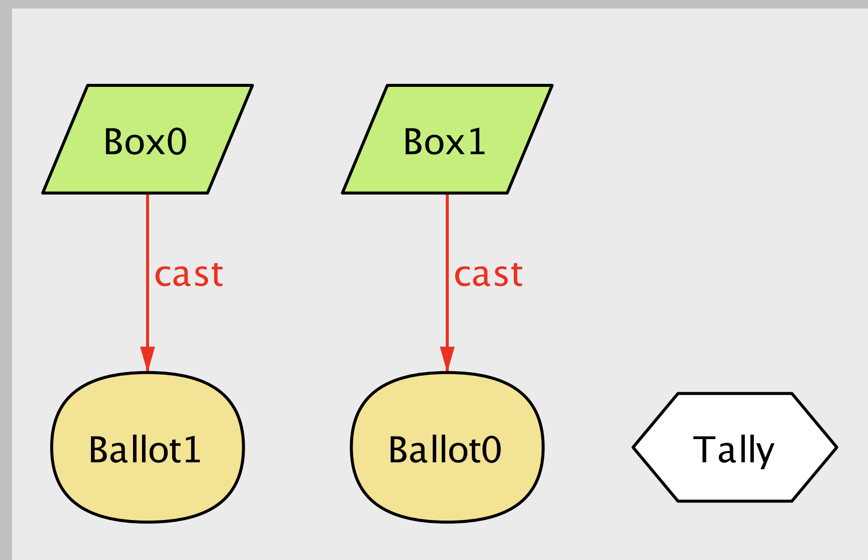
run { **some** cast }

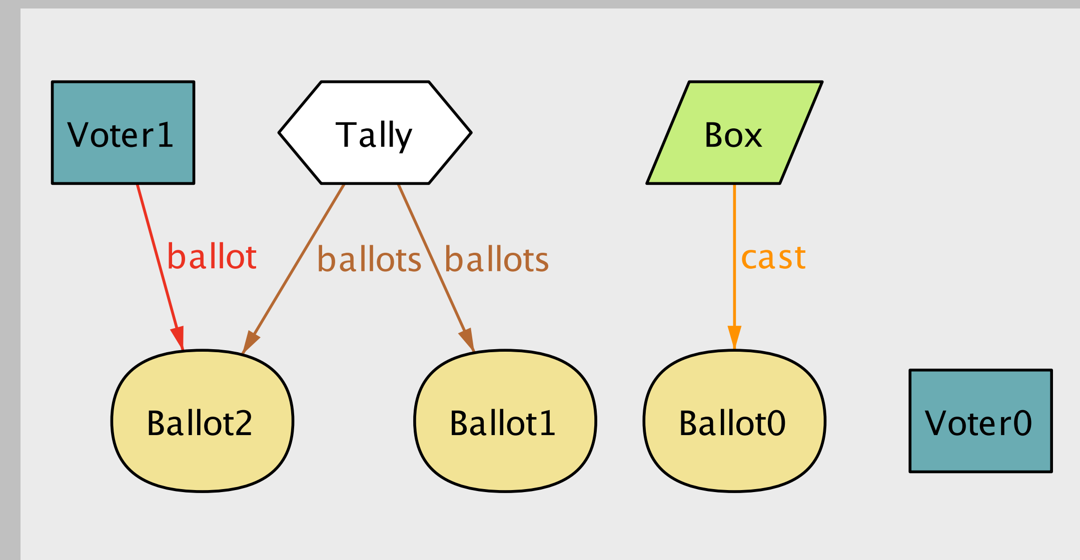
… *show next*

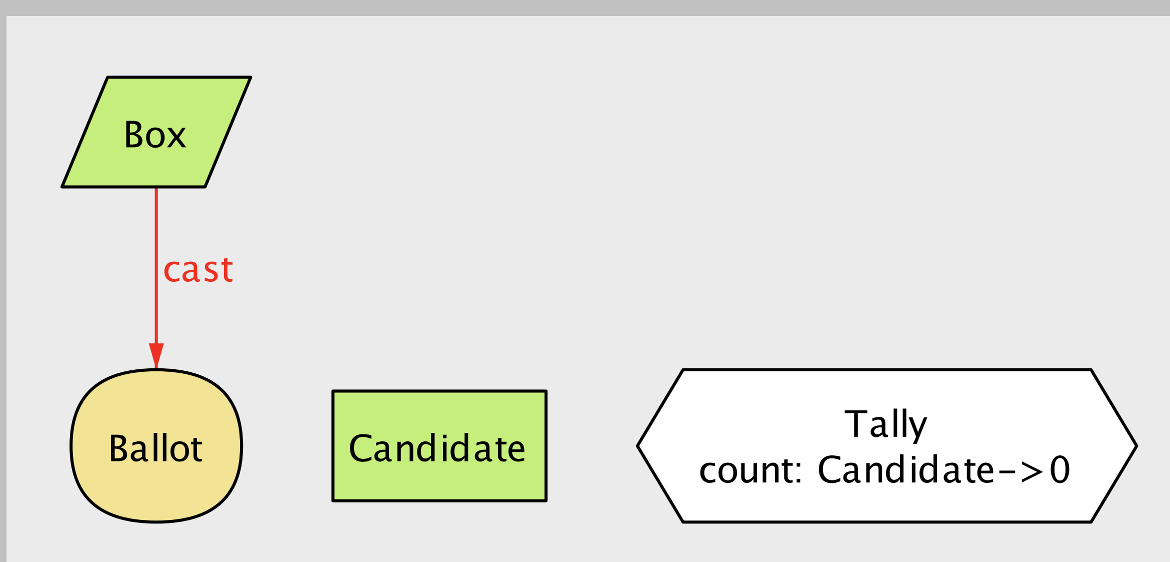
animation

instance









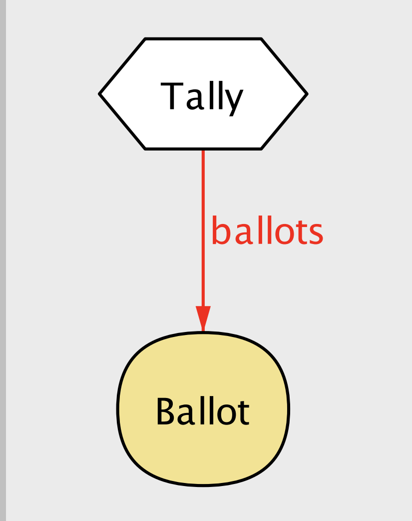
run { }

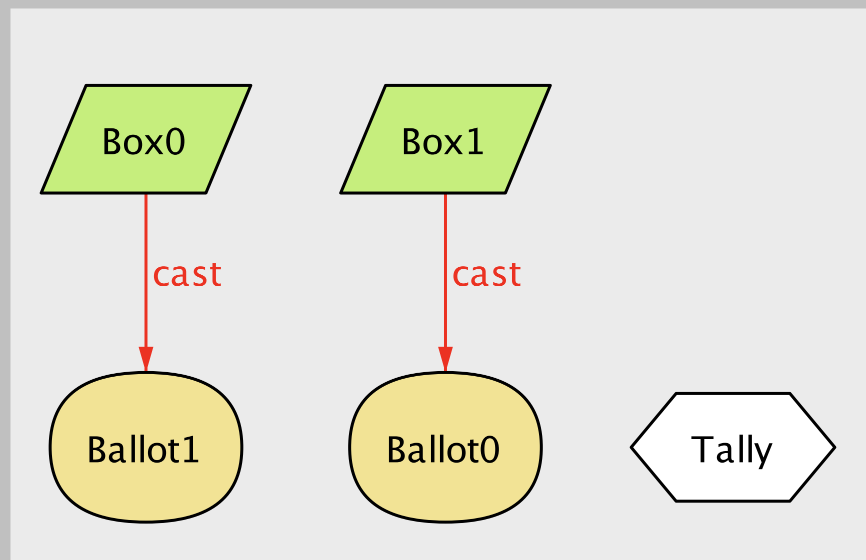
run { **some** cast }

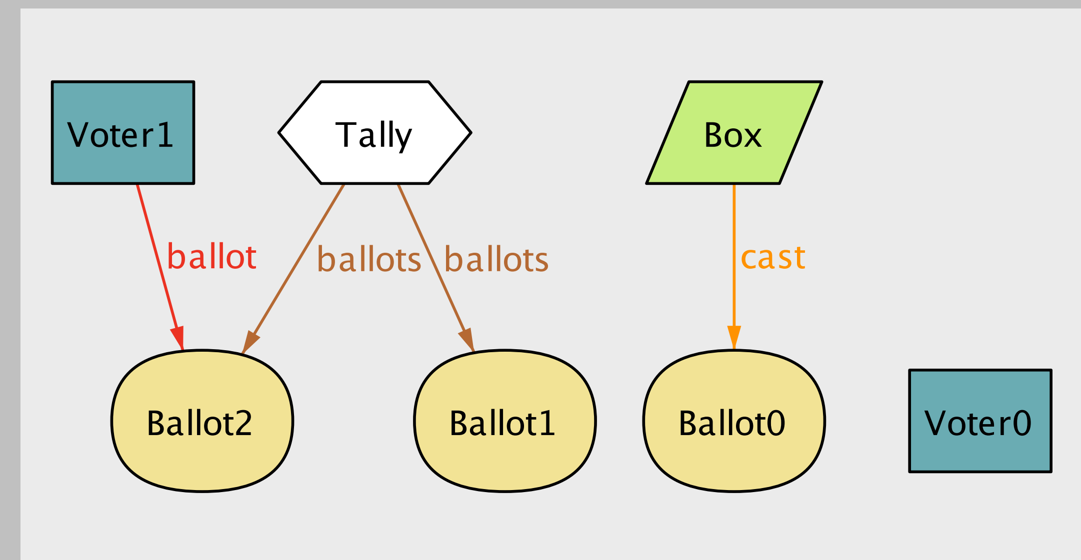
… *show next*

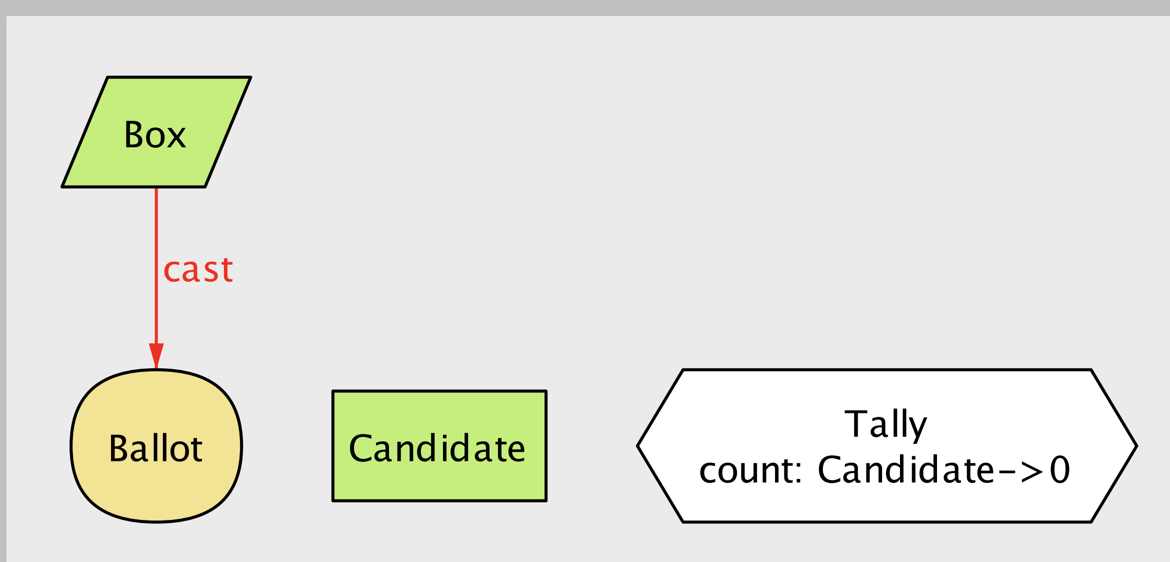
animation

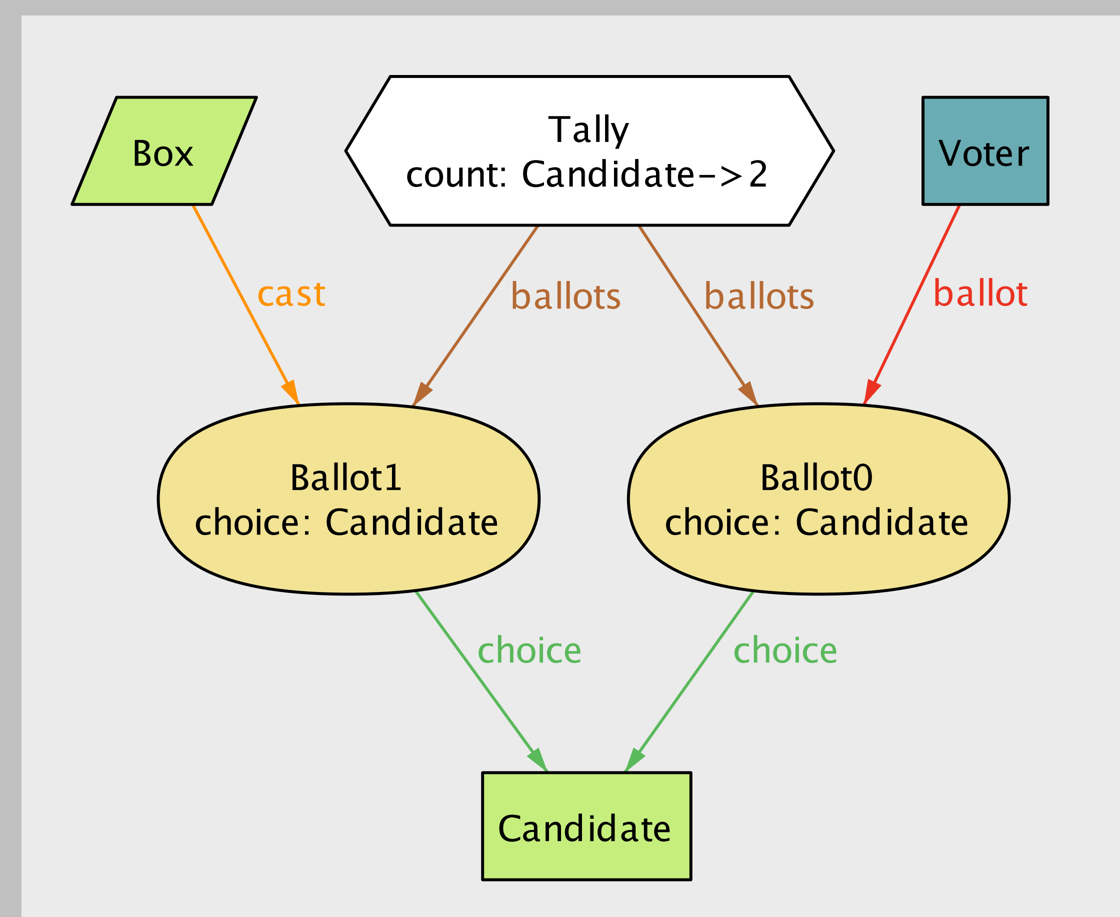
instance











run { }

run { **some** cast }

… *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) }
```

```
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 }
```

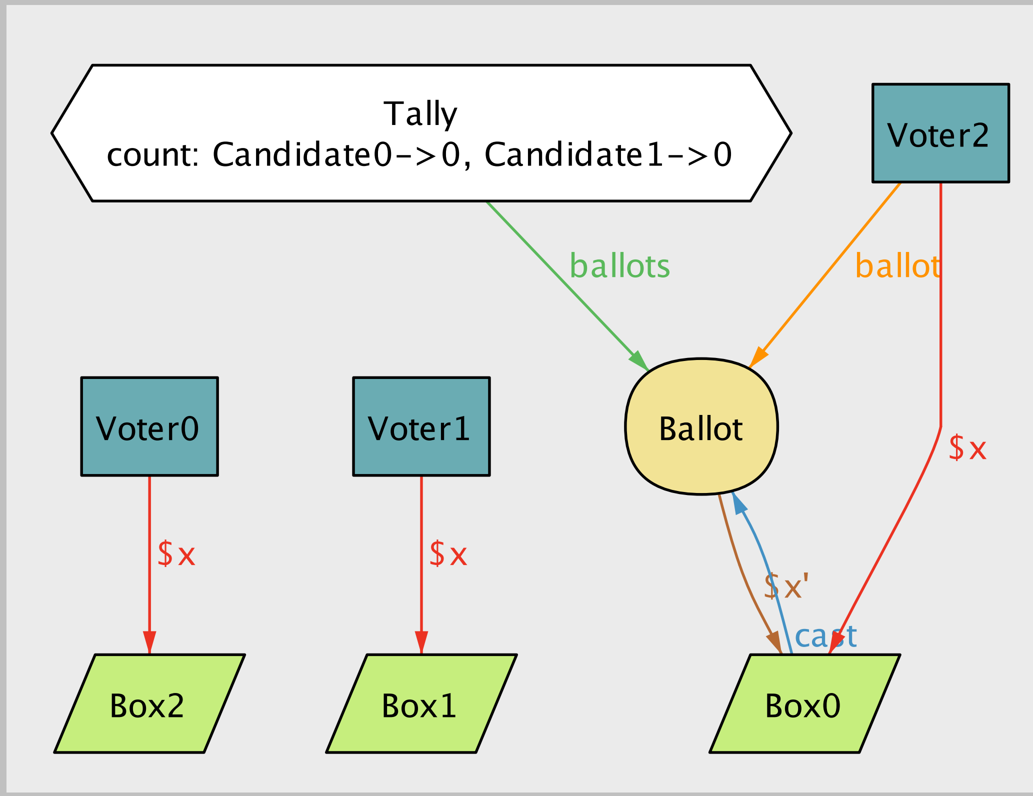
adding facts

```
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 }
```

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 { }
```

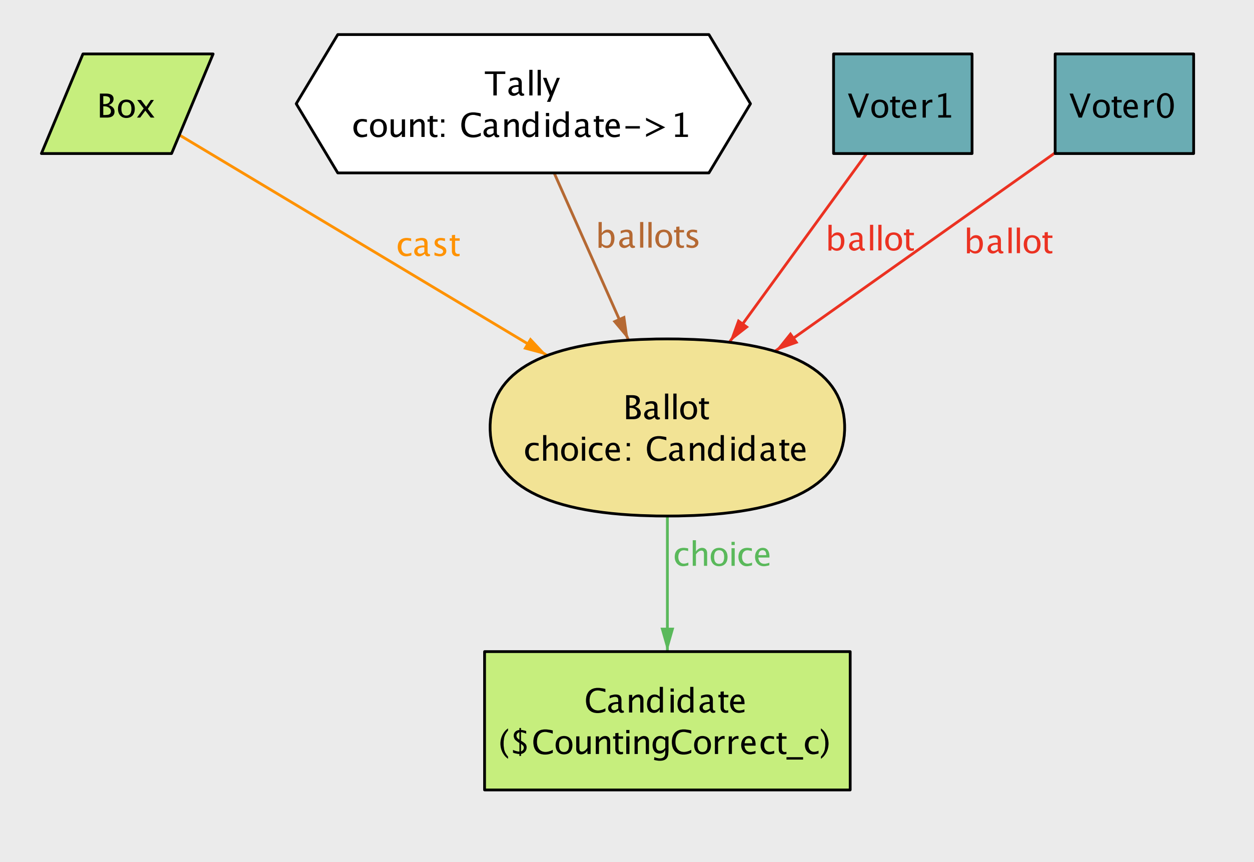
checking an assertion

```
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 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 }
```

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
```

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
```

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 }
```

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

```
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.
```

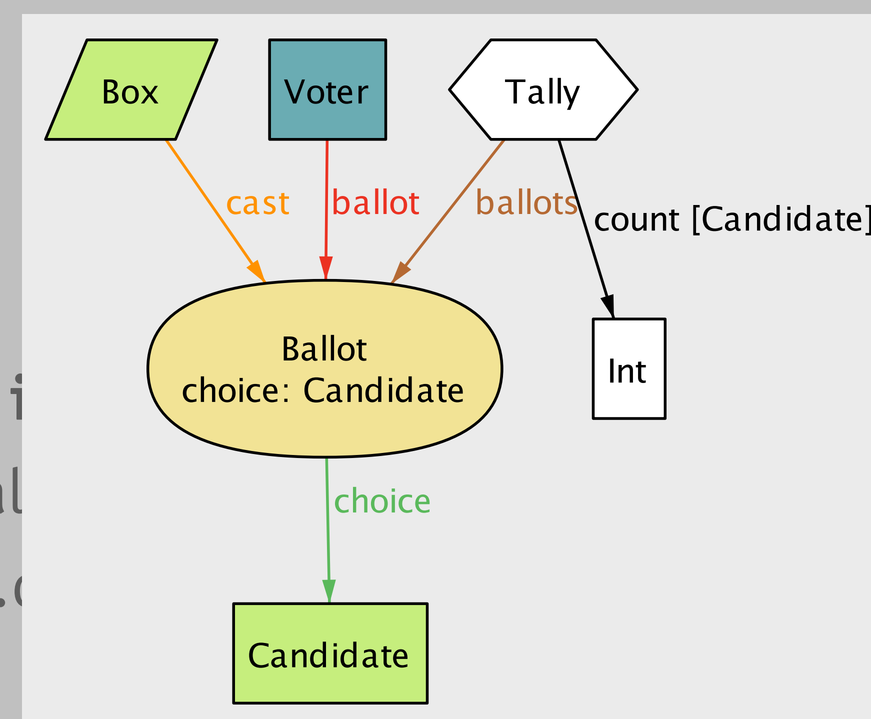
```
sig Car
sig Vot
sig Bal
sig Box
one sig Tally {
	ballots: set Ballot,
	count: Candidate -> one Int }
	{
	all c: Candidate | count[c] = #(ballots & choice.c)
	}
fact AllVotersCastBallots { all v: Voter | some x: Box | v.ballot in x.cast }
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.ballot = 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 Car
sig Vot
sig Bal
sig Box
```

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.

```
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 i
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.bal
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.c
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```



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

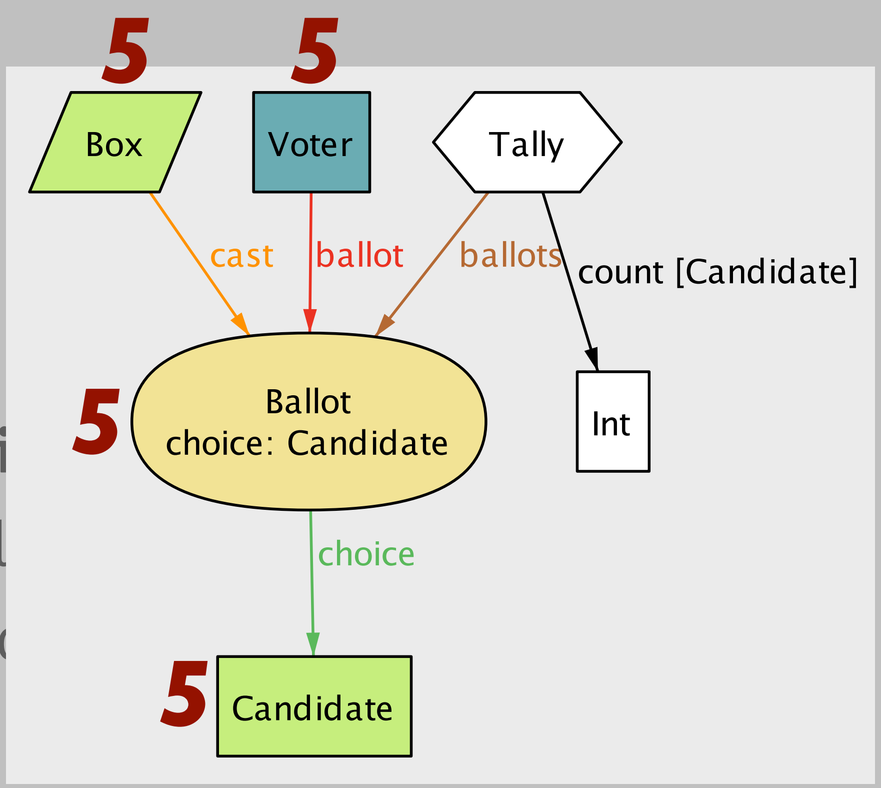
check CountingCorrect **for** 5

fixing & rechecking

```
sig Car
sig Vot
sig Bal
sig Box
```

```
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.
```

```
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 i
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.bal
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.c
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```



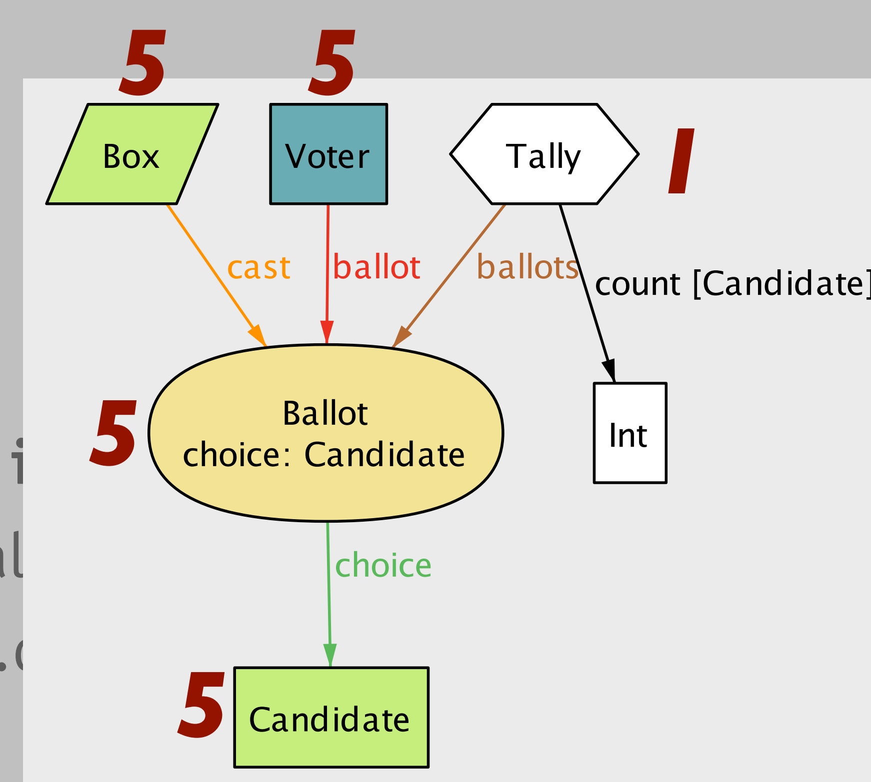

```
fact NoSharedBallots { all b: Ballot | lone ballot.b }
check CountingCorrect for 5
```

fixing & rechecking

```
sig Car
sig Vot
sig Bal
sig Box
```

```
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.
```

```
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 i
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.bal
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.c
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```

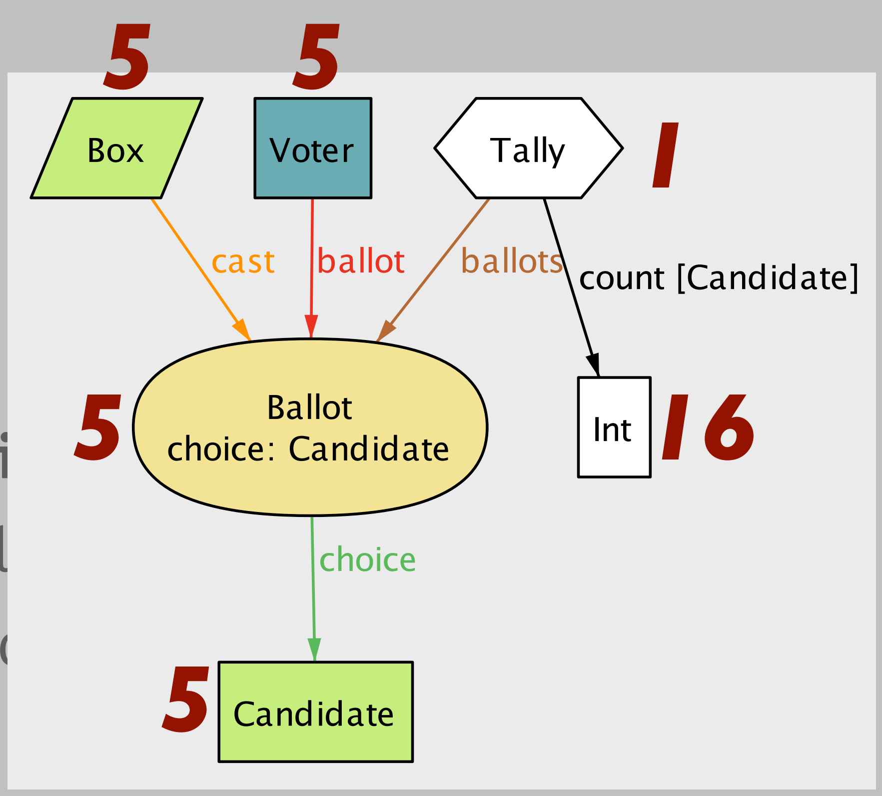



```
fact NoSharedBallots { all b: Ballot | lone ballot.b }
check CountingCorrect for 5
```

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 Car
sig Vot
sig Bal
sig Bo
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 i
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.bal
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```



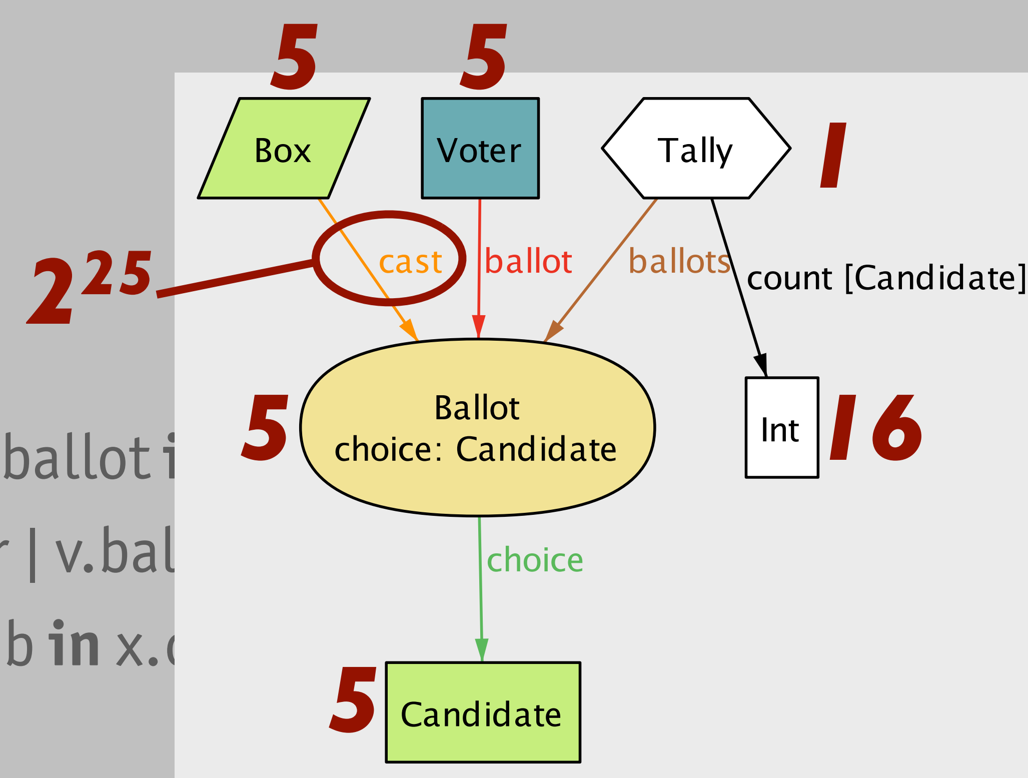
fact NoSharedBallots { **all** b: Ballot | **lone** ballot.b }

check CountingCorrect **for** 5

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 Car
sig Vot
sig Bal
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 i
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.bal
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.c
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```



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

check CountingCorrect **for** 5

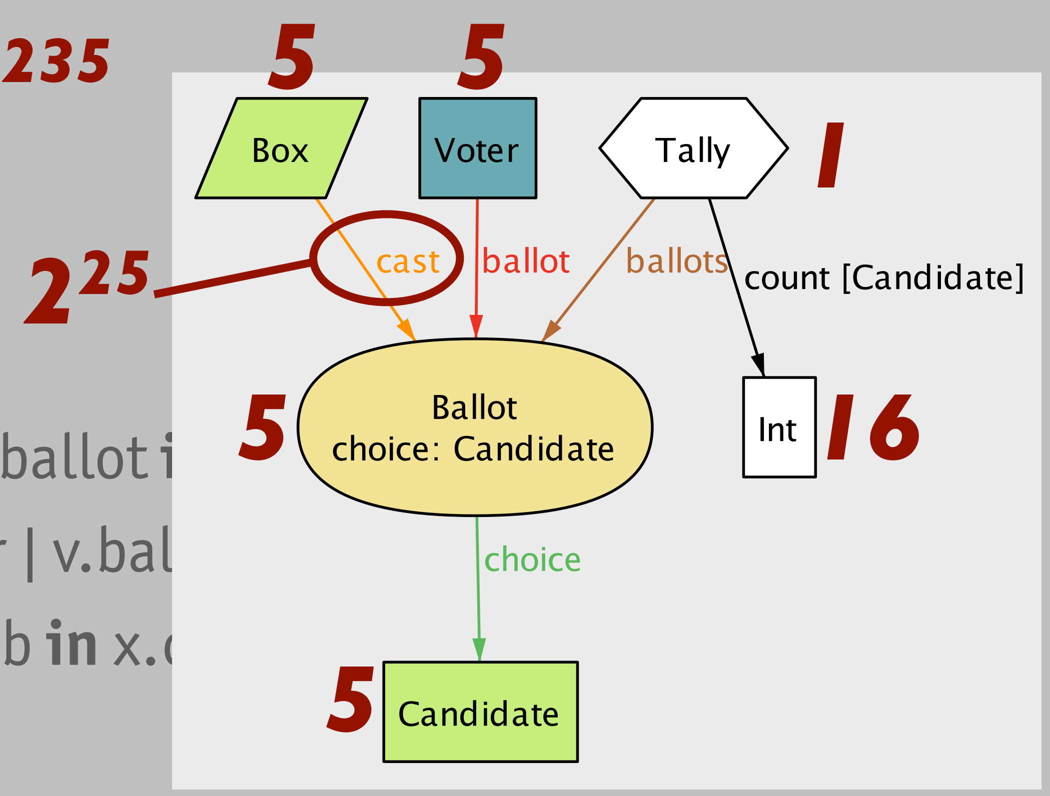
fixing & rechecking

sig Car
sig Vot
sig Bal
sig Box

```
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.
```

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

total space: 2^{235}



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

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

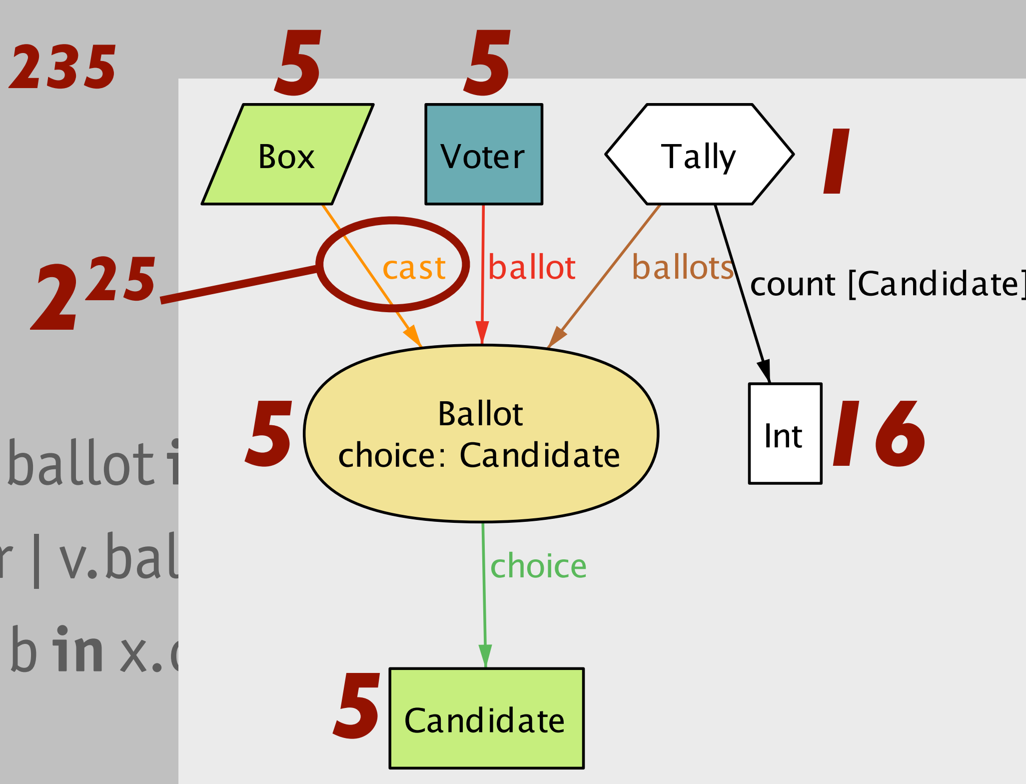
check CountingCorrect **for** 5

fixing & rechecking

```
sig Car
sig Vot
sig Bal
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 i
fact NoBoxStuffing { all x: Box, b: x.cast | some v: Voter | v.bal
fact NoTallyStuffing { all b: Tally.ballots | some x: Box | b in x.c
fact NoBallotsLost { all x: Box | x.cast in Tally.ballots }
```

```
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.
```

total space: 2^{235}




```
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?

$S(beh) \wedge \neg P(beh)$

idea: analysis = solving

does system S have property P?

$$S(beh) \wedge \neg P(beh)$$

is operation *op* deterministic?

$$op(s,s') \wedge op(s,s'') \wedge s' \neq s''$$

idea: analysis = solving

does system S have property P?

$S(beh) \wedge \neg P(beh)$

is operation *op* deterministic?

$op(s,s') \wedge op(s,s'') \wedge s' \neq s''$

does method *m* meet *pre*, *post*?

$m(s,s') \wedge pre(s) \wedge \neg post(s,s')$

idea: analysis = solving

does system S have property P?

$S(beh) \wedge \neg P(beh)$

is operation *op* deterministic?

$op(s,s') \wedge op(s,s'') \wedge s' \neq s''$

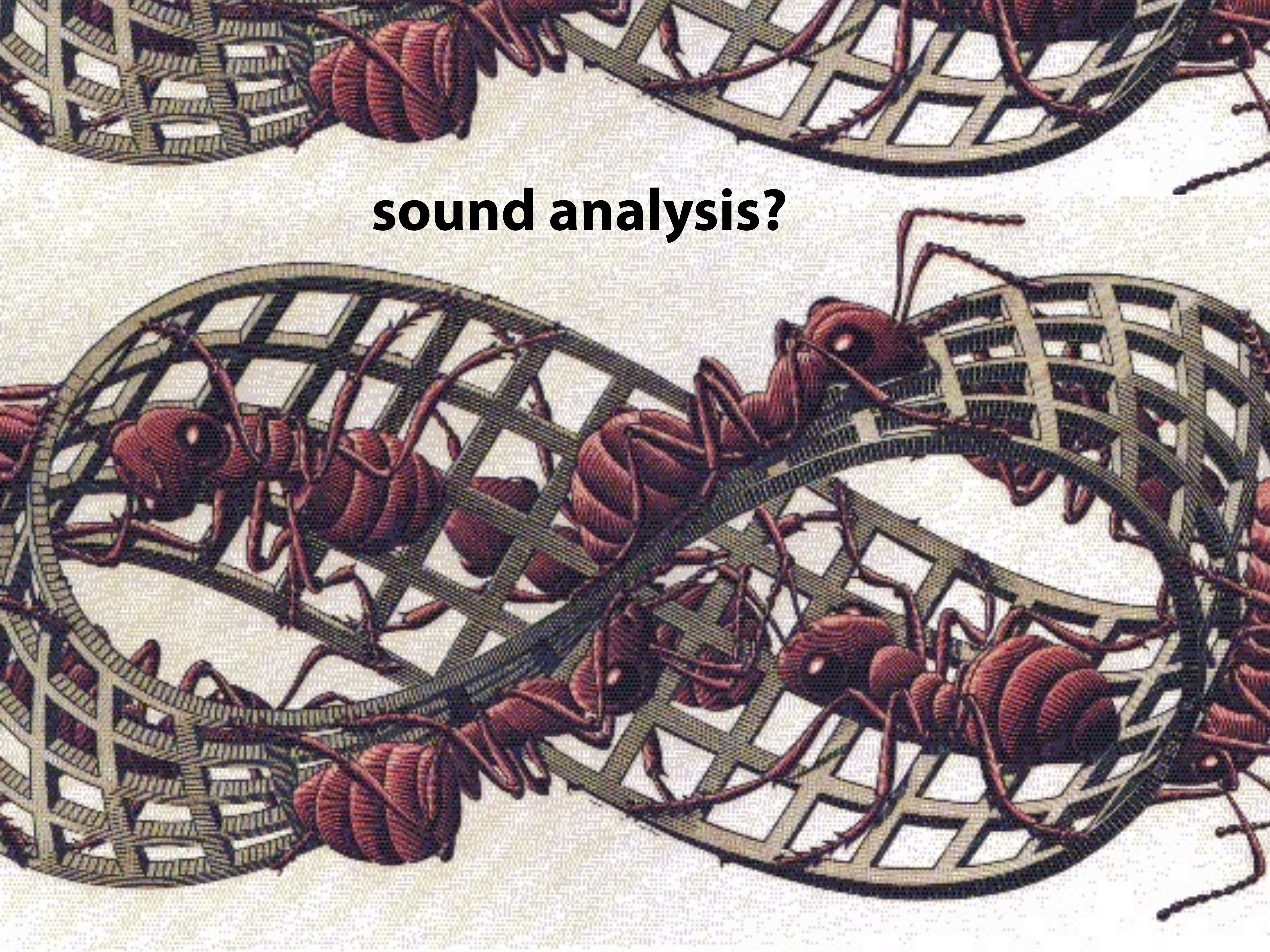
does method *m* meet *pre*, *post*?

$m(s,s') \wedge pre(s) \wedge \neg post(s,s')$

is *op'* a valid refactoring of *op*?

$op'(s,s') \wedge \neg op(s,s')$

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*

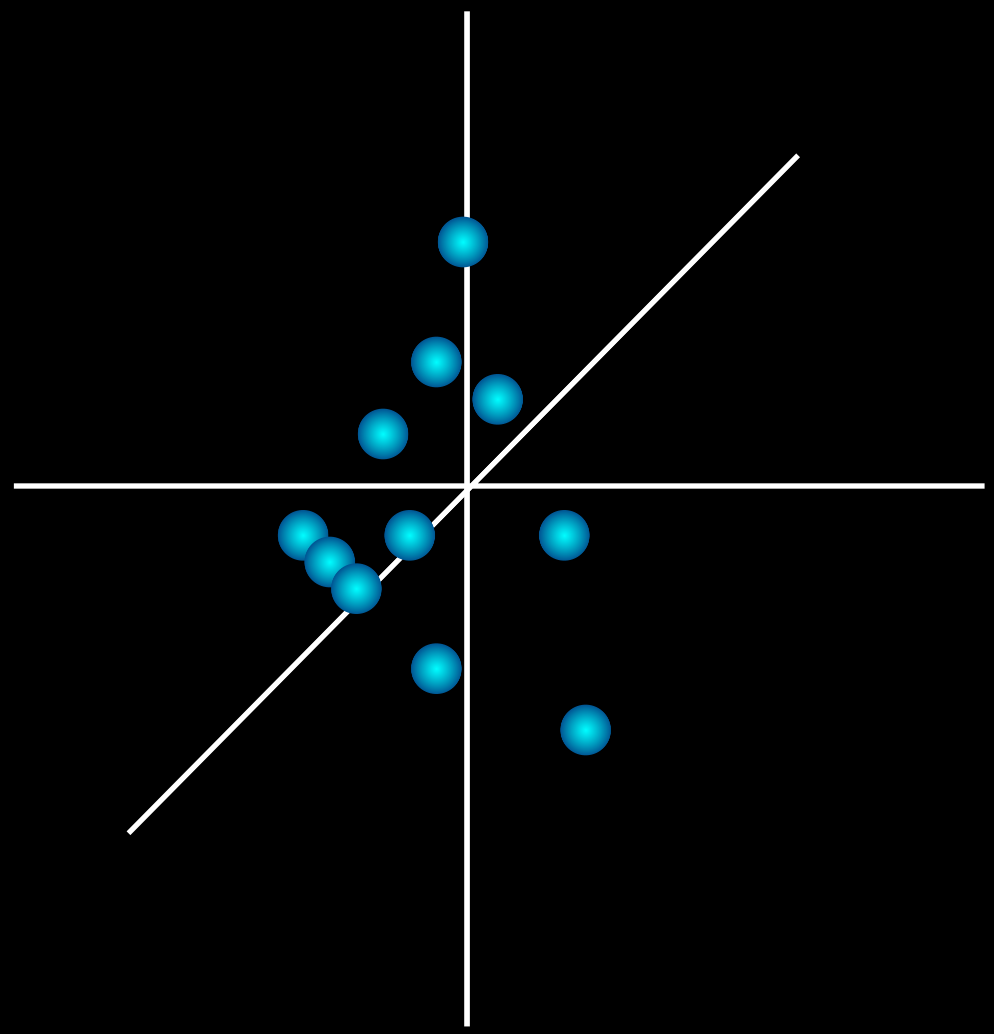
and not a guarantee anyway?

even theorem proving isn’t watertight

inconsistent axioms may undermine proof

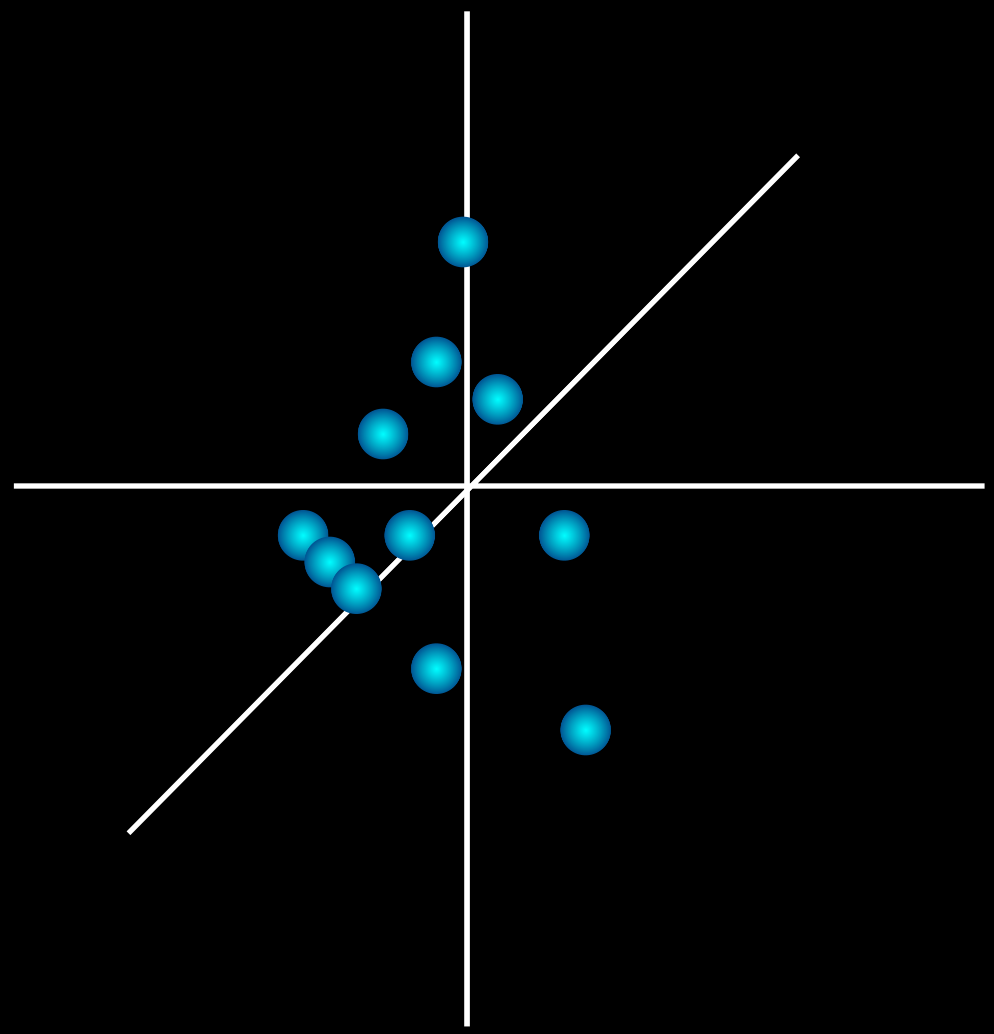
idea: small scope analysis

idea: small scope analysis

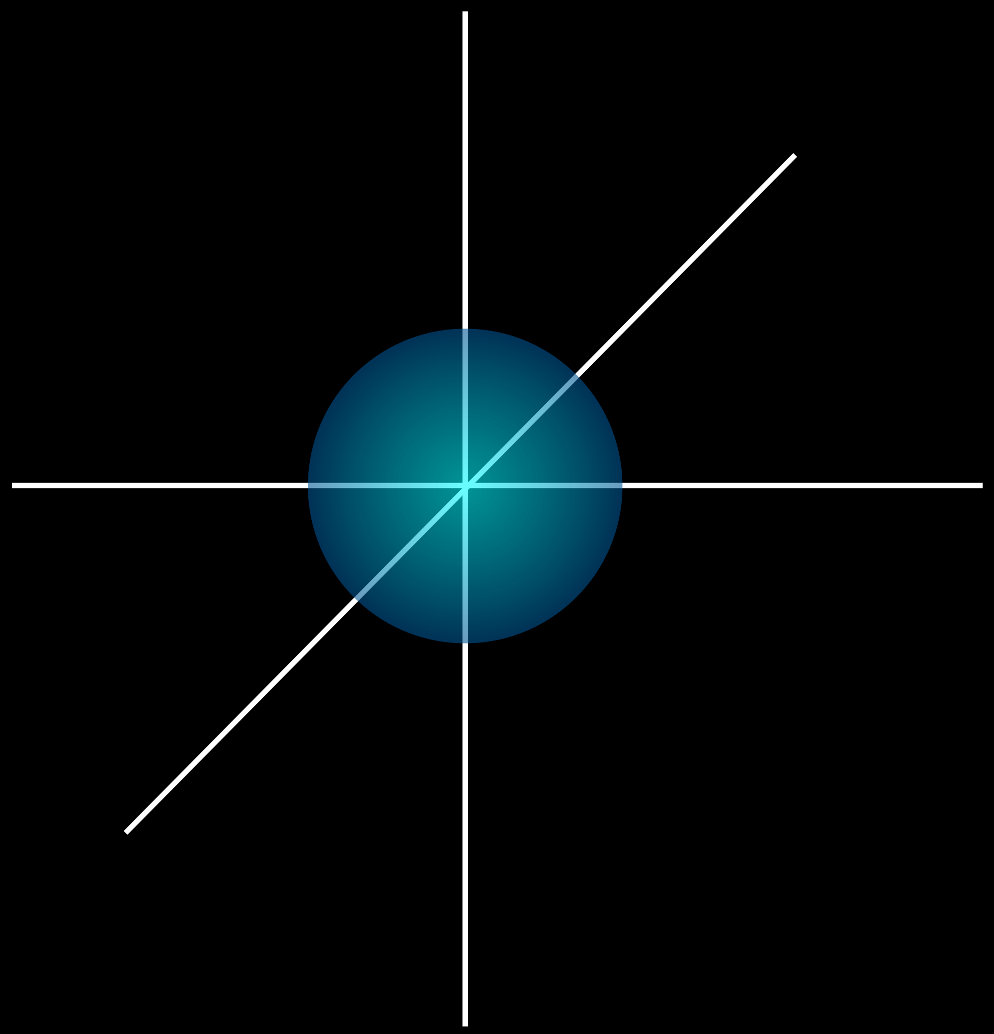


random testing:
a few big tests

idea: small scope analysis



random testing:
a few big tests



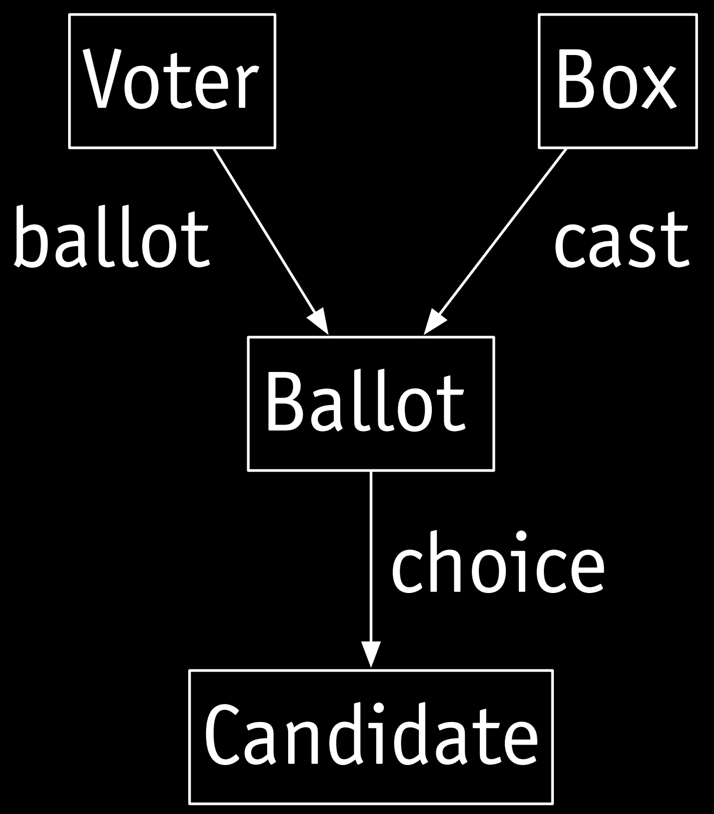
small scope analysis:
'all small tests'

idea: generalizing dot

x.f

idea: generalizing dot

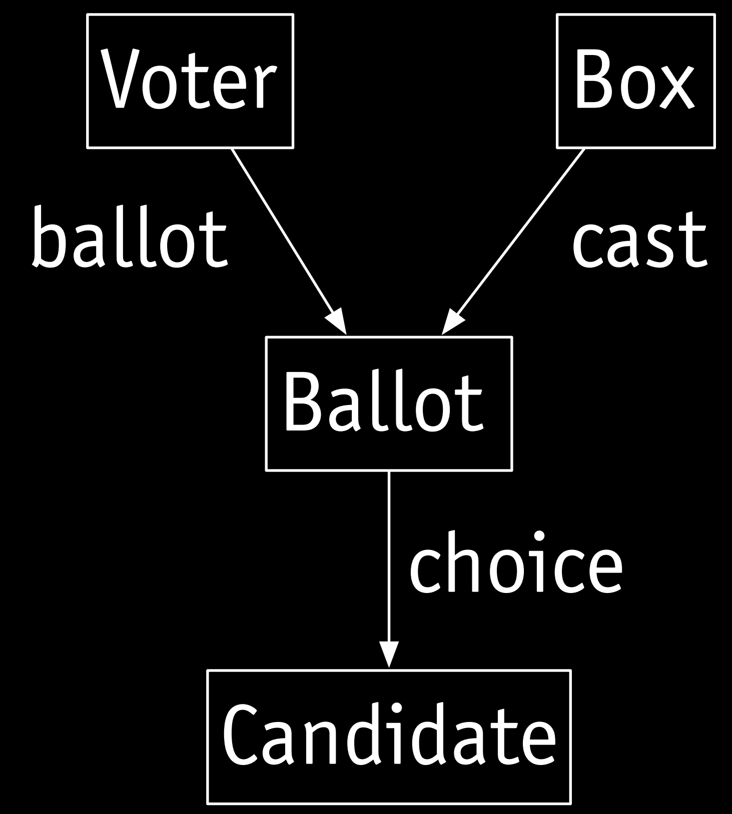
x.f



idea: generalizing dot

x.f

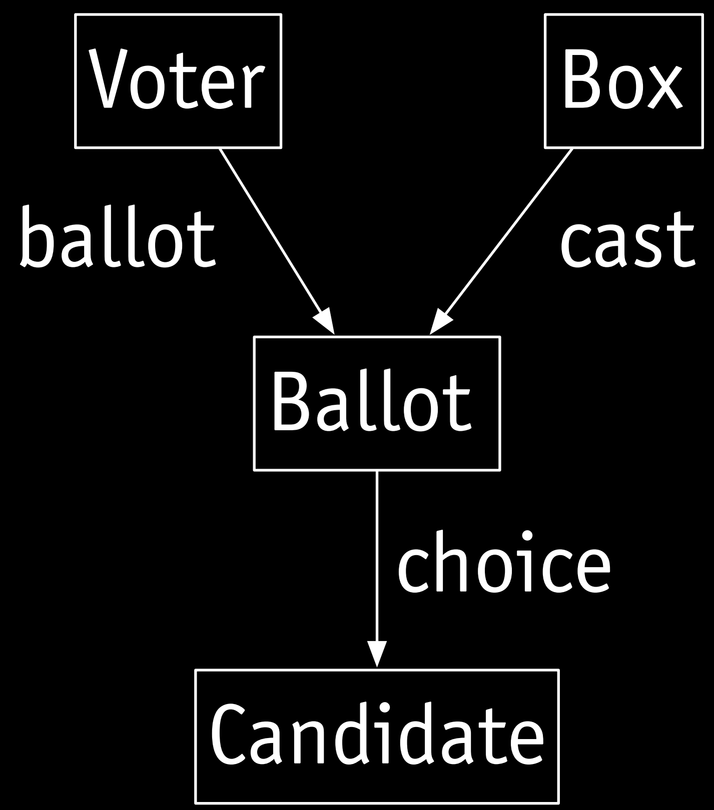
v.ballot



idea: generalizing dot

x.f

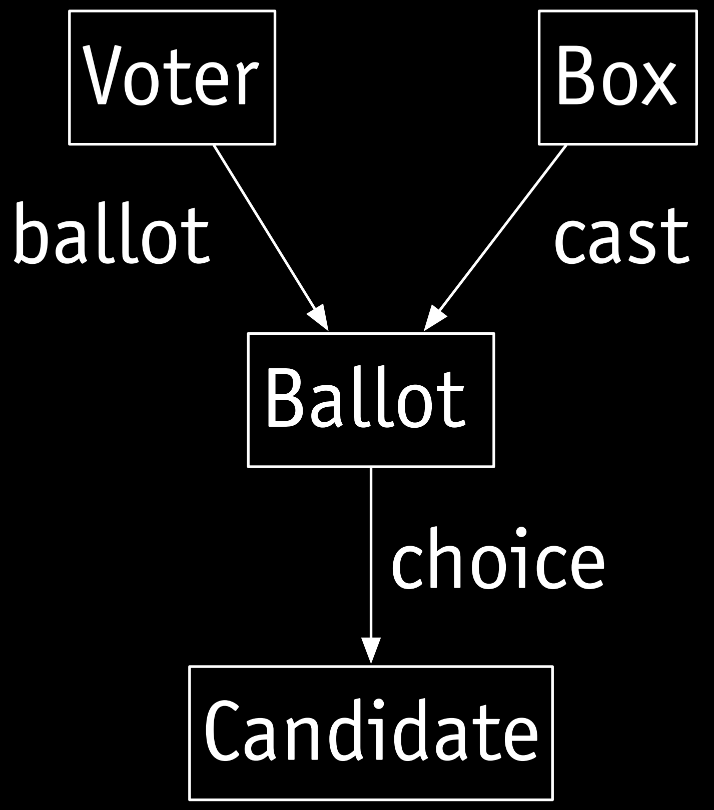
v.ballot
v.ballot.choice



idea: generalizing dot

x.f

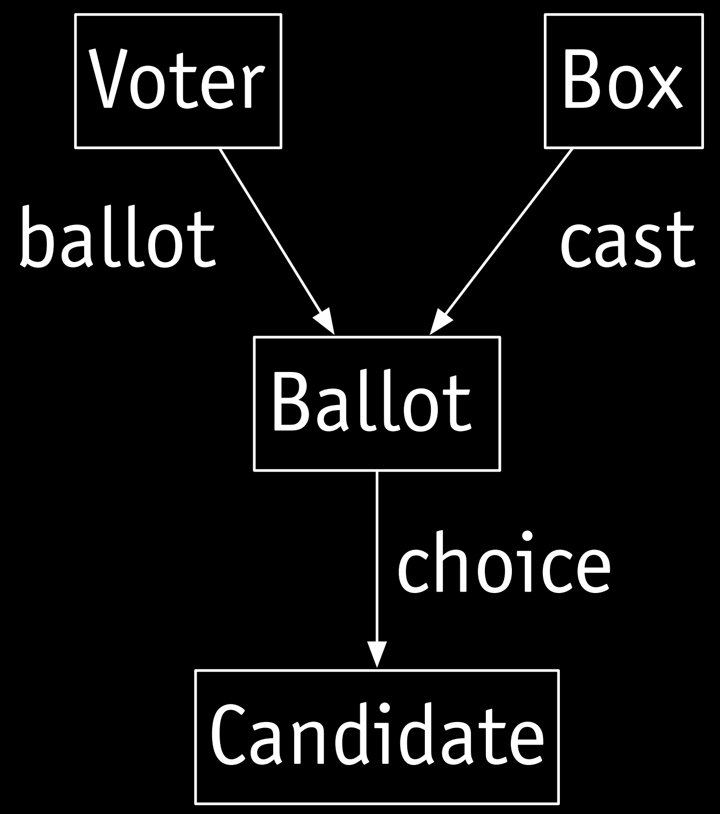
v.ballot
v.ballot.choice
b.cast



idea: generalizing dot

x.f

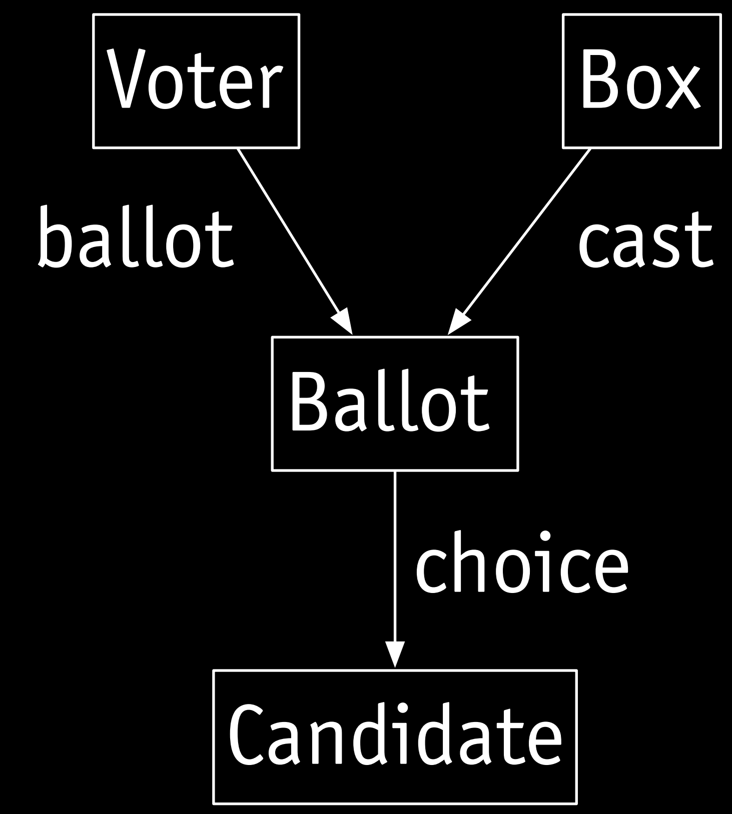
v.ballot
v.ballot.choice
b.cast
b.cast.choice



idea: generalizing dot

x.f

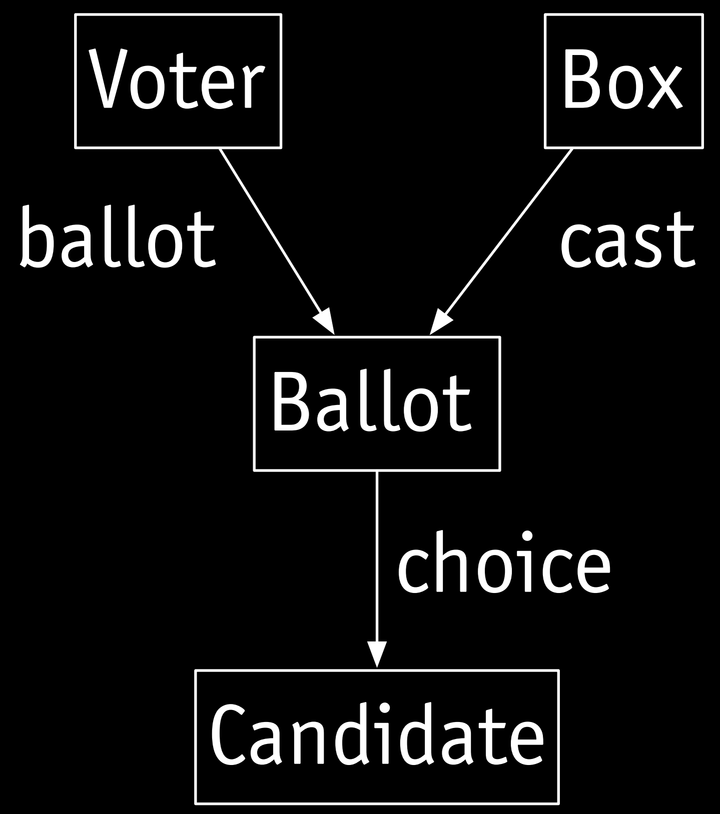
v.ballot
v.ballot.choice
b.cast
b.cast.choice
choice.c



idea: generalizing dot

x.f

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



a definition

generalized join

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

a definition

generalized join

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

v.ballot.choice

a definition

generalized join

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

v.ballot.choice

V0

a definition

generalized join

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

v.ballot.choice

V0

V0	B0
V1	B2
V2	B1

a definition

generalized join

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

v.ballot.choice

V0

V0	B0
V1	B2
V2	B1

a definition

generalized join

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

v.ballot.choice

V0

V0	B0
V1	B2
V2	B1

B0

a definition

generalized join

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

v.ballot.choice

B0

a definition

generalized join

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

v.ballot.choice

B0	C0
B1	C1
B2	C0

B0

a definition

generalized join

$$a.b = \{\ (a_0,\ldots,\ a_{n-1},\ b_1,\ldots,\ b_m)\ |\ (a_0,\ldots,\ a_n) \in a \wedge (a_n,\ b_1,\ldots,\ b_m) \in b\}$$

v.ballot.choice

B0	C0
B1	C1
B2	C0

B0

a definition

generalized join

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

v.ballot.choice

B0	C0
B1	C1
B2	C0

B0

C0

a definition

generalized join

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

v.ballot.choice

C0

right associating

generalized join

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

right associating

generalized join

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

v.ballot.choice

right associating

generalized join

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

v.ballot.choice

V0	B0
V1	B2
V2	B1

right associating

generalized join

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

v.ballot.choice

V0	B0
V1	B2
V2	B1

B0	C0
B1	C1
B2	C0

right associating

generalized join

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

v.*ballot*.*choice*

V0	B0
V1	B2
V2	B1

B0	C0
B1	C1
B2	C0

right associating

generalized join

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

v.ballot.choice

V0	B0
V1	B2
V2	B1

B0	C0
B1	C1
B2	C0

V0	C0
V1	C0
V2	C1

right associating

generalized join

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

v.ballot.choice

V0	C0
V1	C0
V2	C1

right associating

generalized join

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

v . *ballot* . *choice*

V0

V0	C0
V1	C0
V2	C1

right associating

generalized join

$$a.b = \{\ (a_0,\ldots,\ a_{n-1},\ b_1,\ldots,\ b_m)\ |\ (a_0,\ldots,\ a_n) \in a \wedge (a_n,\ b_1,\ldots,\ b_m) \in b\}$$

v . ballot . choice

V0

V0	C0
V1	C0
V2	C1

right associating

generalized join

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

v.ballot.choice

V0

V0	C0
V1	C0
V2	C1

C0

right associating

generalized join

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

v.ballot.choice

C0

navigating back

generalized join

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

ballot.choice.c

navigating back

generalized join

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

ballot.choice.c

C0

navigating back

generalized join

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

ballot.choice.c

B0	C0
B1	C1
B2	C0

C0

navigating back

generalized join

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

ballot.*choice*.*c*

B0	C0
B1	C1
B2	C0

C0

navigating back

generalized join

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

ballot . *choice* . *c*

B0	C0
B1	C1
B2	C0

C0

B0
B2

navigating back

generalized join

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

ballot.choice.c

B0
B2

navigating back

generalized join

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

ballot.choice.c

V0	B0
V1	B2
V2	B1

B0
B2

navigating back

generalized join

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

ballot.choice.c

V0	B0
V1	B2
V2	B1

B0
B2

navigating back

generalized join

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

ballot.choice.c

V0	B0
V1	B2
V2	B1

B0
B2

V0
V1

idea: solving by SAT

```
v.ballot = b
```

idea: **solving by SAT**

v.ballot = b

V0

idea: solving by SAT

```
v.ballot = b
```

V0	1
V1	0
V2	0

V0

idea: solving by SAT

```
v.ballot = b
```

V0	1
V1	0
V2	0

idea: solving by SAT

v.ballot = b

V0	1
V1	0
V2	0

B0

idea: solving by SAT

v.ballot = b

V0	1
V1	0
V2	0

B0	1
B1	0
B2	0

B0

idea: **solving by SAT**

```
v.ballot = b
```

V0	1
V1	0
V2	0

B0	1
B1	0
B2	0

idea: solving by SAT

```
v.ballot = b
```

V0	1
V1	0
V2	0

B0	1
B1	0
B2	0

V0	B0
V1	B2
V2	B1

idea: solving by SAT

`v.ballot = b`

V0	1
V1	0
V2	0

	V0	V1	V2
B0	1	0	0
B1	0	0	1
B2	0	1	0

B0	1
B1	0
B2	0

V0	B0
V1	B2
V2	B1

idea: solving by SAT

$$v.\text{ballot} = b$$

V0	1
V1	0
V2	0

	V0	V1	V2
B0	1	0	0
B1	0	0	1
B2	0	1	0

B0	1
B1	0
B2	0

idea: solving by SAT

v . ballot = b

V0 V1 V2

V0
V1
V2

B0
B1
B2

B0
B1
B2

idea: solving by SAT

v.ballot = b

V0 V1 V2

V0	v_0
V1	v_1
V2	v_2

B0
B1
B2

B0
B1
B2

idea: solving by SAT

v.ballot = b

V0 V1 V2

V0	v_0
V1	v_1
V2	v_2

B0
B1
B2

B0	b_0
B1	b_1
B2	b_2

idea: solving by SAT

$$v.ballot = b$$

V0	v_0
V1	v_1
V2	v_2

	V0	V1	V2
B0	f_{00}	f_{01}	f_{02}
B1	f_{10}	f_{11}	f_{12}
B2	f_{20}	f_{21}	f_{22}

B0	b_0
B1	b_1
B2	b_2

idea: solving by SAT

```
v.ballot = b
```

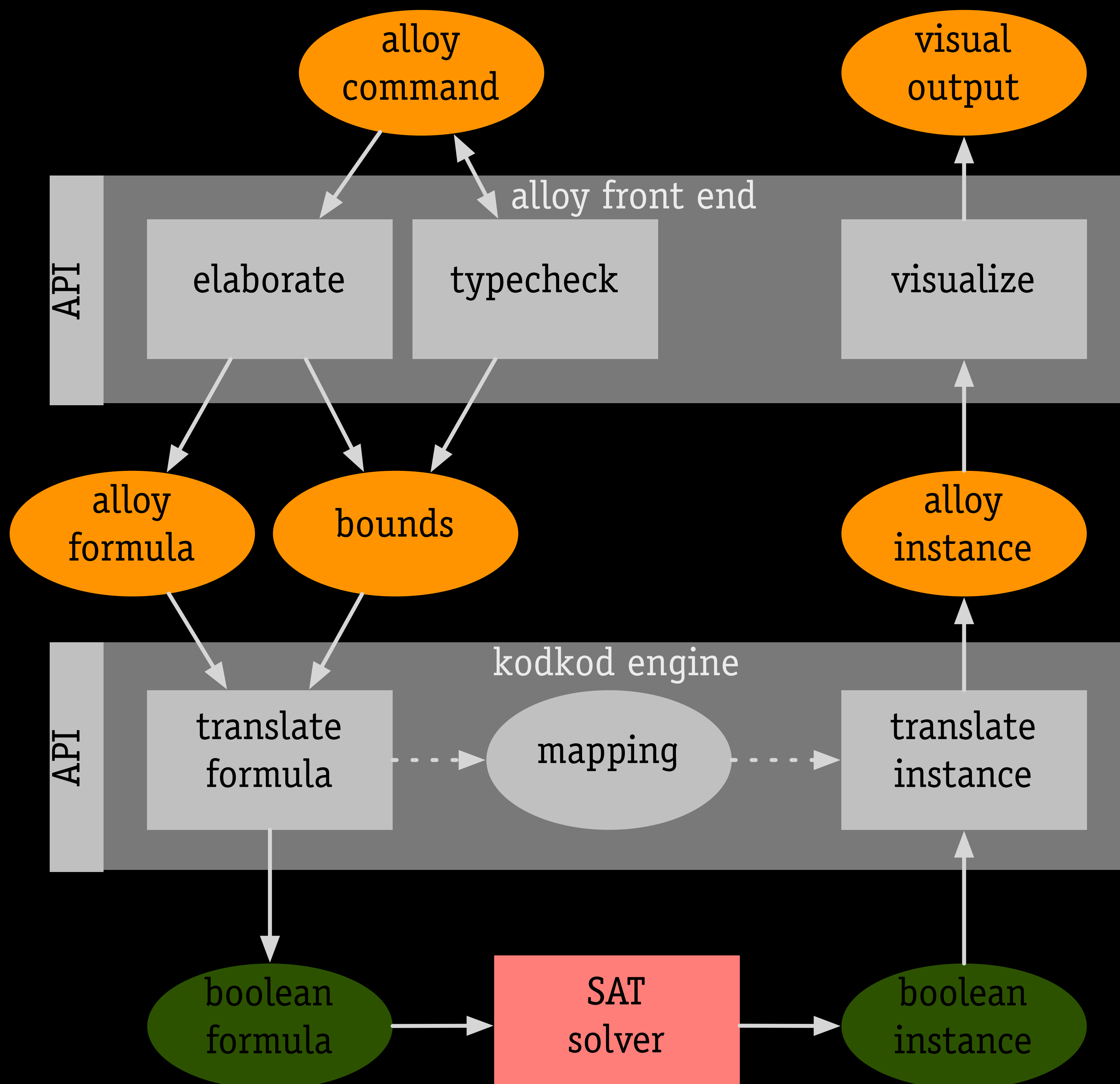
V0	v_0
V1	v_1
V2	v_2

	V0	V1	V2
B0	f_{00}	f_{01}	f_{02}
B1	f_{10}	f_{11}	f_{12}
B2	f_{20}	f_{21}	f_{22}

B0	b_0
B1	b_1
B2	b_2

$$b_0 = v_0 \wedge f_{00} \;\vee\; v_1 \wedge f_{10} \;\vee\; v_{02} \wedge f_{20}$$
$$b_1 = v_0 \wedge f_{01} \;\vee\; v_1 \wedge f_{11} \;\vee\; v_{02} \wedge f_{21}$$
$$b_2 = v_0 \wedge f_{02} \;\vee\; v_1 \wedge f_{12} \;\vee\; v_{02} \wedge f_{22}$$

analyzer architecture



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

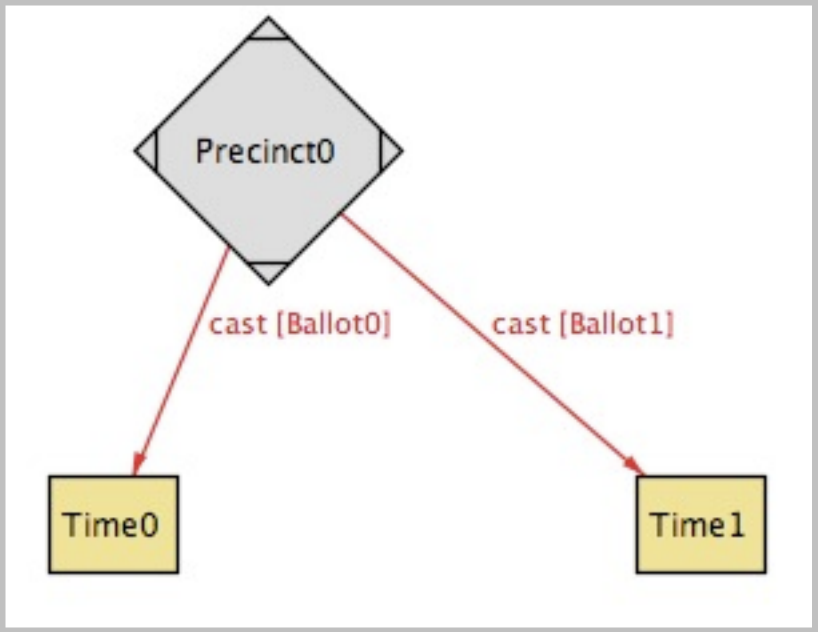
least conventional most useful

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

```
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

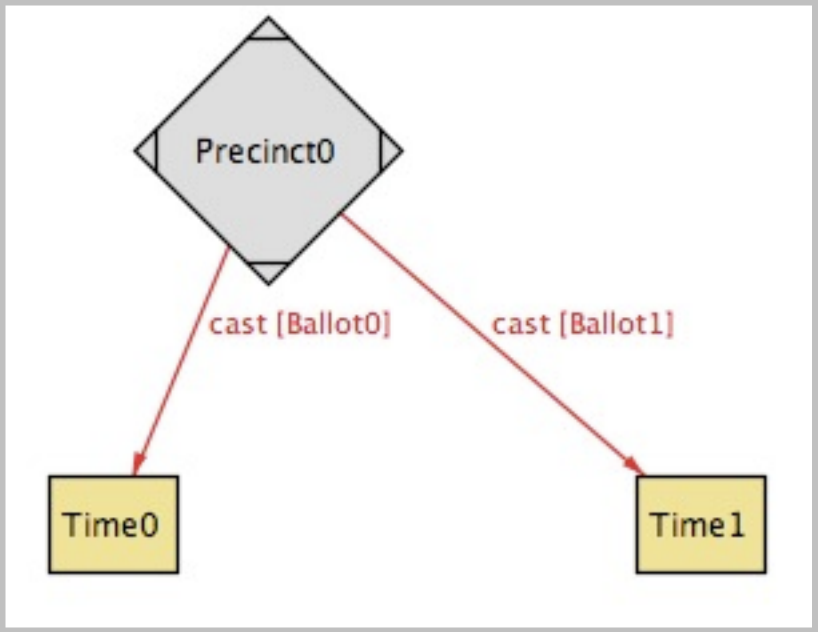
sig Candidate {}

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

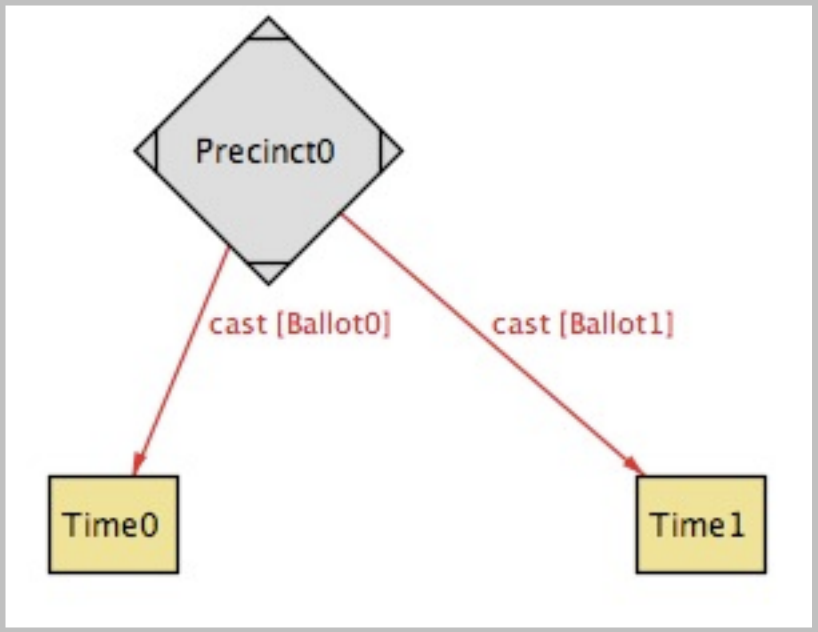
sig Candidate {}

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
    }

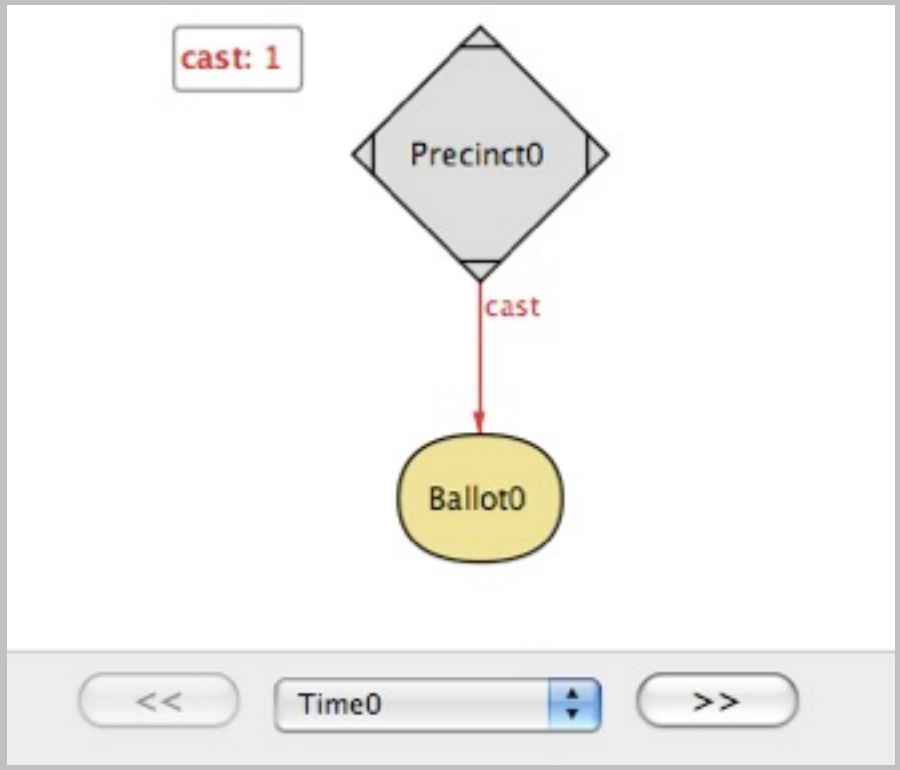


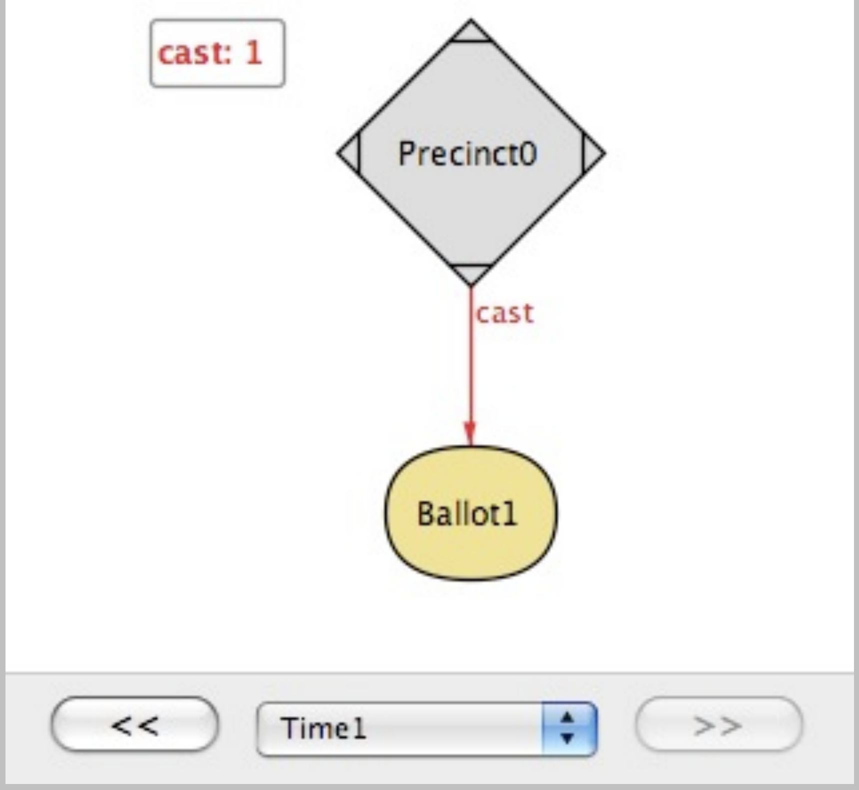
cast: a ternary relation

cast: Precinct -> Ballot -> Time

p.cast: Ballot -> Time

cast.t: Precinct -> Ballot





cast: projected onto Time

funky 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	-- one event starting in every time instant
	all e: Event | e.after = e.before.next	-- events execute in time order
	}
```

funky idioms: events

a generic model of 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              -- one event starting in every time instant
    all e: Event | e.after = e.before.next         -- events execute in time order
    }
```

funky idioms: events

a generic model of 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              -- one event starting in every time instant
	all e: Event | e.after = e.before.next         -- events execute in time order
	}
```

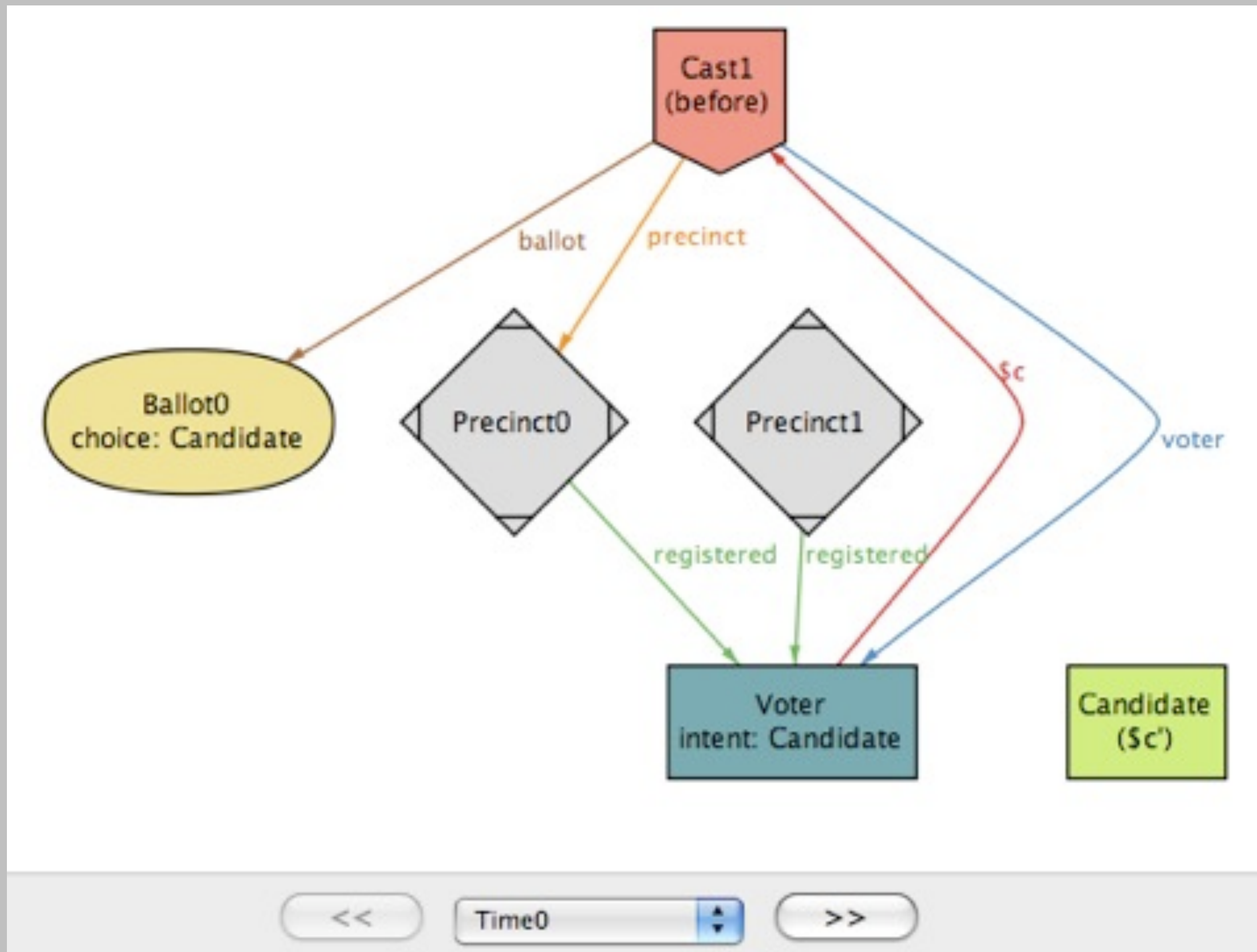
```
sig Cast extends Event { voter: Voter, ballot: Ballot, precinct: Precinct } {
	ballot.choice = voter.intent                           -- marks ballot according to intent
	voter in precinct.registered - precinct.voted.before   -- registered but hasn't voted yet
	voted.after = voted.before + precinct -> voter         -- has now voted
	cast.after = cast.before + precinct -> ballot          -- ballot is cast
	}
sig Count extends Event { tally: Tally } {
	tally.ballots = Precinct.cast.before                   -- ballots tallied are those cast in all precincts
	}
```

funky idioms: dynamics

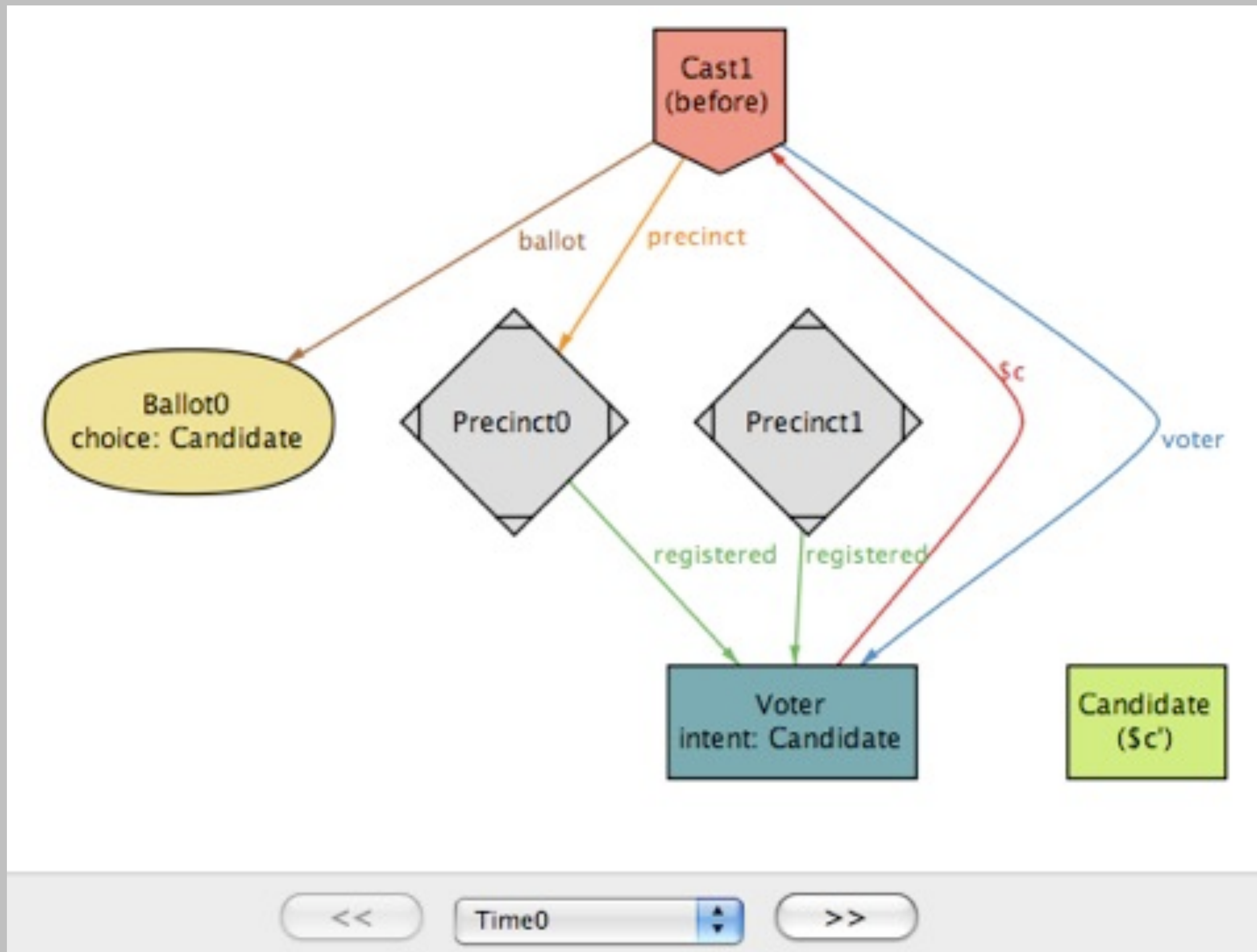
```
fact {
    no cast.first                                  -- at first, ballot boxes are empty
    all v: Voter | some c: Cast | c.voter = v      -- all voters cast ballots
    no ca: Cast, co: Count | ca.follows [co]       -- no casting after counting
    }
```

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
```

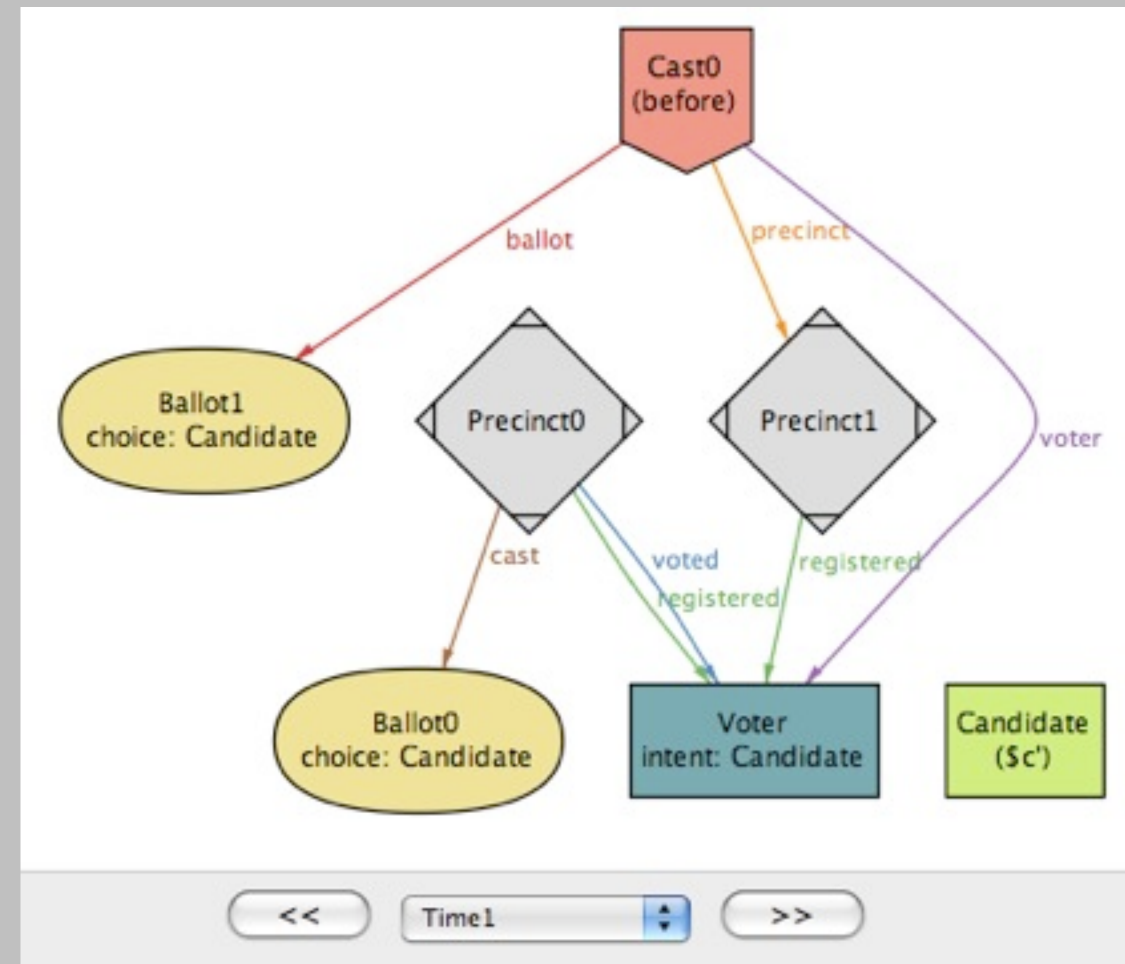


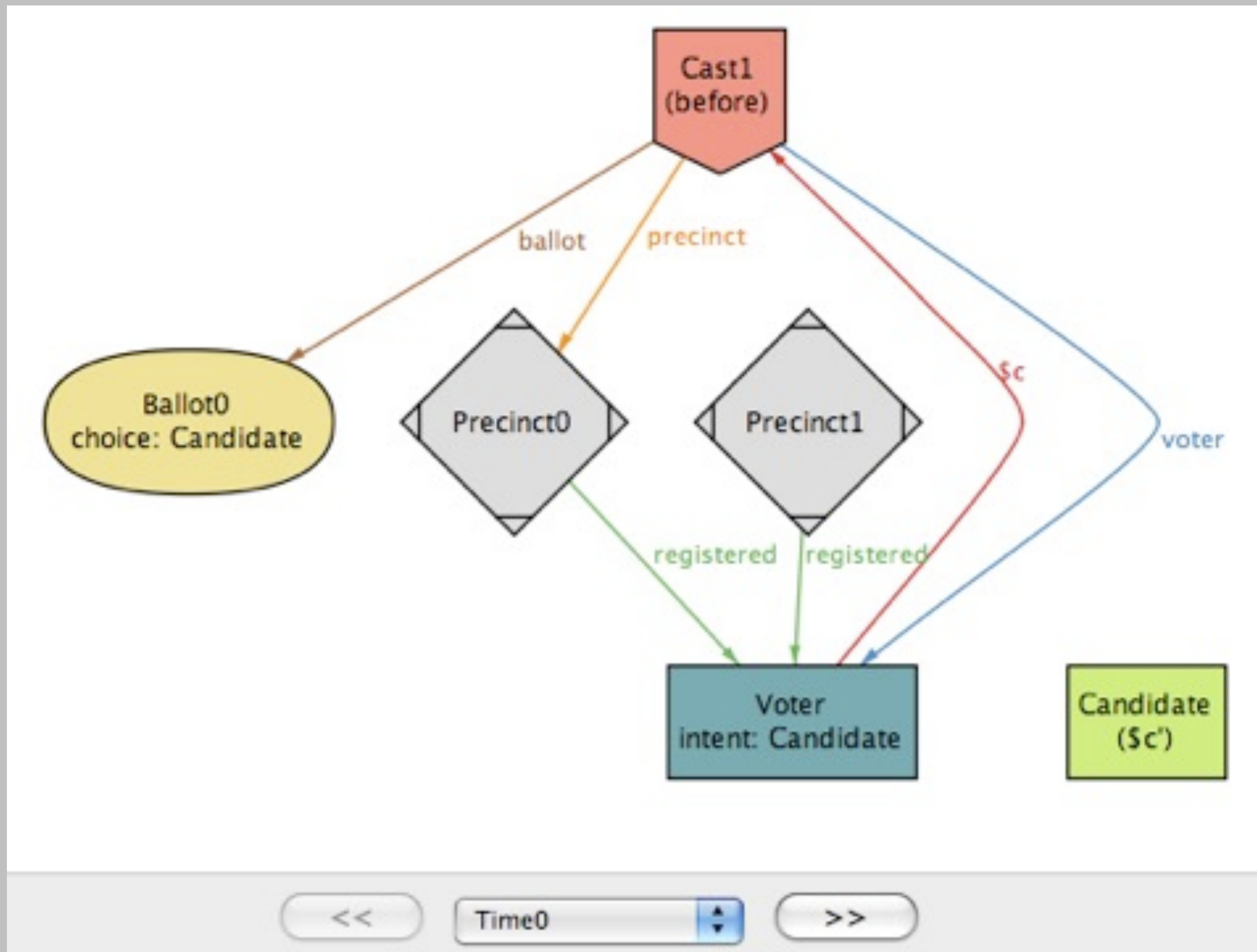
Voter casts Ballot0 in Precinct0



Voter casts Ballot0 in Precinct0

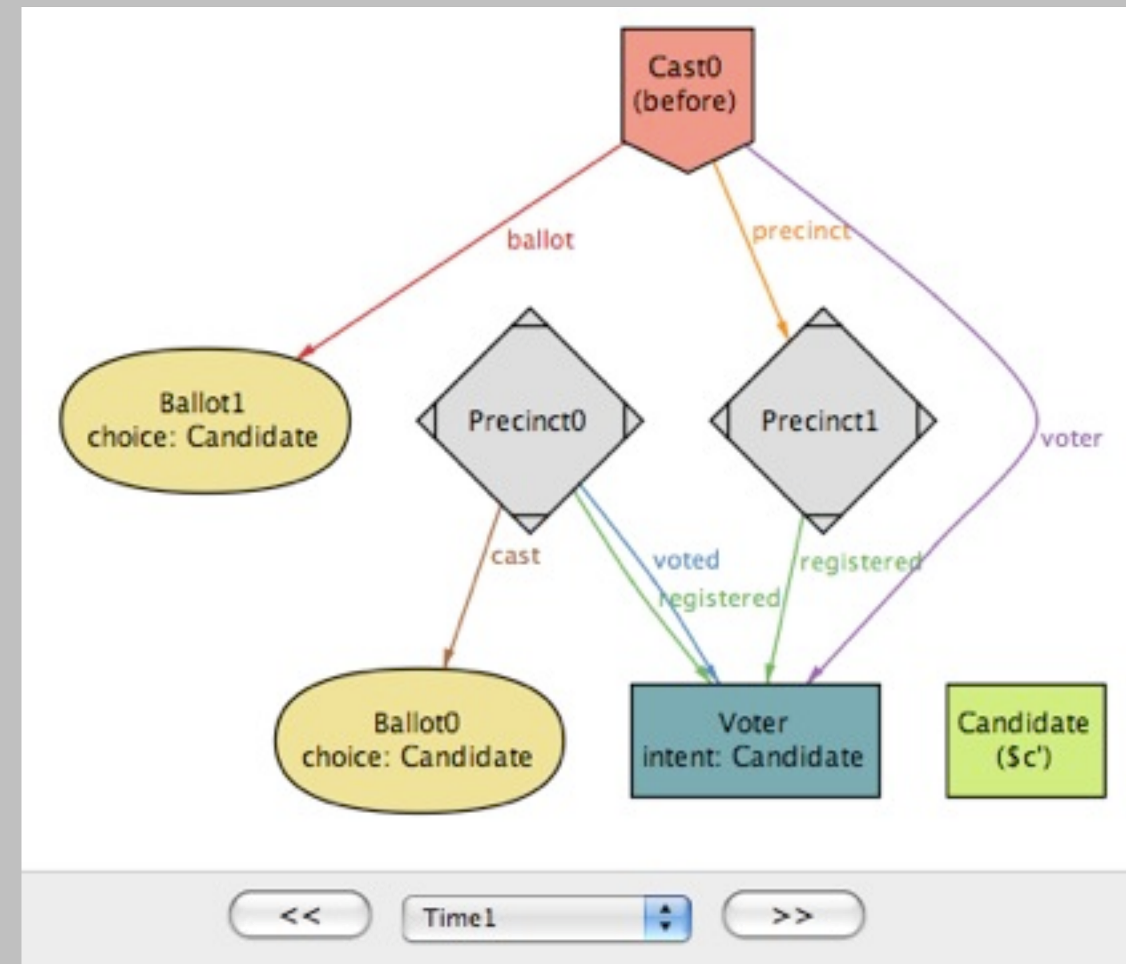
Voter casts Ballot1 in Precinct1

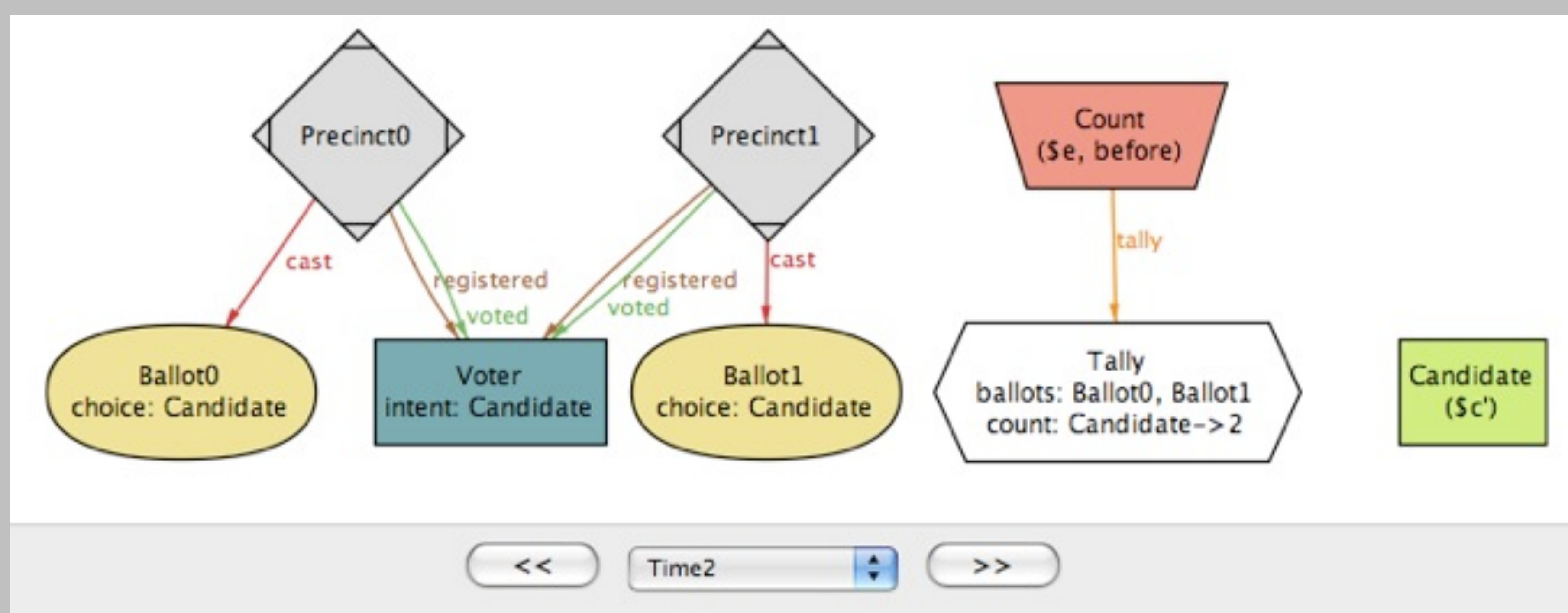




Voter casts Ballot0 in Precinct0

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

tally count for candidate c is # intent.c

navigations are succinct

tally count for candidate c is $\# \, intent.c$

in Z

$| \, intent^{\sim} (\!| \{c\} |\!) \, |$

navigations are succinct

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

in Z

$|\, \mathit{intent}^{\sim} (\!| \{c\} |\!) \,|$

in SQL

select count(*) from voter where intent = c

navigations are succinct

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

in Z

$| \mathit{intent}^{\sim} (\!| \{c\} |\!) |$

in SQL

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)

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)

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

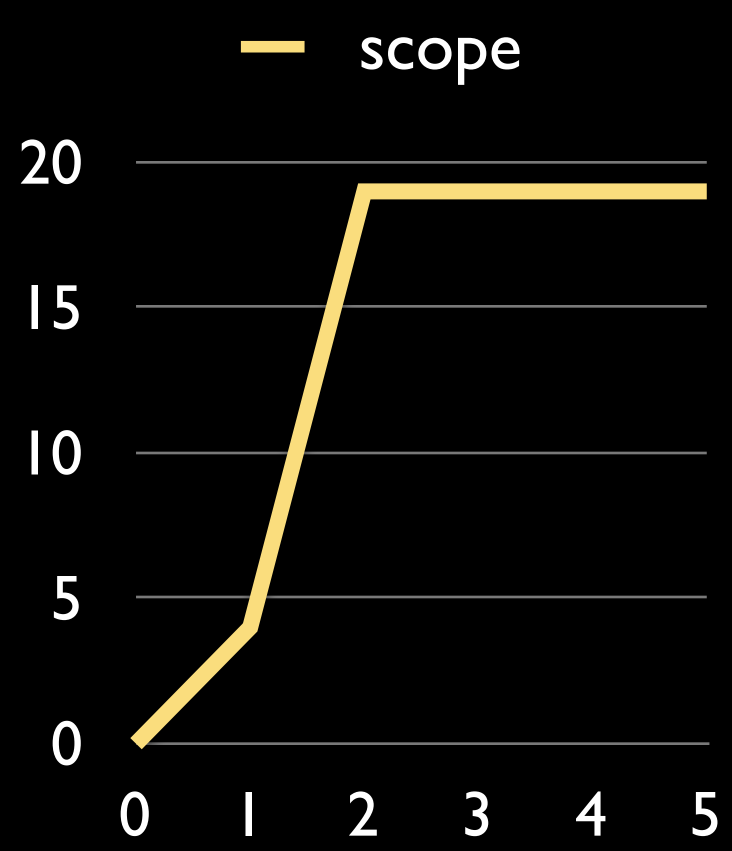
analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]

small scope hypothesis



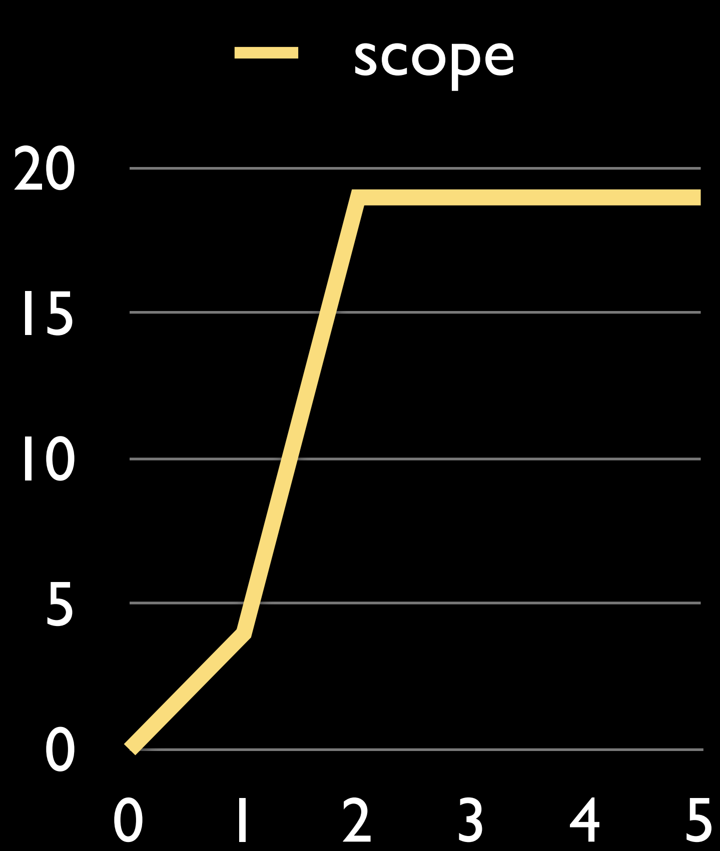
analysis of KOA voting code

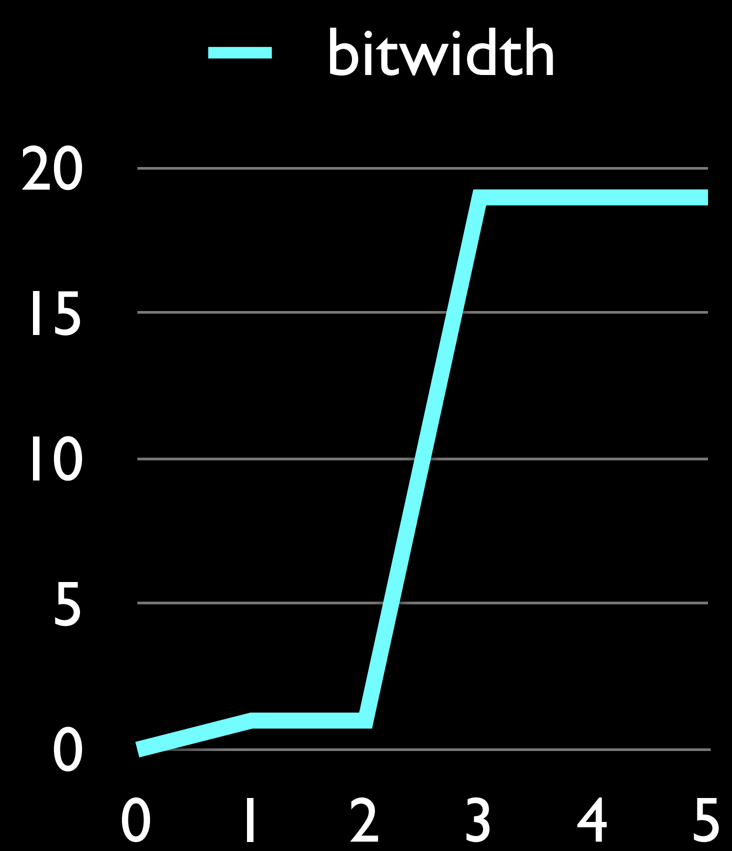
19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]

small scope hypothesis





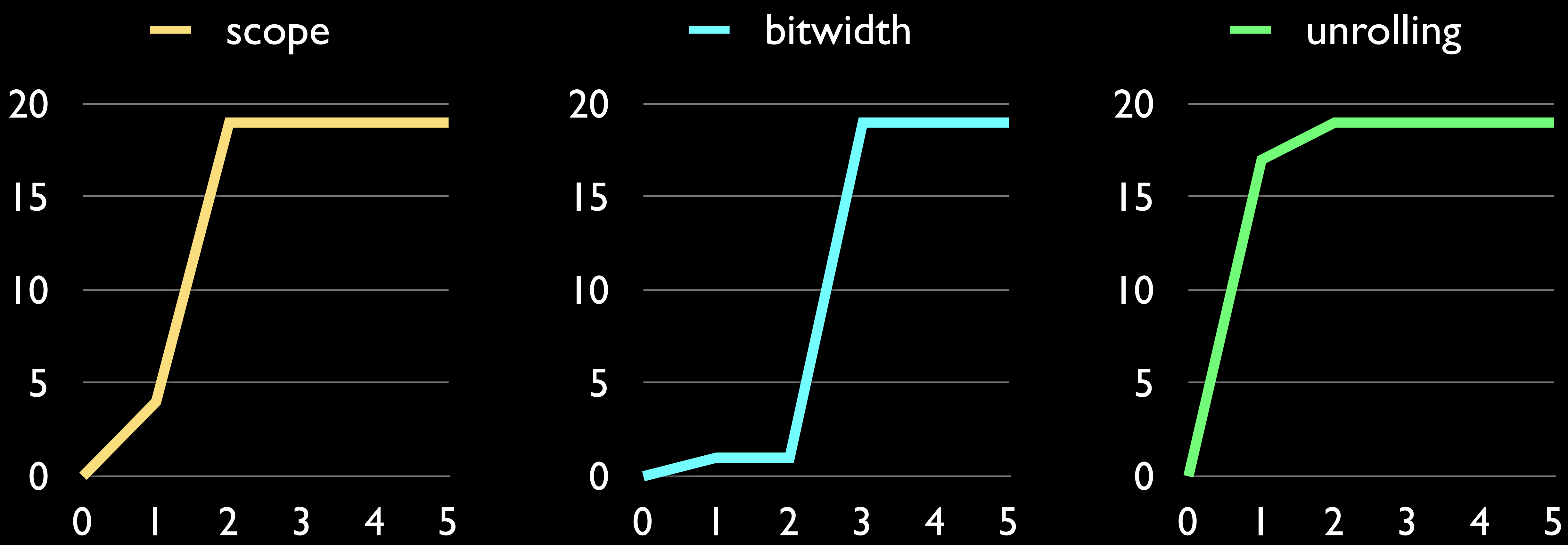
analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

[Greg Dennis, 2008]

small scope hypothesis



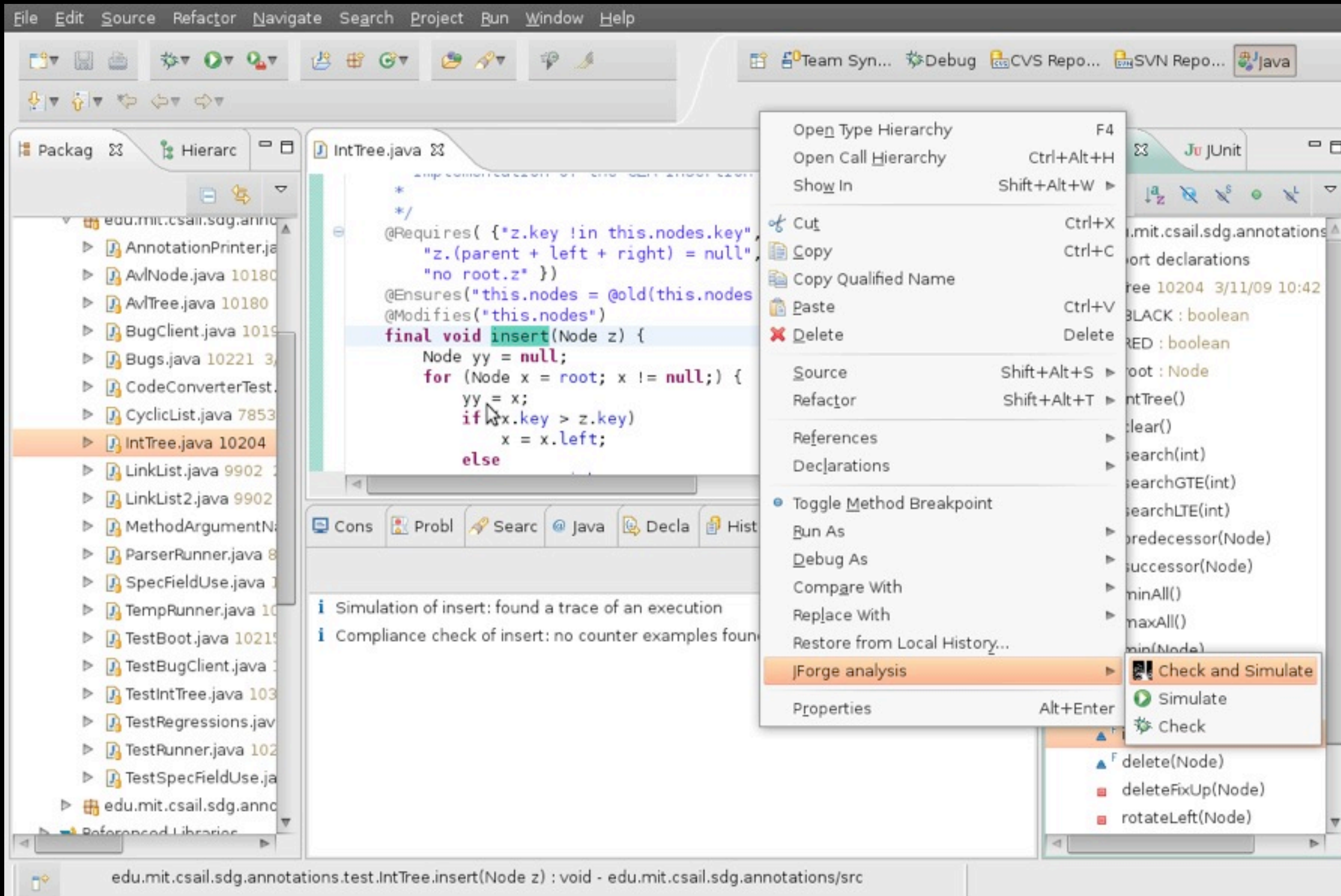
analysis of KOA voting code

19 methods violating specs

how many bugs found in scope of k?

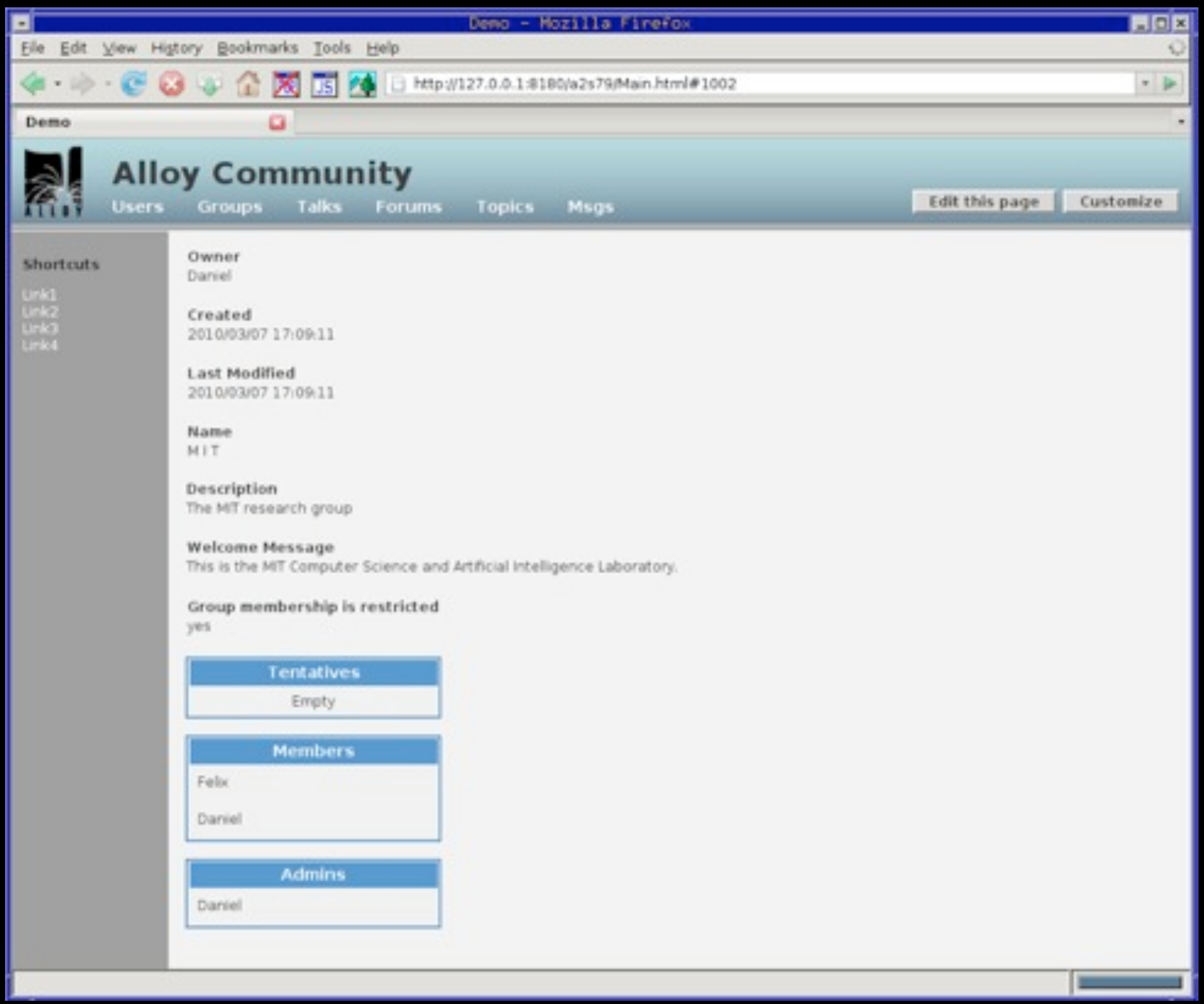
[Greg Dennis, 2008]

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

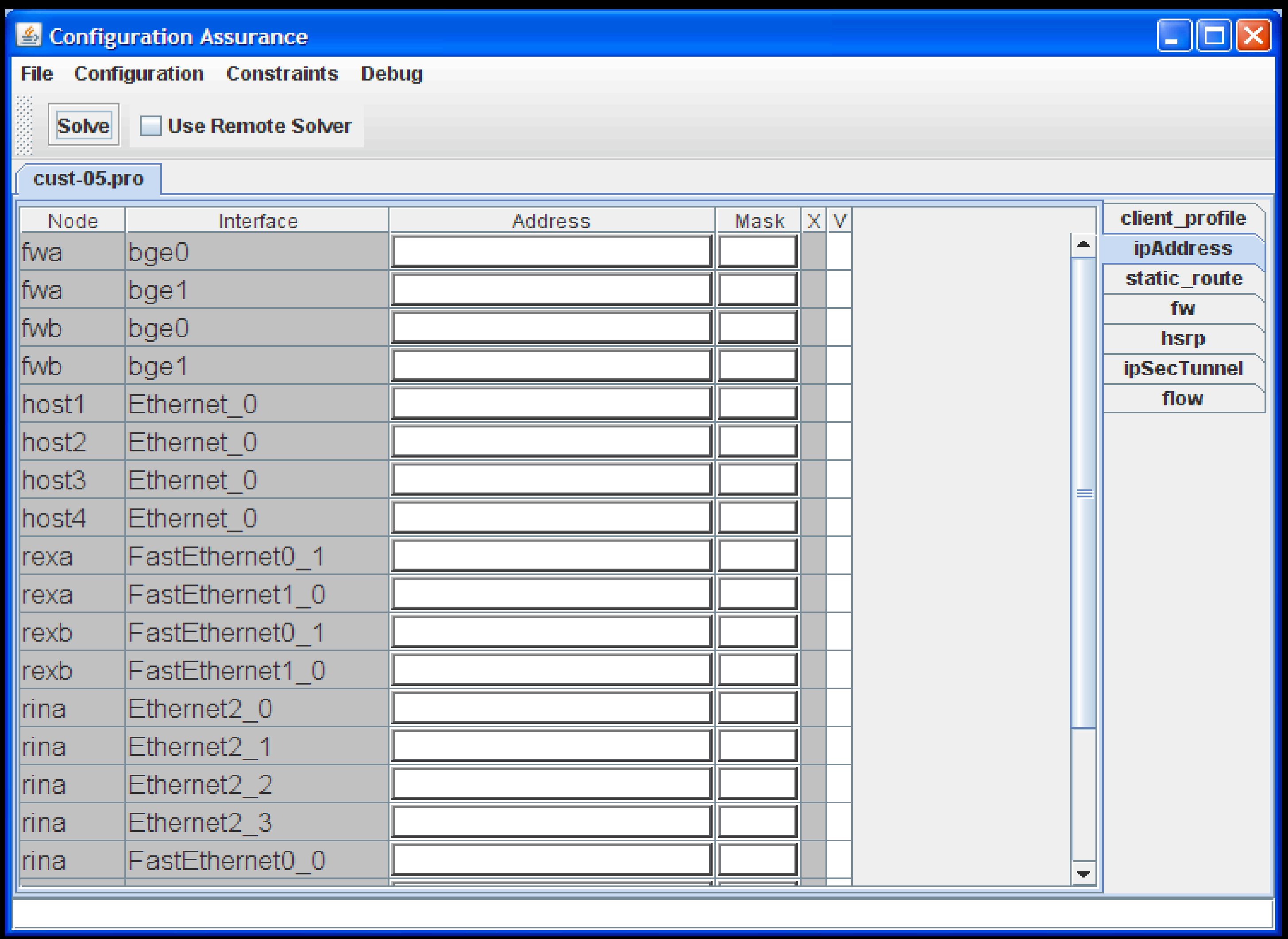
'We don't need hackers
to break systems because
they're falling apart
by themselves'
Peter Neumann

bad configurations responsible for
80% of USAF vulnerabilities
NSA

declarative configuration

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

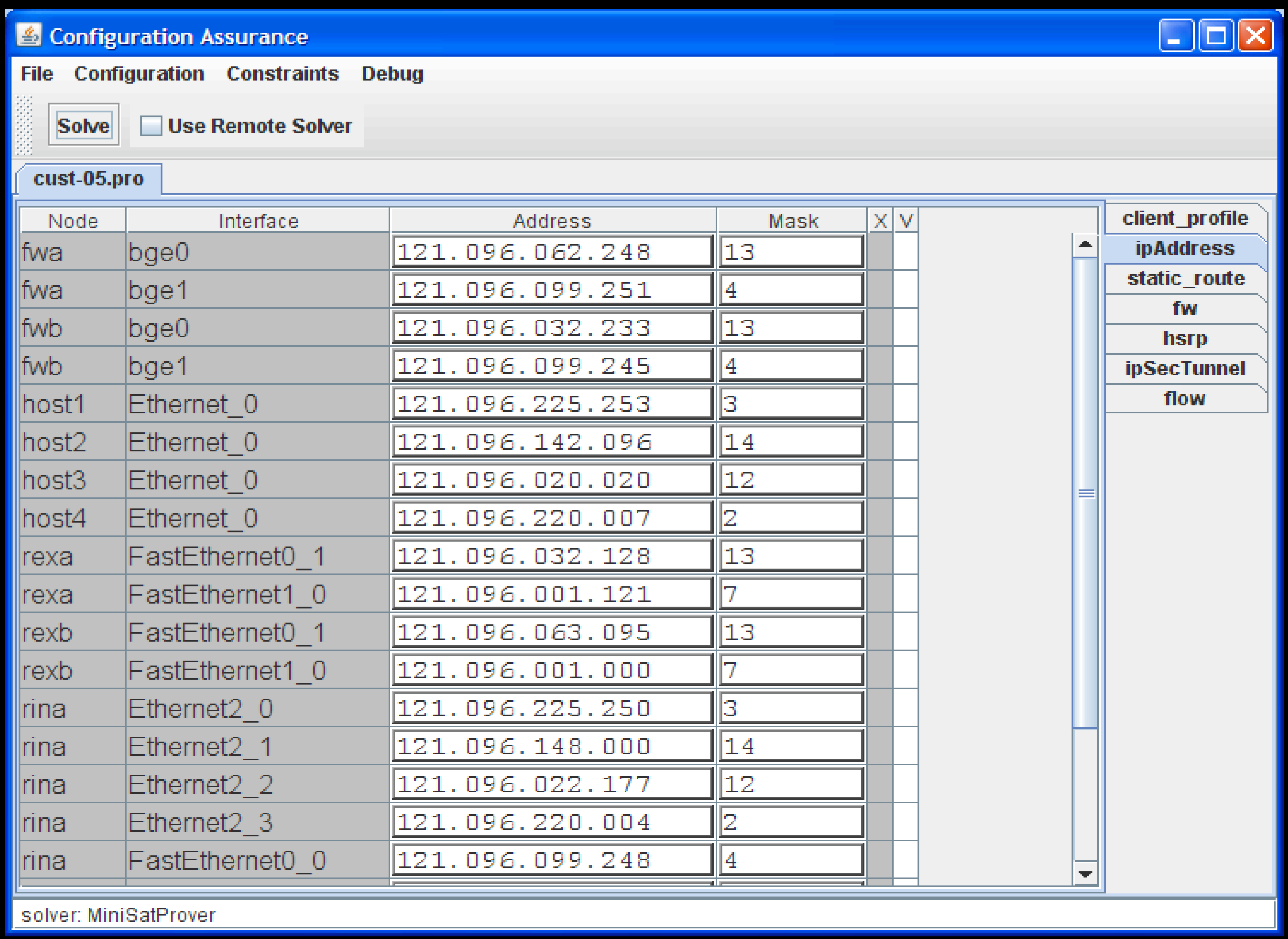
declarative configuration



ConfigAssure [Telcordia, 2008]

use Kodkod to find network configuration that meets rules & goals

declarative configuration



Node	Interface	Address	Mask	X	V
fwa	bge0	121.096.062.248	13		
fwa	bge1	121.096.099.251	4		
fwb	bge0	121.096.032.233	13		
fwb	bge1	121.096.099.245	4		
host1	Ethernet_0	121.096.225.253	3		
host2	Ethernet_0	121.096.142.096	14		
host3	Ethernet_0	121.096.020.020	12		
host4	Ethernet_0	121.096.220.007	2		
rexa	FastEthernet0_1	121.096.032.128	13		
rexa	FastEthernet1_0	121.096.001.121	7		
rexb	FastEthernet0_1	121.096.063.095	13		
rexb	FastEthernet1_0	121.096.001.000	7		
rina	Ethernet2_0	121.096.225.250	3		
rina	Ethernet2_1	121.096.148.000	14		
rina	Ethernet2_2	121.096.022.177	12		
rina	Ethernet2_3	121.096.220.004	2		
rina	FastEthernet0_0	121.096.099.248	4		

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

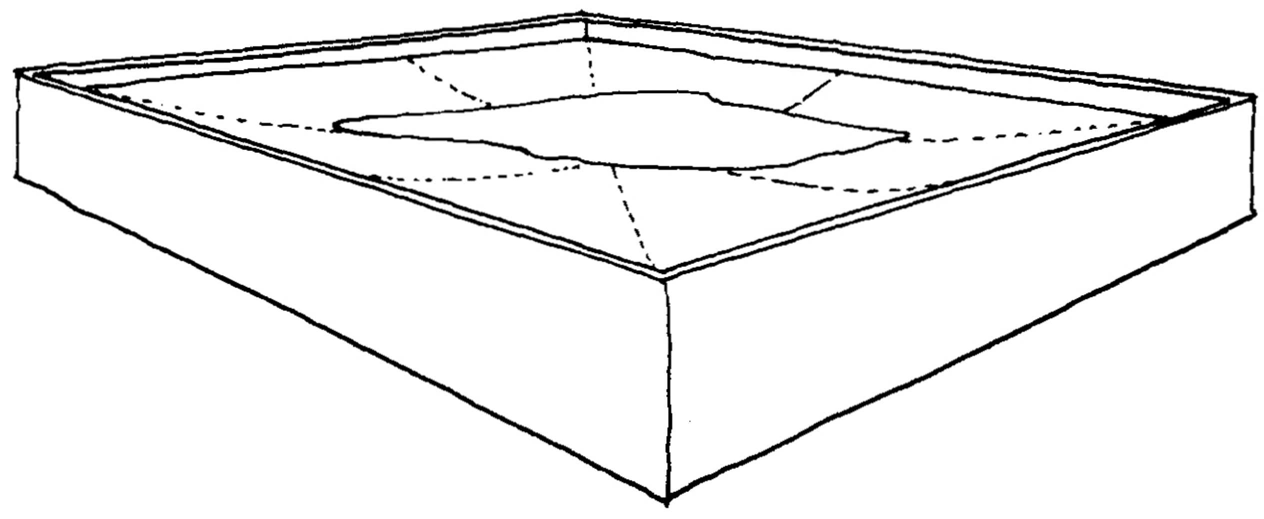
multiobjective optimization
Rayside & Estler

a new direction

kemper arena, kansas city, 2007

kemper arena, 1979

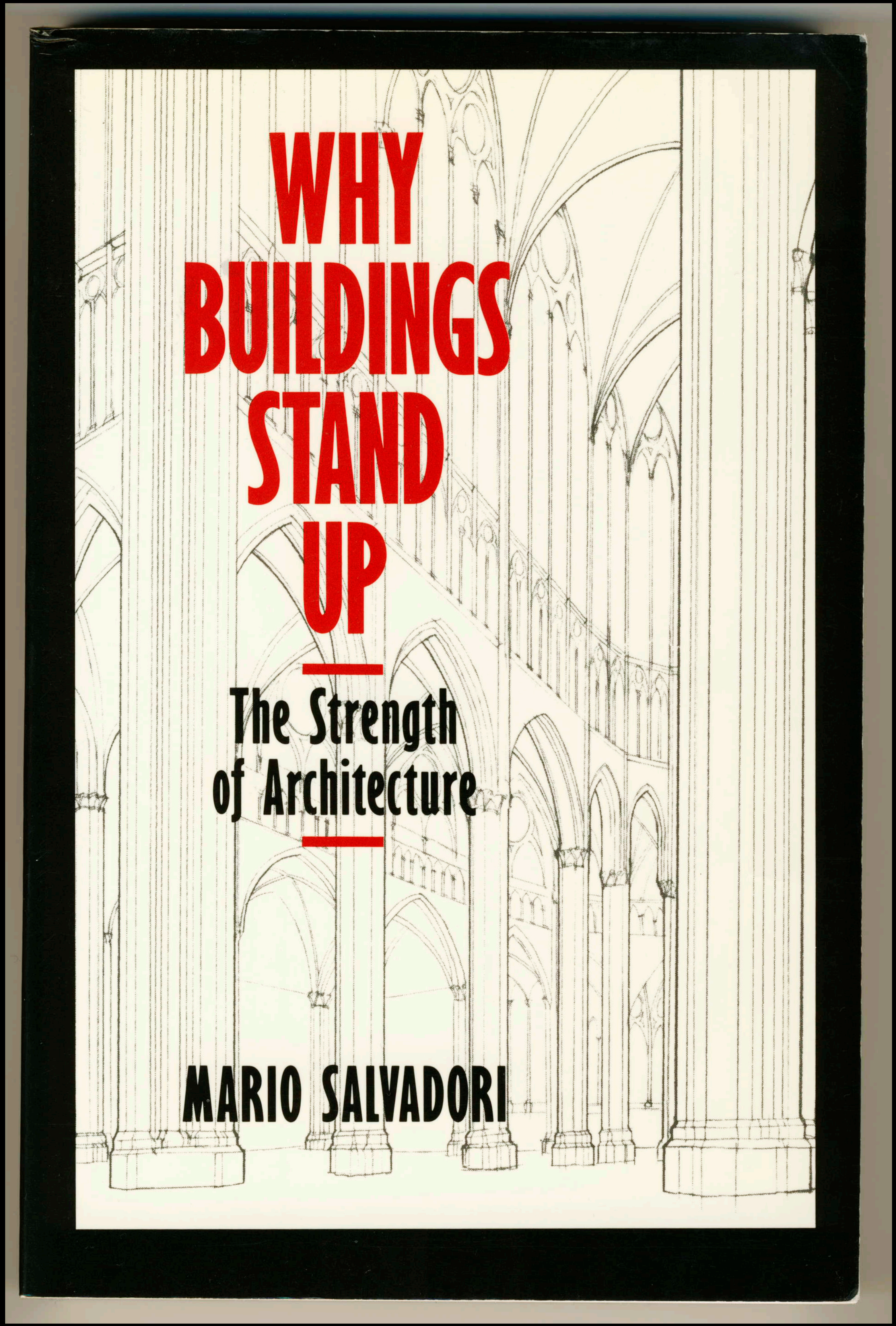
what happened?



4.8 Ponding of a Flat Roof

Levy & Salvadori, *Why Buildings Fall Down*

failure = flawed success story



Therac 25

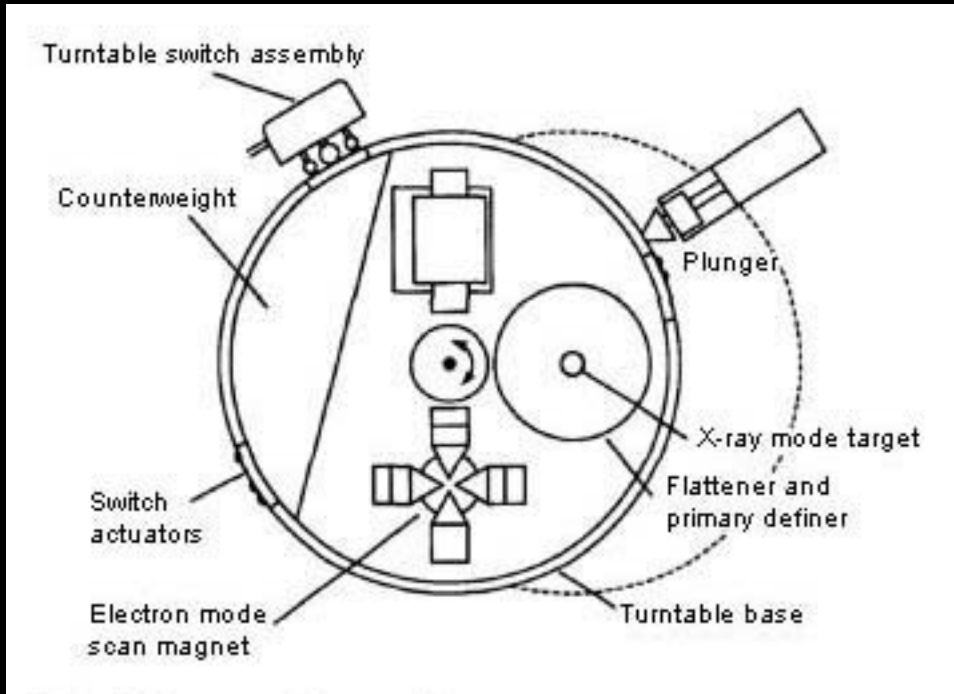


Figure B. Upper turntable assembly

AECL fault tree analysis (1983)

did not include software

P(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

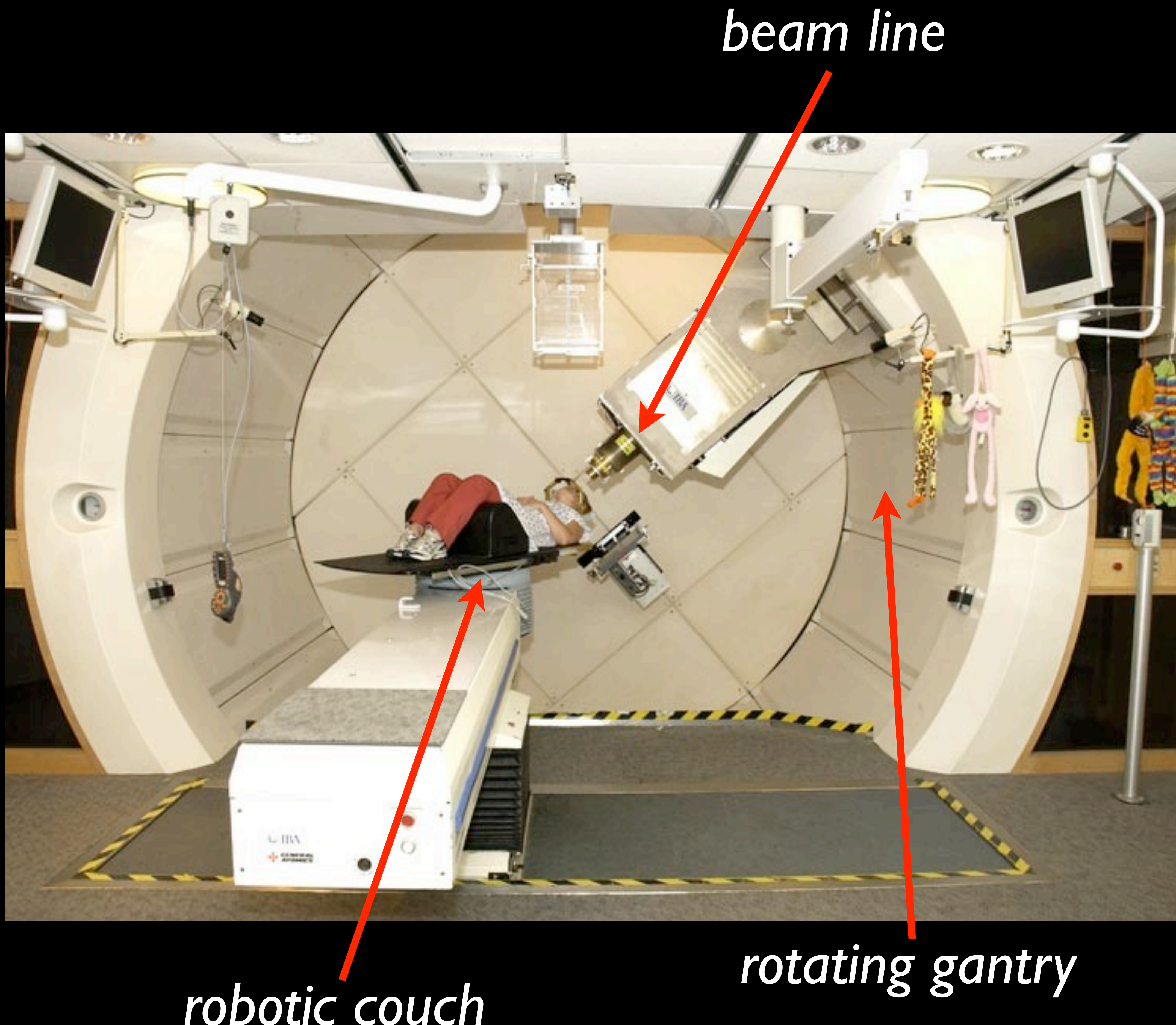
be realistic

not 'full correctness': prioritize requirements

don't be a masochist

use a safe language

design to localize critical properties


beam line
rotating gantry
robotic couch

a dependence diagram

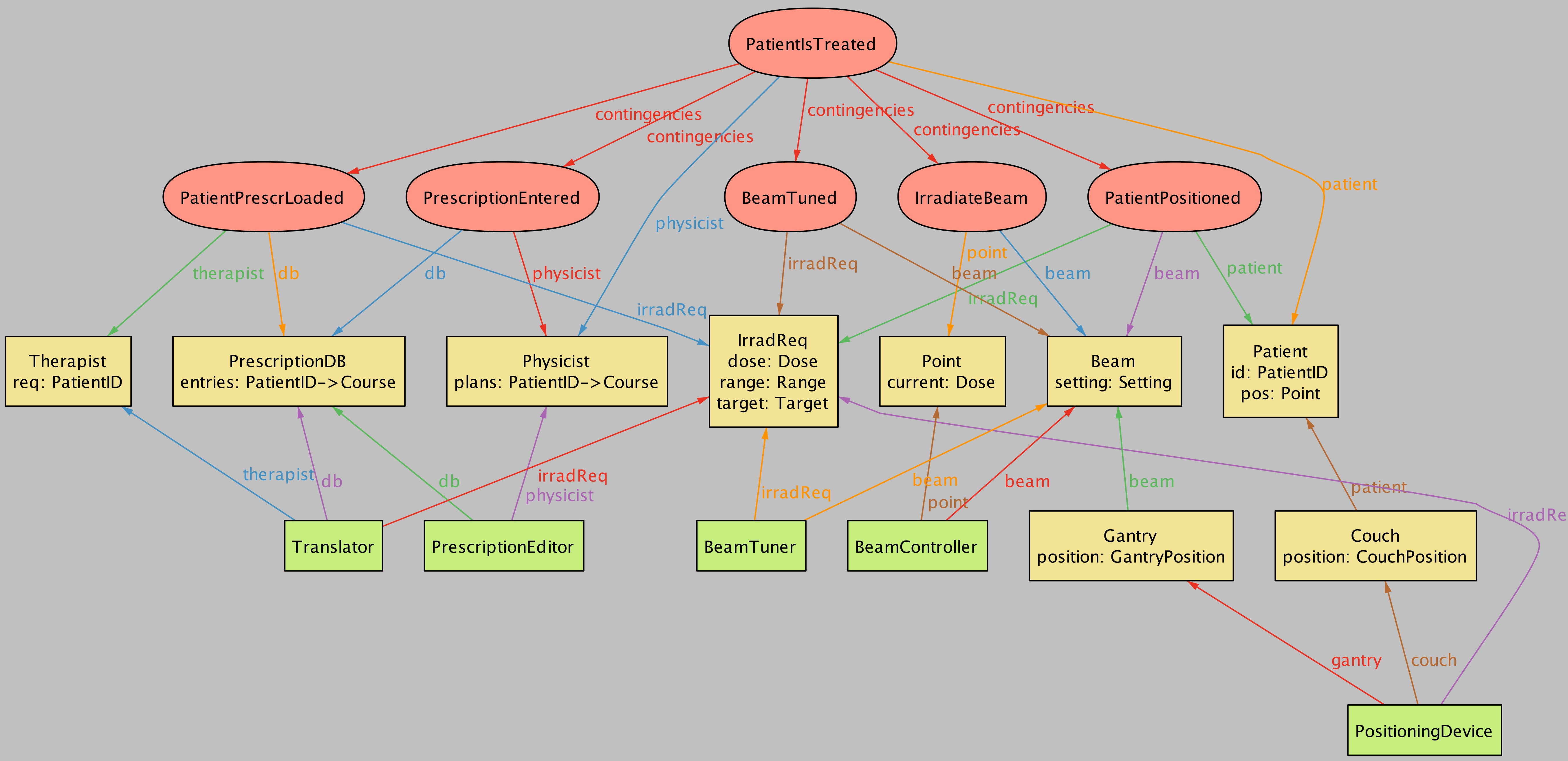


diagram by Eunsuk Kang

reflections

the future of modelling

the future of modelling

it will happen, really

domain-specific approaches

Rails <u>is</u> 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 <u>is</u> 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 <u>really</u> 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 <u>really</u> hard sell
easy to get new idea funded once popular

a mistake?

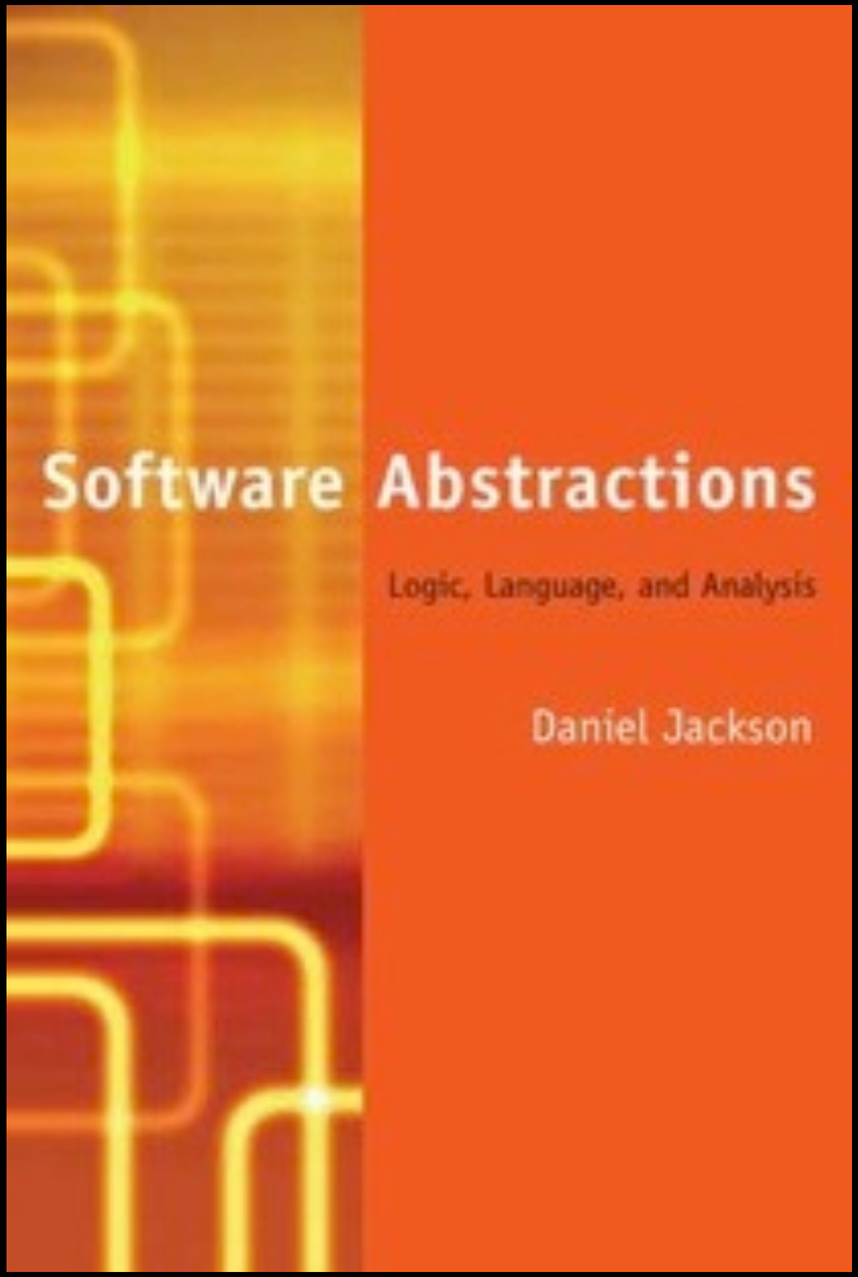
keeping Alloy general & pure
limited impact, except in teaching

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