The Alloyed Joys of Software Engineering Research

Daniel Jackson

Keynote · ICSE 2017 · Buenos Aires

Alloy

alloy’s cultural origins

alloy's cultural origins

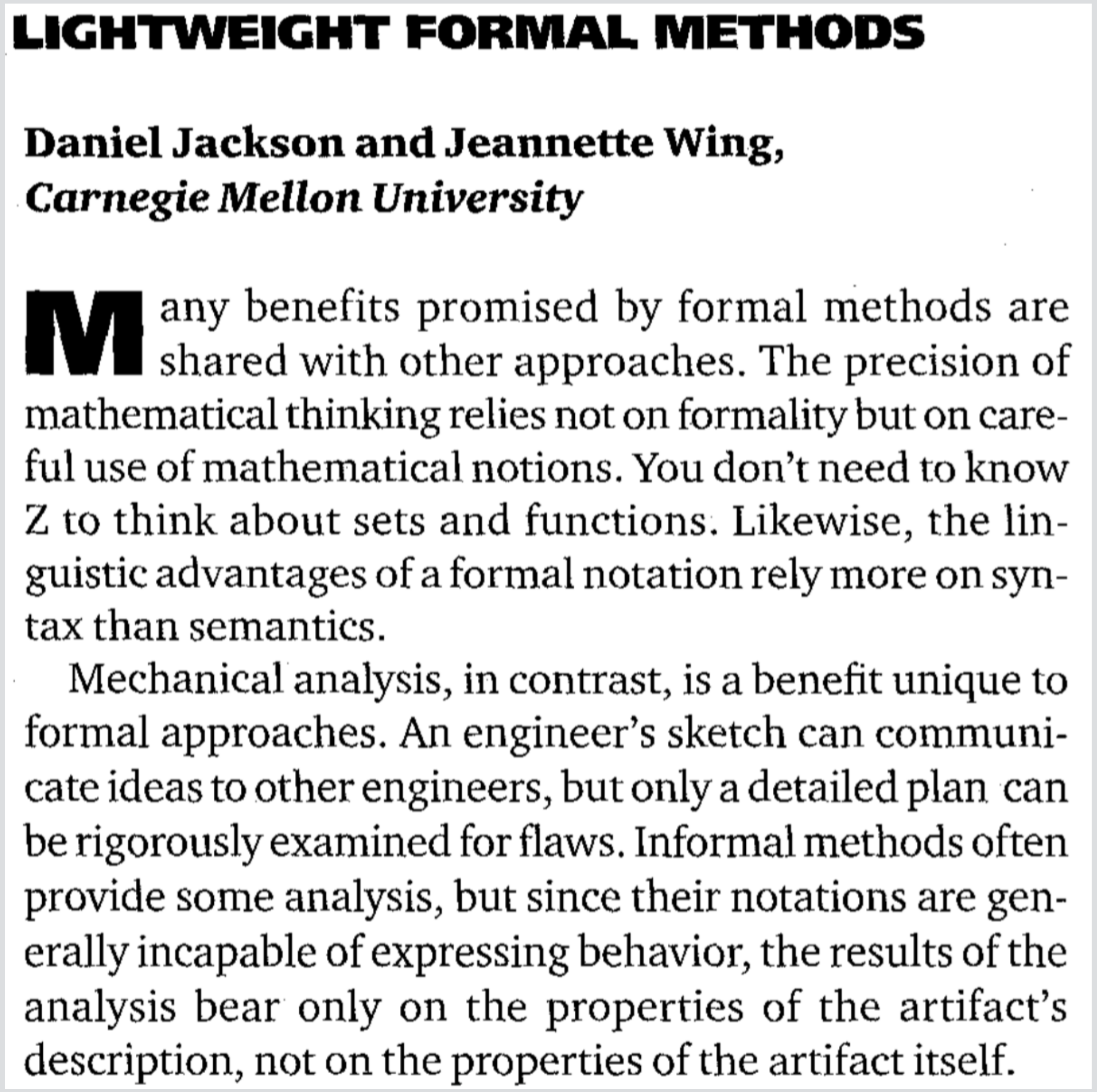
Oxford, home of Z

alloy's cultural origins

Oxford, home of Z

Pittsburgh, home of SMV

lightweight formal methods



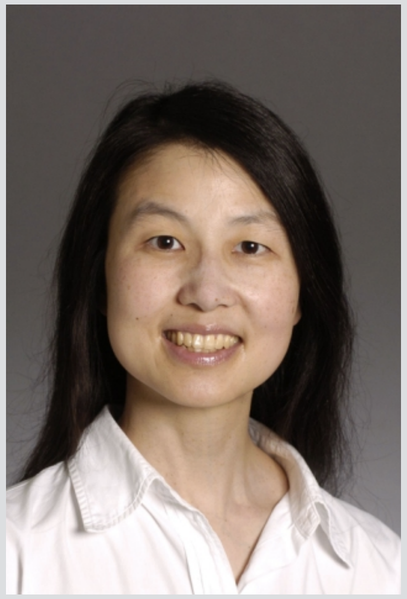
LIGHTWEIGHT FORMAL METHODS

Daniel Jackson and Jeannette Wing,
Carnegie Mellon University

Many benefits promised by formal methods are shared with other approaches. The precision of mathematical thinking relies not on formality but on careful use of mathematical notions. You don't need to know Z to think about sets and functions. Likewise, the linguistic advantages of a formal notation rely more on syntax than semantics.

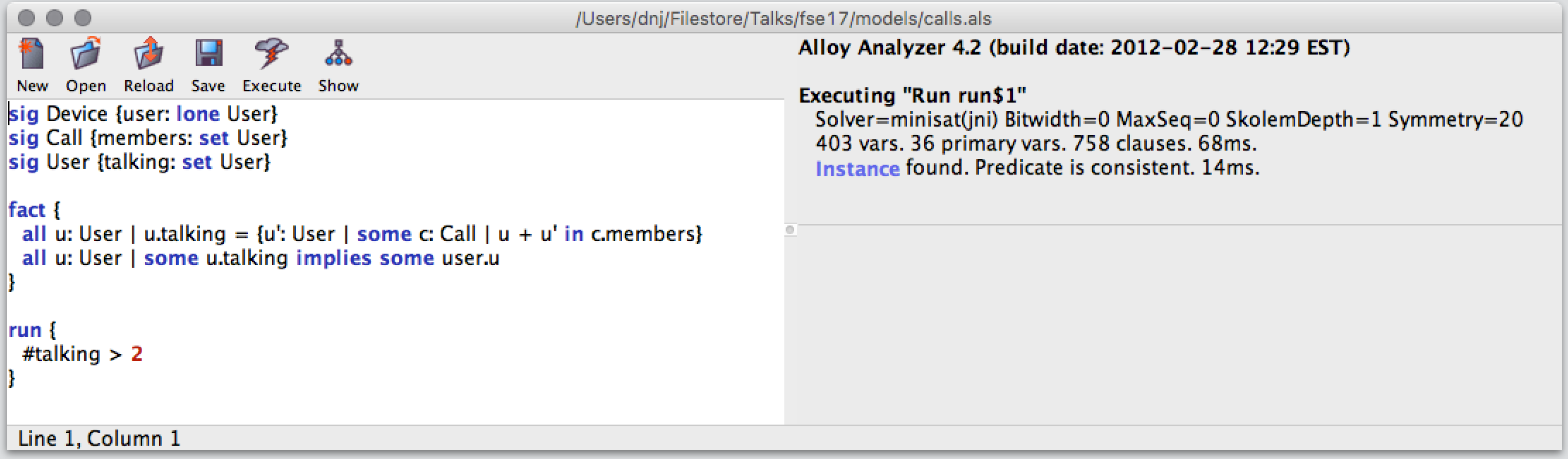
Mechanical analysis, in contrast, is a benefit unique to formal approaches. An engineer's sketch can communicate ideas to other engineers, but only a detailed plan can be rigorously examined for flaws. Informal methods often provide some analysis, but since their notations are generally incapable of expressing behavior, the results of the analysis bear only on the properties of the artifact's description, not on the properties of the artifact itself.

IEEE Computer, 1996



traditional FM
full model of behavior
analysis to show no bugs

lightweight FM
model of critical aspect
analysis to find bugs

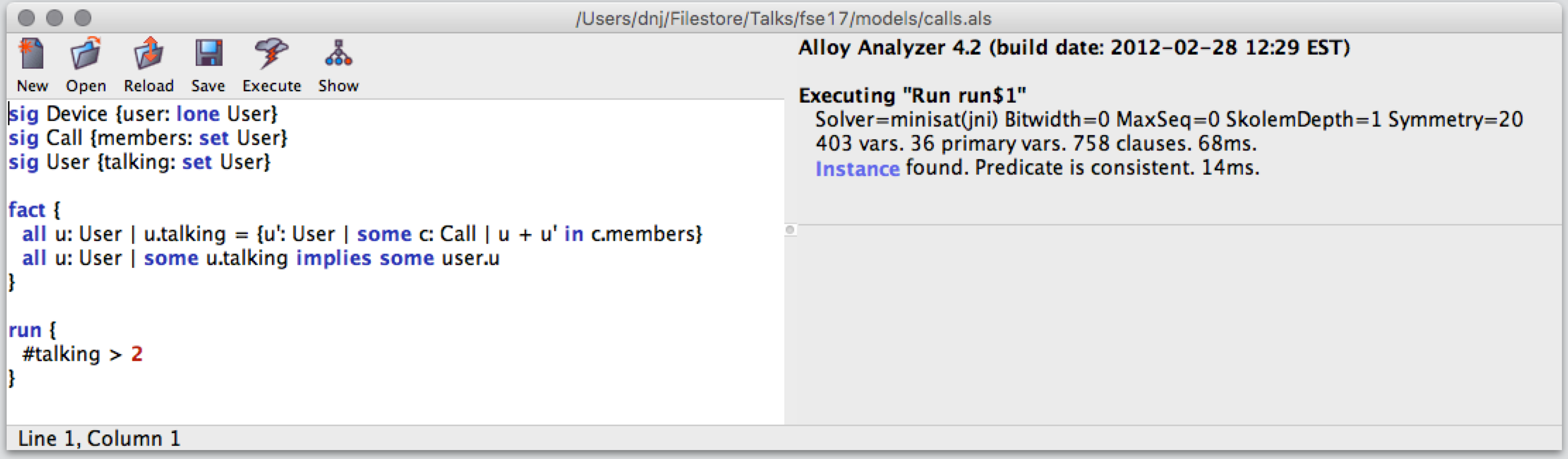


/Users/dnj/Filestore/Talks/fse17/models/calls.als

```
sig Device {user: lone User}
sig Call {members: set User}
sig User {talking: set User}

fact {
  all u: User | u.talking = {u': User | some c: Call | u + u' in c.members}
  all u: User | some u.talking implies some user.u
}

run {
  #talking > 2
}
```

```
Alloy Analyzer 4.2 (build date: 2012-02-28 12:29 EST)

Executing "Run run$1"
  Solver=minisat(jni) Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  403 vars. 36 primary vars. 758 clauses. 68ms.
  Instance found. Predicate is consistent. 14ms.
```

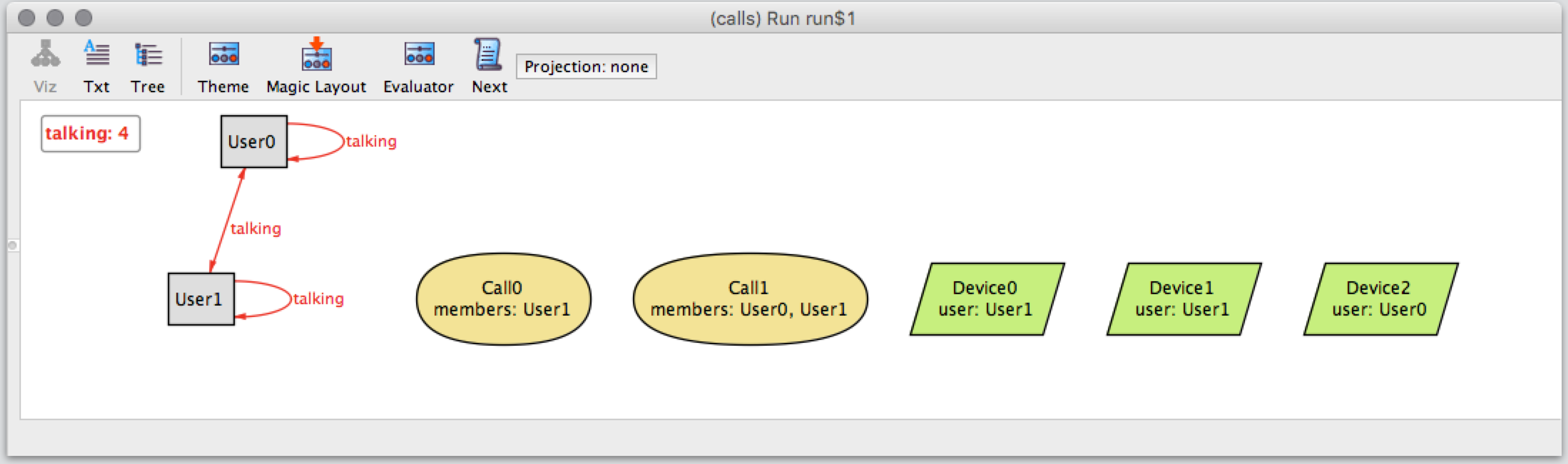
Line 1, Column 1



```
sig Device {user: lone User}
sig Call {members: set User}
sig User {talking: set User}

fact {
  all u: User | u.talking = {u': User | some c: Call | u + u' in c.members}
  all u: User | some u.talking implies some user.u
}

run {
  #talking > 2
}
```

```
Alloy Analyzer 4.2 (build date: 2012-02-28 12:29 EST)

Executing "Run run$1"
  Solver=minisat(jni) Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  403 vars. 36 primary vars. 758 clauses. 68ms.
  Instance found. Predicate is consistent. 14ms.
```

Line 1, Column 1

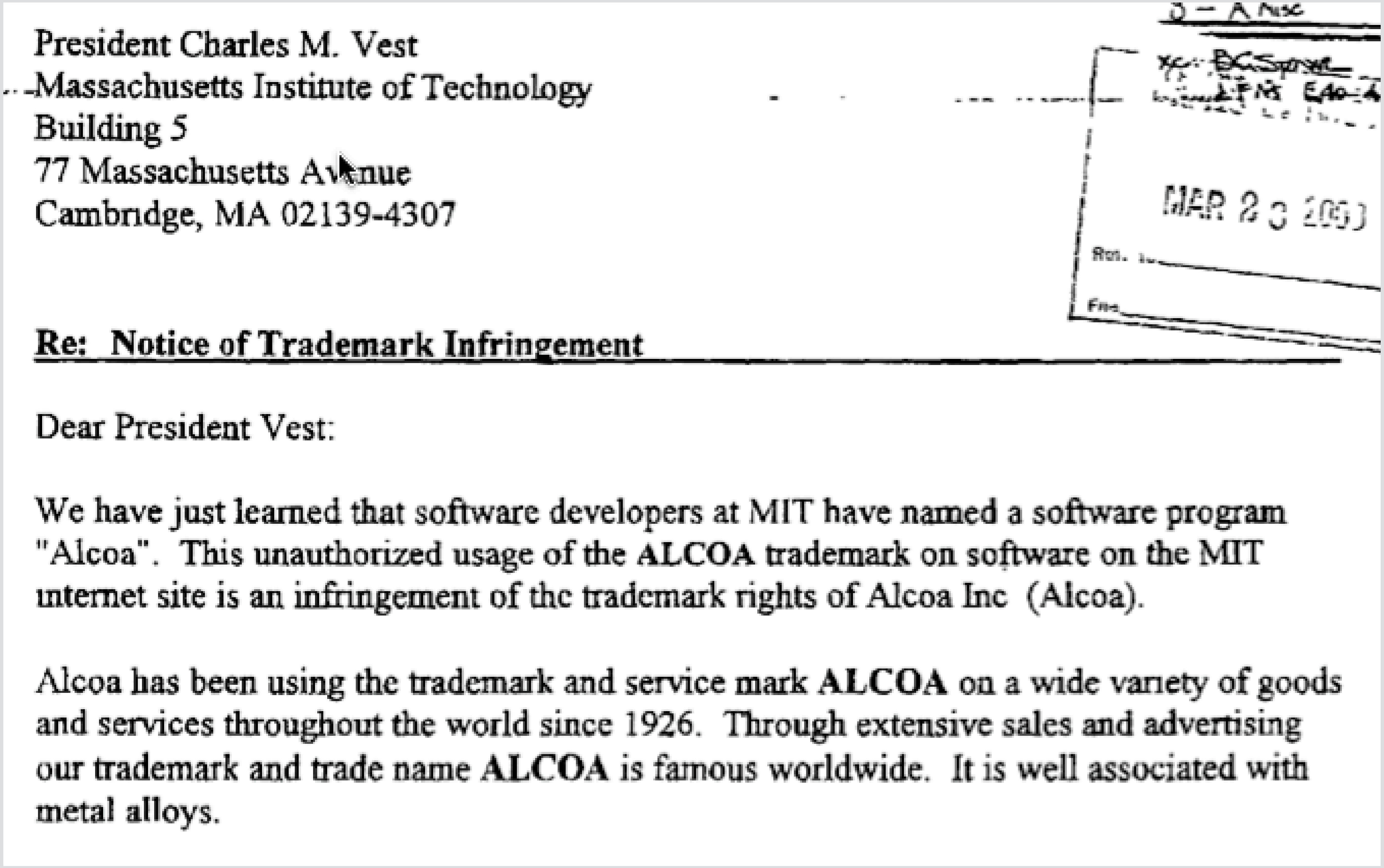


alloy timeline

version	*language*	*analysis*	*sample case study*
Nitpick (1995)	**relational** calculus subset of Z	relation enumeration	IPv6 routing
Alloy 1 (1999)	+ navigation exps **quantifiers**	WalkSAT, **Davis Putnam**	intentional naming
Alloy 2 (2001)	+ non-binary relations **signatures**	Chaff, Berkmin symmetry, sharing	Unison filesync
Alloy 3 (2004)	+ subtyping overloading	atomization (bad)	Mondex smartcard
Alloy 4 (2007)	+ meta, sequences arithmetic	**bounds** better sharing	flash filesystem

the **al**loy **co**nstraint **a**nalyzer

the alloy constraint analyzer



3 – A NSC

xc: BCSprse [illegible]

MAR 2 0 2003

Rec.

File

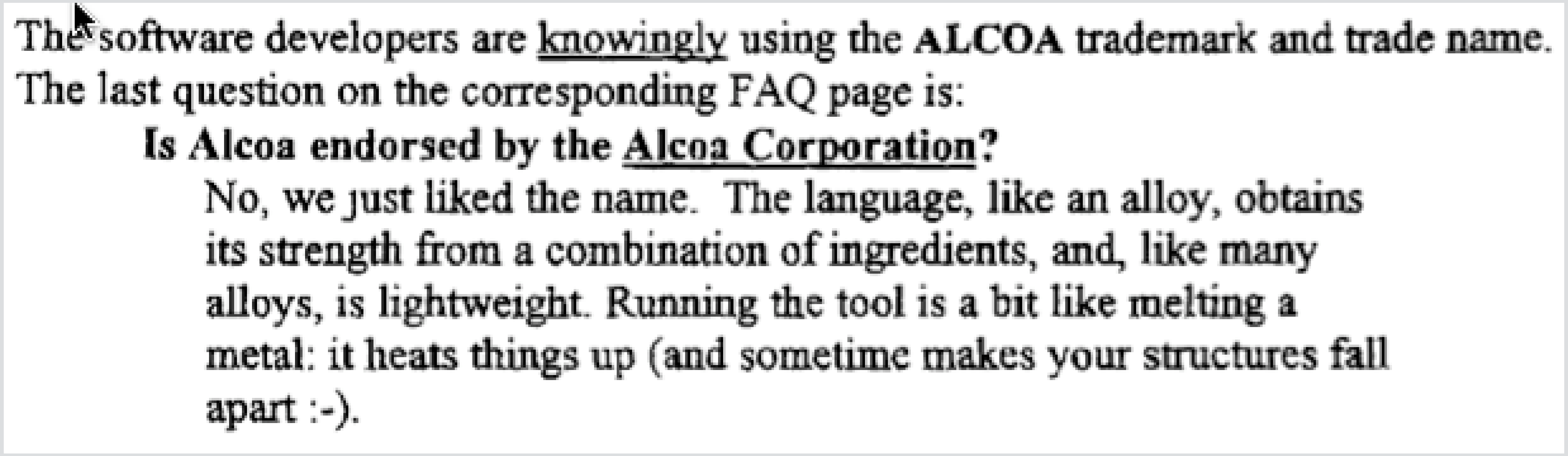
President Charles M. Vest
Massachusetts Institute of Technology
Building 5
77 Massachusetts Avenue
Cambridge, MA 02139-4307

<u>Re: Notice of Trademark Infringement</u>

Dear President Vest:

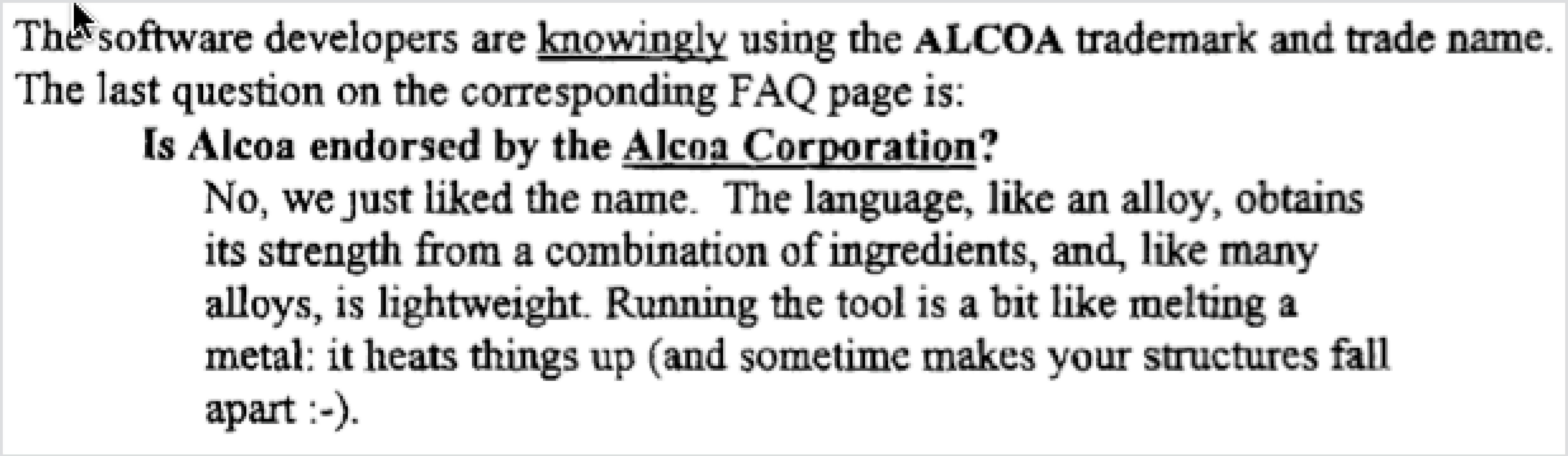
We have just learned that software developers at MIT have named a software program "Alcoa". This unauthorized usage of the **ALCOA** trademark on software on the MIT internet site is an infringement of the trademark rights of Alcoa Inc (Alcoa).

Alcoa has been using the trademark and service mark **ALCOA** on a wide variety of goods and services throughout the world since 1926. Through extensive sales and advertising our trademark and trade name **ALCOA** is famous worldwide. It is well associated with metal alloys.

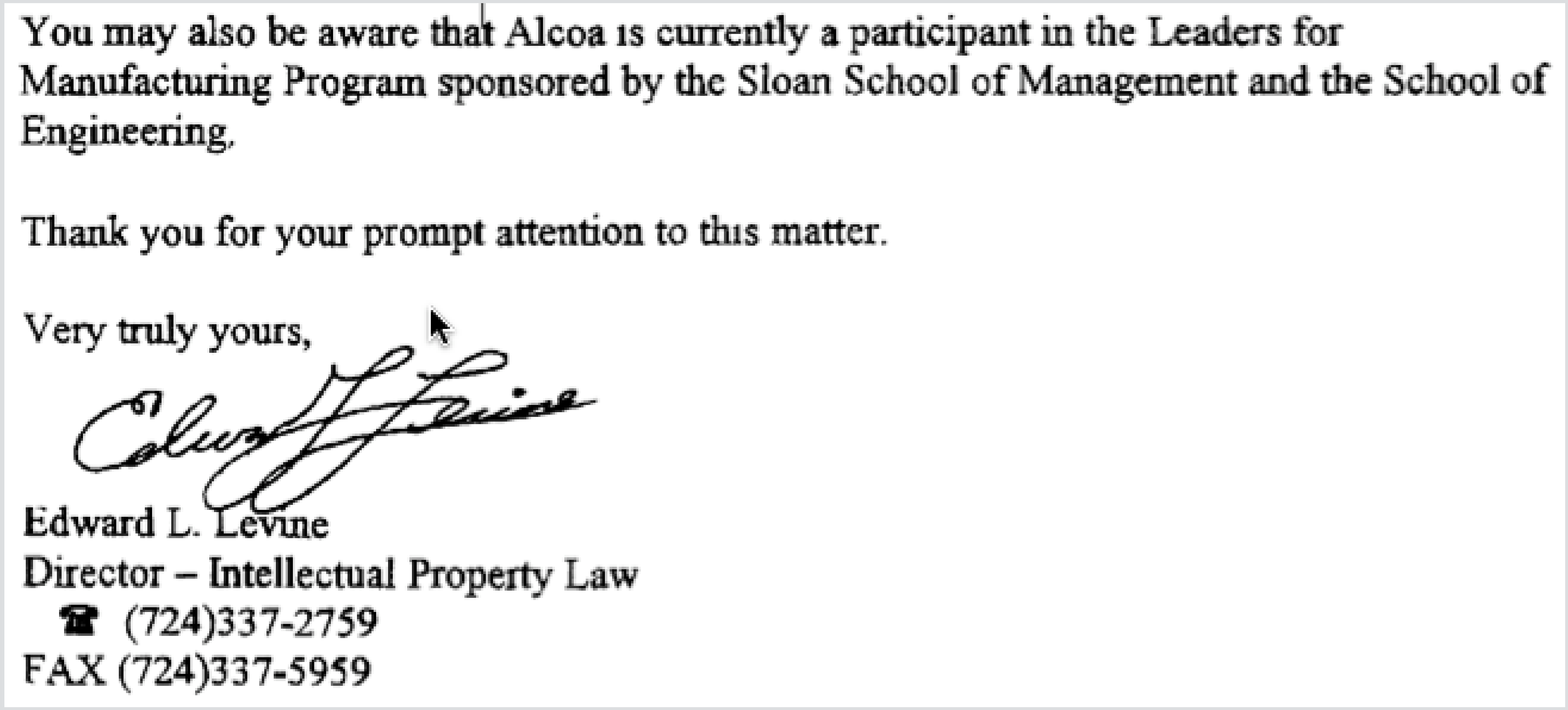


The software developers are knowingly using the **ALCOA** trademark and trade name. The last question on the corresponding FAQ page is:

> **Is Alcoa endorsed by the Alcoa Corporation?**
>
> No, we just liked the name. The language, like an alloy, obtains its strength from a combination of ingredients, and, like many alloys, is lightweight. Running the tool is a bit like melting a metal: it heats things up (and sometime makes your structures fall apart :-).


The software developers are <u>knowingly</u> using the **ALCOA** trademark and trade name. The last question on the corresponding FAQ page is:

> **Is Alcoa endorsed by the <u>Alcoa Corporation</u>?**
>
> No, we just liked the name. The language, like an alloy, obtains its strength from a combination of ingredients, and, like many alloys, is lightweight. Running the tool is a bit like melting a metal: it heats things up (and sometime makes your structures fall apart :-).


You may also be aware that Alcoa is currently a participant in the Leaders for Manufacturing Program sponsored by the Sloan School of Management and the School of Engineering.

Thank you for your prompt attention to this matter.

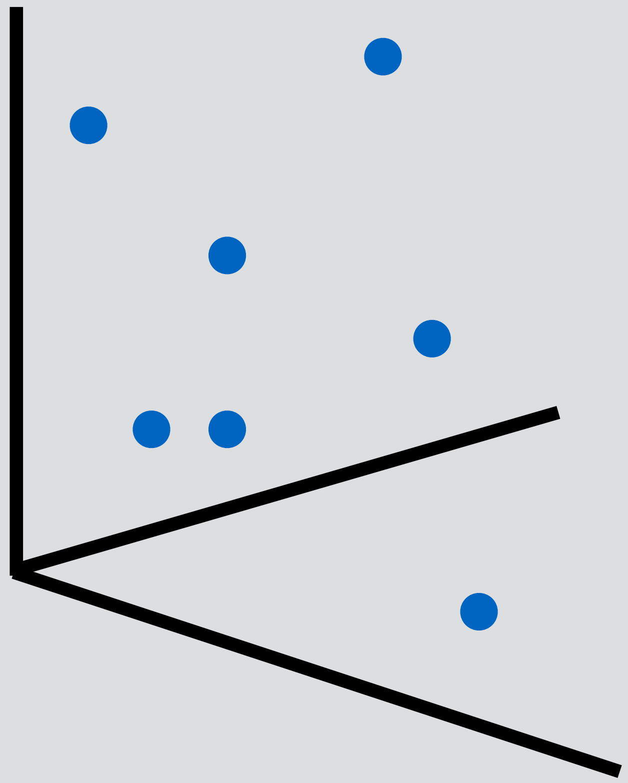
Very truly yours,

[illegible]

Edward L. Levine
Director – Intellectual Property Law
☎ (724)337-2759
FAX (724)337-5959

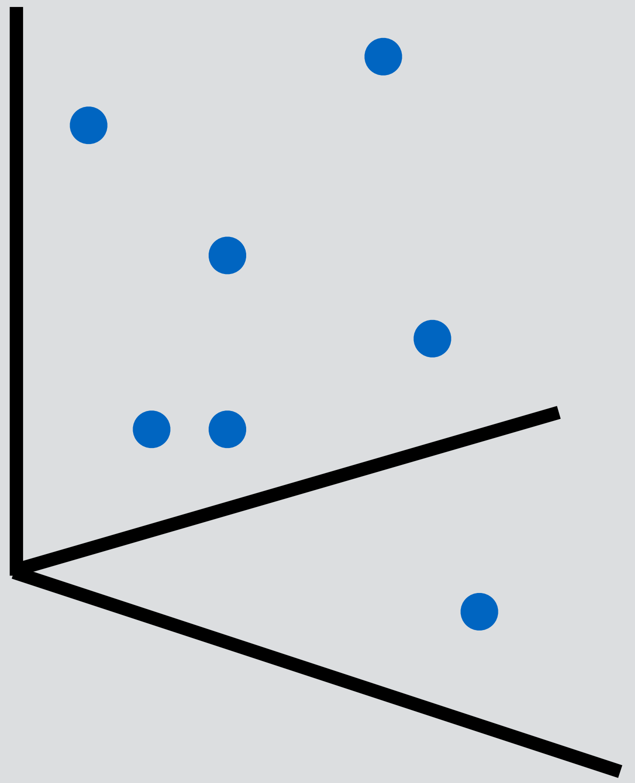
5
ideas

all small tests

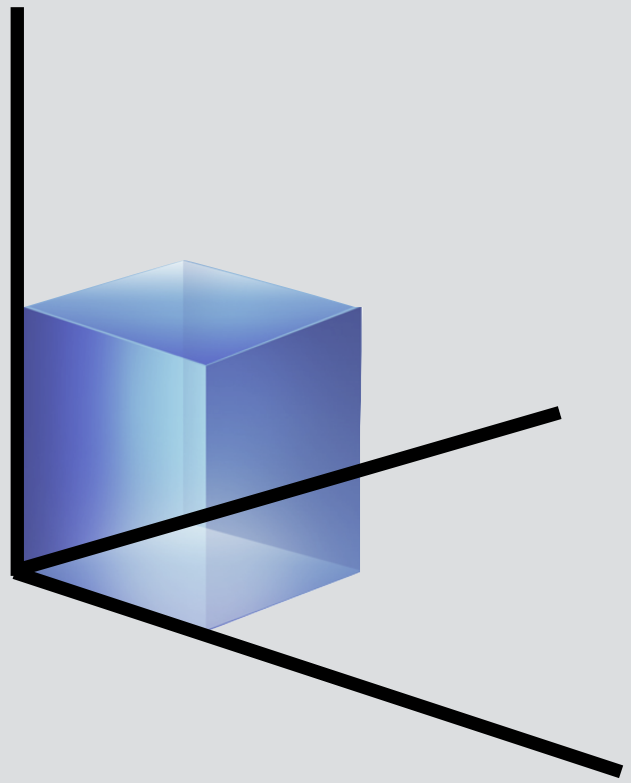


traditional testing

all small tests

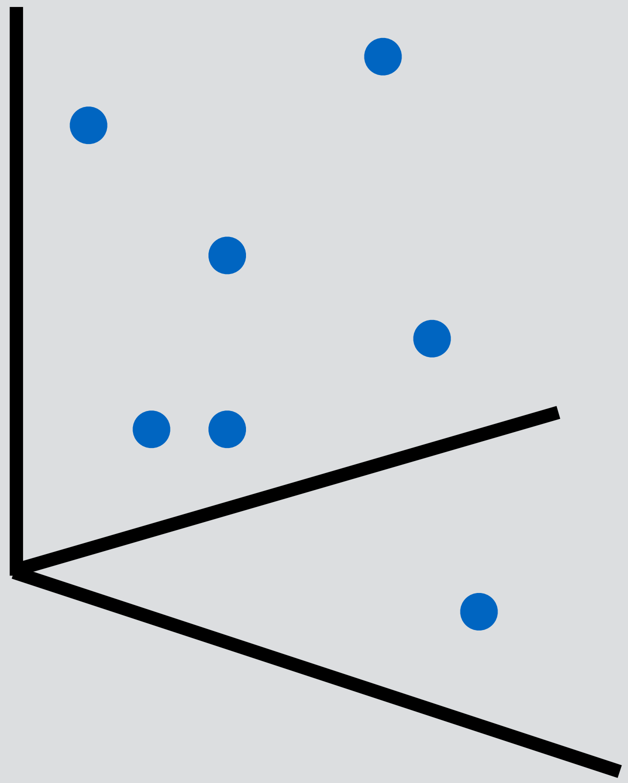


traditional testing

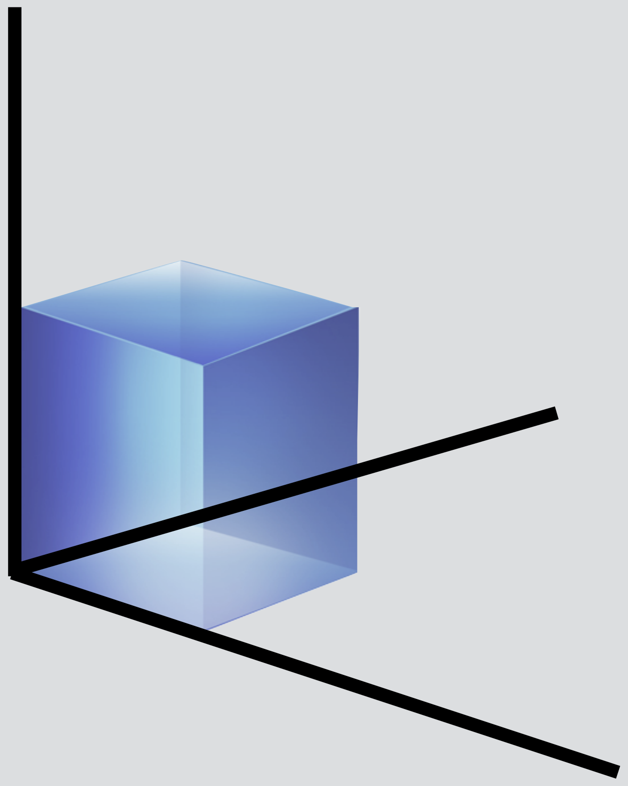


bounded analysis

all small tests



traditional testing



bounded analysis

5 users, calls, devices
2^{25} user-call, user-device relations
so $2^{50} = 10^{15}$ states

a signature style

```
sig Call {members: set User}
```

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id\colon CallId \times members\colon \mathbb{P}\ User)$
$User = \mathbb{P}\ (id\colon UserId \times talking\colon \mathbb{P}\ User)$

traditional interpretation

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id\colon CallId \times members\colon \mathbb{P}\ User)$
$User = \mathbb{P}\ (id\colon UserId \times talking\colon \mathbb{P}\ User)$

traditional interpretation

```
all c, c’: Call {no c.members & c’.members}
```

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id: CallId \times members: \mathbb{P}\ User)$
$User = \mathbb{P}\ (id: UserId \times talking: \mathbb{P}\ User)$

traditional interpretation

```
all c, c': Call {no c.members & c'.members}
```

$\forall\ c, c' \in \mathbb{P}\ (id: CallId \times members: \mathbb{P}\ User) \mid \ldots$

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id\!: CallId \times members\!: \mathbb{P}\ User)$
$User = \mathbb{P}\ (id\!: UserId \times talking\!: \mathbb{P}\ User)$

traditional interpretation

```
all c, c': Call {no c.members & c'.members}
```

$\forall\ c, c' \in\ \mathbb{P}\ (id\!: CallId \times members\!:\ \mathbb{P}\ User) \mid \ldots$

higher order quantification: ouch!

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id: CallId \times members: \mathbb{P}\ User)$
$User = \mathbb{P}\ (id: UserId \times talking: \mathbb{P}\ User)$

traditional interpretation

$Call, User: \mathbb{P}\, Univ$
$members: Call \leftrightarrow User$

Alloy interpretation

```
all c, c': Call {no c.members & c'.members}
```

$\forall\ c, c' \in\ \mathbb{P}\ (id: CallId \times members:\ \mathbb{P}\ User) \mid \ldots$

higher order quantification: ouch!

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id: CallId \times members: \mathbb{P}\ User)$
$User = \mathbb{P}\ (id: UserId \times talking: \mathbb{P}\ User)$

traditional interpretation

$Call, User: \mathbb{P}\ Univ$
$members: Call \leftrightarrow User$

Alloy interpretation

```
all c, c′: Call {no c.members & c′.members}
```

$\forall\ c, c' \in \mathbb{P}\ (id: CallId \times members: \mathbb{P}\ User)\ |\ \ldots$

higher order quantification: ouch!

$\exists\ members: Call \leftrightarrow User\ |$
$\quad \forall\ c, c' \in Call\ |\ \ldots$

a signature style

```
sig Call {members: set User}
```

$Call = \mathbb{P}\ (id{:}\ CallId \times members{:}\ \mathbb{P}\ User)$
$User = \mathbb{P}\ (id{:}\ UserId \times talking{:}\ \mathbb{P}\ User)$

traditional interpretation

$Call, User{:}\ \mathbb{P}\, Univ$
$members{:}\ Call \leftrightarrow User$

Alloy interpretation

```
all c, c': Call {no c.members & c'.members}
```

$\forall\ c, c' \in\ \mathbb{P}\ (id{:}\ CallId \times members{:}\ \mathbb{P}\ User) \mid \ldots$

higher order quantification: ouch!

$\exists\ members{:}\ Call \leftrightarrow User \mid$
$\quad \forall\ c, c' \in Call \mid \ldots$

first order quantification: solve with SAT

everything's a relation

```
sig Call {members: set User}
sig User {talking: set User}
```

everything's a relation

```
sig Call {members: set User}
sig User {talking: set User}
```

some expressions:

```
c.members
members.u
u.talking
c.members.talking
u.talking = u'
```

everything's a relation

```
sig Call {members: set User}
```

```
sig User {talking: set User}
```

some expressions:

```
c.members
members.u
u.talking
c.members.talking
u.talking = u'
```

navigation: dot is just join, not overloaded

everything's a relation

```
sig Call {members: set User}
sig User {talking: set User}
```

some expressions:

```
c.members
members.u
u.talking
c.members.talking
u.talking = u'
```

navigation: dot is just join, not overloaded

no syntax difference: fun vs relation

everything's a relation

```
sig Call {members: set User}
sig User {talking: set User}
```

some expressions:

```
c.members
members.u
u.talking
c.members.talking
u.talking = u'
```

navigation: dot is just join, not overloaded

no syntax difference: fun vs relation

no undefined value, follows Parnas

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

u	talking	u.talking
{(U0)}	{U0,U1), (U1, U0)}	{(U0), (U1)}

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

U0	1
U1	0
U2	0

u	talking	u.talking
{(U0)}	{U0,U1), (U1, U0)}	{(U0), (U1)}

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

U0	1
U1	0
U2	0

u

{(U0)}

	U0	*U1*	*U2*
U0	0	1	0
U1	1	0	0
U2	0	0	0

talking

{U0,U1), (U1, U0)}

u.talking

{(U0), (U1)}

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

$$\begin{array}{c|c|}
U0 & 1 \\
U1 & 0 \\
U2 & 0
\end{array}$$

u

{(U0)}

$$\begin{array}{c|c|c|c|}
 & U0 & U1 & U2 \\
U0 & 0 & 1 & 0 \\
U1 & 1 & 0 & 0 \\
U2 & 0 & 0 & 0
\end{array}$$

talking

{U0,U1), (U1, U0)}

$$\begin{array}{c|c|}
U0 & 0 \\
U1 & 1 \\
U2 & 0
\end{array}$$

u.talking

{(U0), (U1)}

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

u

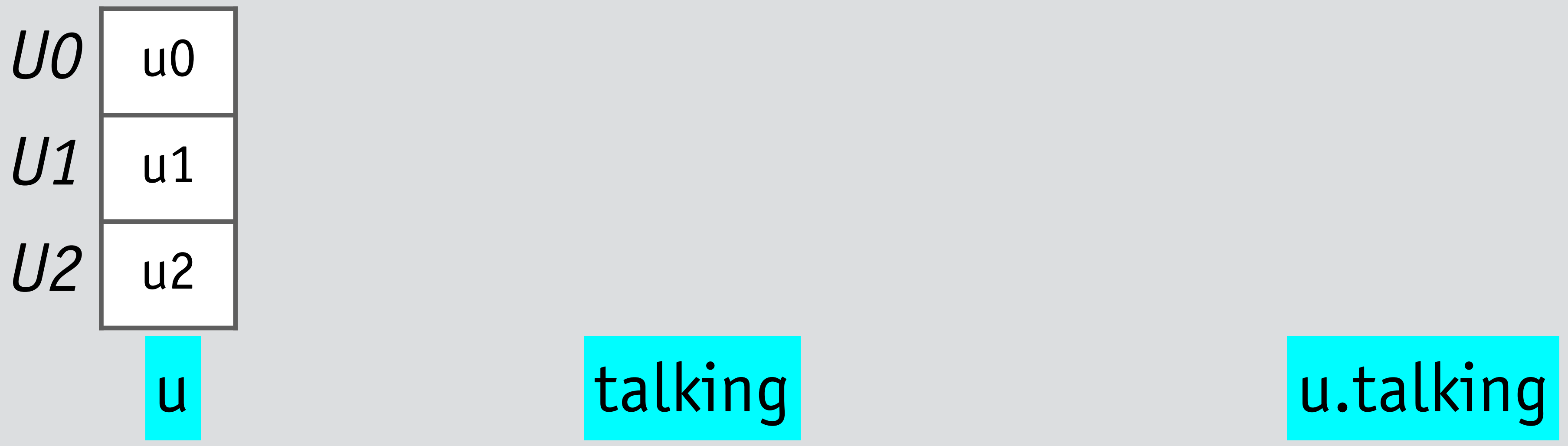
talking

u.talking

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```



getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

$$\begin{array}{c|c} & \\ U0 & u0 \\ U1 & u1 \\ U2 & u2 \end{array}$$

u

$$\begin{array}{c|ccc} & U0 & U1 & U2 \\ \hline U0 & t00 & t01 & t02 \\ U1 & t10 & t11 & t12 \\ U2 & t20 & t21 & t22 \end{array}$$

talking

u.talking

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

U0	u0
U1	u1
U2	u2

u

	U0	*U1*	*U2*
U0	t00	t01	t02
U1	t10	t11	t12
U2	t20	t21	t22

talking

U0	$(u0 \wedge t00) \vee (u1 \wedge t10) \vee (u2 \wedge t20)$
U1	$(u1 \wedge t01) \vee (u1 \wedge t11) \vee (u2 \wedge t21)$
U2	$(u0 \wedge t02) \vee (u1 \wedge t12) \vee (u2 \wedge t22)$

u.talking

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

U0	u0
U1	u1
U2	u2

u

	U0	*U1*	*U2*
U0	t00	t01	t02
U1	t10	t11	t12
U2	t20	t21	t22

talking

U0	$(u0 \wedge t00) \vee (u1 \wedge t10) \vee (u2 \wedge t20)$
U1	$(u1 \wedge t01) \vee (u1 \wedge t11) \vee (u2 \wedge t21)$
U2	$(u0 \wedge t02) \vee (u1 \wedge t12) \vee (u2 \wedge t22)$

u.talking

some u: User | u **in** u.talking

getting satisfaction

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

$$\begin{array}{c|c|} U0 & u0 \\ U1 & u1 \\ U2 & u2 \end{array} \qquad \begin{array}{c|c|c|c|} & U0 & U1 & U2 \\ U0 & t00 & t01 & t02 \\ U1 & t10 & t11 & t12 \\ U2 & t20 & t21 & t22 \end{array} \qquad \begin{array}{c|c|} U0 & (u0 \wedge t00) \vee (u1 \wedge t10) \vee (u2 \wedge t20) \\ U1 & (u1 \wedge t01) \vee (u1 \wedge t11) \vee (u2 \wedge t21) \\ U2 & (u0 \wedge t02) \vee (u1 \wedge t12) \vee (u2 \wedge t22) \end{array}$$

u talking u.talking

$$\begin{array}{ll} u0 \Rightarrow (u0 \wedge t00) \vee (u1 \wedge t10) \vee (u2 \wedge t20) & \wedge \\ u1 \Rightarrow (u1 \wedge t01) \vee (u1 \wedge t11) \vee (u2 \wedge t21) & \wedge \\ u2 \Rightarrow (u0 \wedge t02) \vee (u1 \wedge t12) \vee (u2 \wedge t22) & \end{array}$$

some u: User | u **in** u.talking

getting satisfaction

add symmetry breaking predicates too

```
sig User {talking: set User}
```

```
check {no u: User | u in u.talking}
```

$$\begin{array}{c|c|}
 & \\
\hline
U0 & u0 \\
\hline
U1 & u1 \\
\hline
U2 & u2 \\
\hline
\end{array}$$

u

$$\begin{array}{c|c|c|c|}
 & U0 & U1 & U2 \\
\hline
U0 & t00 & t01 & t02 \\
\hline
U1 & t10 & t11 & t12 \\
\hline
U2 & t20 & t21 & t22 \\
\hline
\end{array}$$

talking

$$\begin{array}{c|c|}
 & \\
\hline
U0 & (u0 \wedge t00) \vee (u1 \wedge t10) \vee (u2 \wedge t20) \\
\hline
U1 & (u1 \wedge t01) \vee (u1 \wedge t11) \vee (u2 \wedge t21) \\
\hline
U2 & (u0 \wedge t02) \vee (u1 \wedge t12) \vee (u2 \wedge t22) \\
\hline
\end{array}$$

u.talking

$$\begin{array}{ll}
u0 \Rightarrow (u0 \wedge t00) \vee (u1 \wedge t10) \vee (u2 \wedge t20) & \wedge \\
u1 \Rightarrow (u1 \wedge t01) \vee (u1 \wedge t11) \vee (u2 \wedge t21) & \wedge \\
u2 \Rightarrow (u0 \wedge t02) \vee (u1 \wedge t12) \vee (u2 \wedge t22) &
\end{array}$$

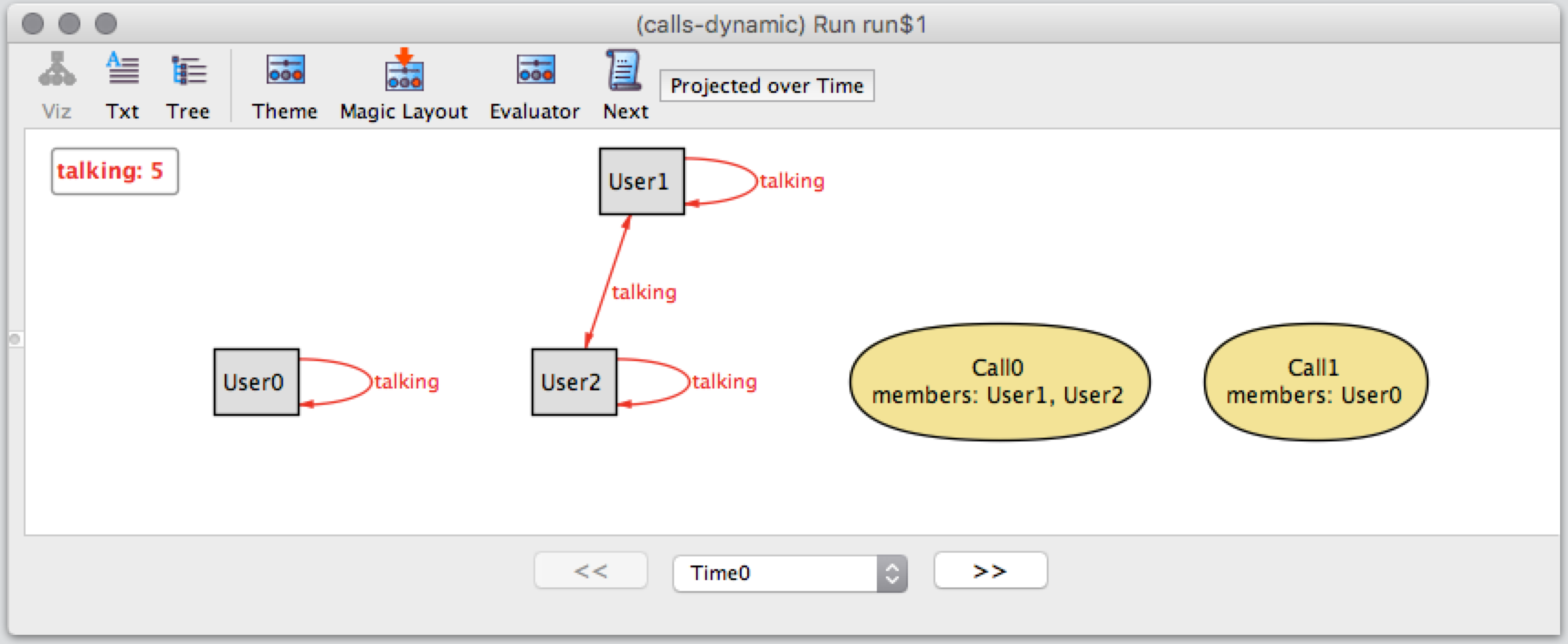
some u: User | u **in** u.talking

roll your own idiom

```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact { all t: Time | let m = members.t | talking.t = ~m.m }
```

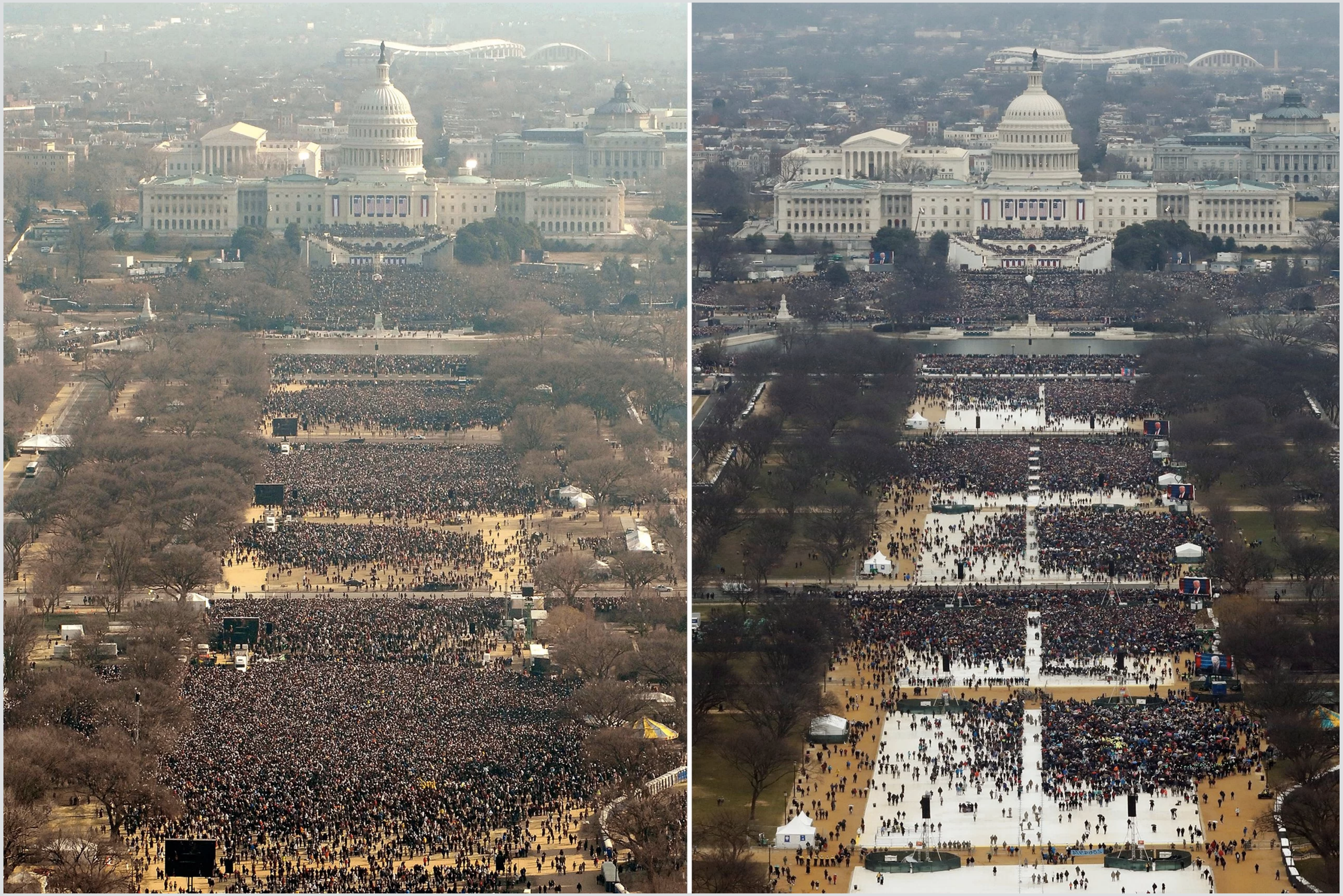
roll your own idiom

```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact { all t: Time | let m = members.t | talking.t = ~m.m }
```

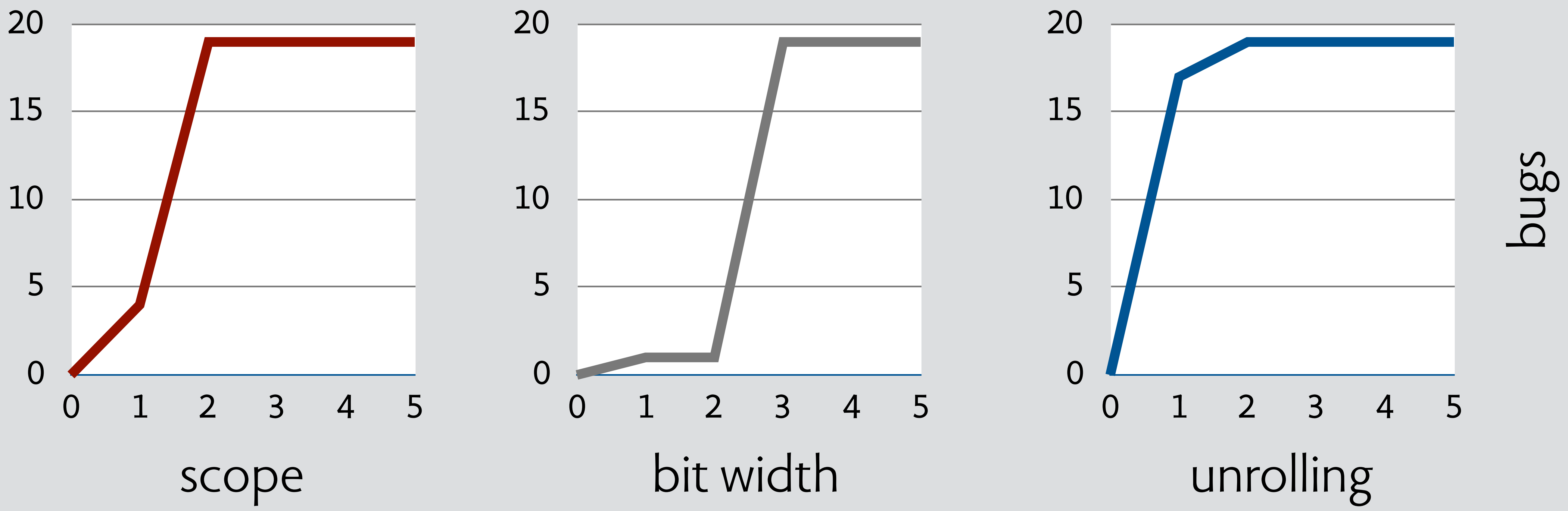


4 outcomes

but does it work? tell us the truth!



are small scopes enough?



analysis of KOA voting code
19 methods violating specs
how many bugs found in scope of k?
[Greg Dennis, 2008]

most bugs in small scopes?

yes, but two caveats
integers are nasty: ‘special’ semantics
trace length must be set higher

why traces are tricky
in scope 5, call-user has ≤ 25 pairs
can check an operation on 2^{25} pre-states
but if initially empty, 25 steps to populate?

is first order enough?

is first order enough?

converting Z (eg) to Alloy
generally straightforward

is first order enough?

converting Z (eg) to Alloy
generally straightforward

Mondex smart card system
NatWest, Oxford U., Logica
[Ramananandro]

is first order enough?

converting Z (eg) to Alloy
generally straightforward

Mondex smart card system
NatWest, Oxford U., Logica
[Ramananandro]

The
Tokeneer
Project
A hands on look at an NSA funded, highly secure biometric software system built in SPARK 2005.

Tokeneer project
Praxis/NSA
50pp Z, 1200 lines Alloy
[Eunsuk Kang]

is first order enough?

converting Z (eg) to Alloy
generally straightforward

minimization may be OK
send packet to nearest neighbor?
easy: just say no shorter option

Mondex smart card system
NatWest, Oxford U., Logica
[Ramananandro]

The Tokeneer Project
A hands on look at an NSA funded, highly secure biometric software system built in SPARK 2005.

Tokeneer project
Praxis/NSA
50pp Z, 1200 lines Alloy
[Eunsuk Kang]

is first order enough?

converting Z (eg) to Alloy
generally straightforward

minimization may be OK
send packet to nearest neighbor?
easy: just say no shorter option

synthesis is higher order
find a program without bugs
∃ p: Program | ∀ s: State | S(p,s)
this motivated Alloy* [Millicevic+]

Mondex smart card system
NatWest, Oxford U., Logica
[Ramananandro]

The Tokeneer Project
A hands on look at an NSA funded, highly secure biometric software system built in SPARK 2005.

Tokeneer project
Praxis/NSA
50pp Z, 1200 lines Alloy
[Eunsuk Kang]

was purity a good idea?

on the one hand
breadth of domains
nice translation target
good for teaching logic

on the other hand
dynamic idioms are complex
frame conditions annoying

was purity a good idea?

on the one hand
breadth of domains
nice translation target
good for teaching logic

on the other hand
dynamic idioms are complex
frame conditions annoying

Just this year, students used Alloy for a broad range of unexpected topics including:

- checking theorems about groups
- generating Feynman Diagrams
- modeling Facebook privacy

Tim Nelson, talking about his Brown course, Logic for Systems

is declarative spec easy?

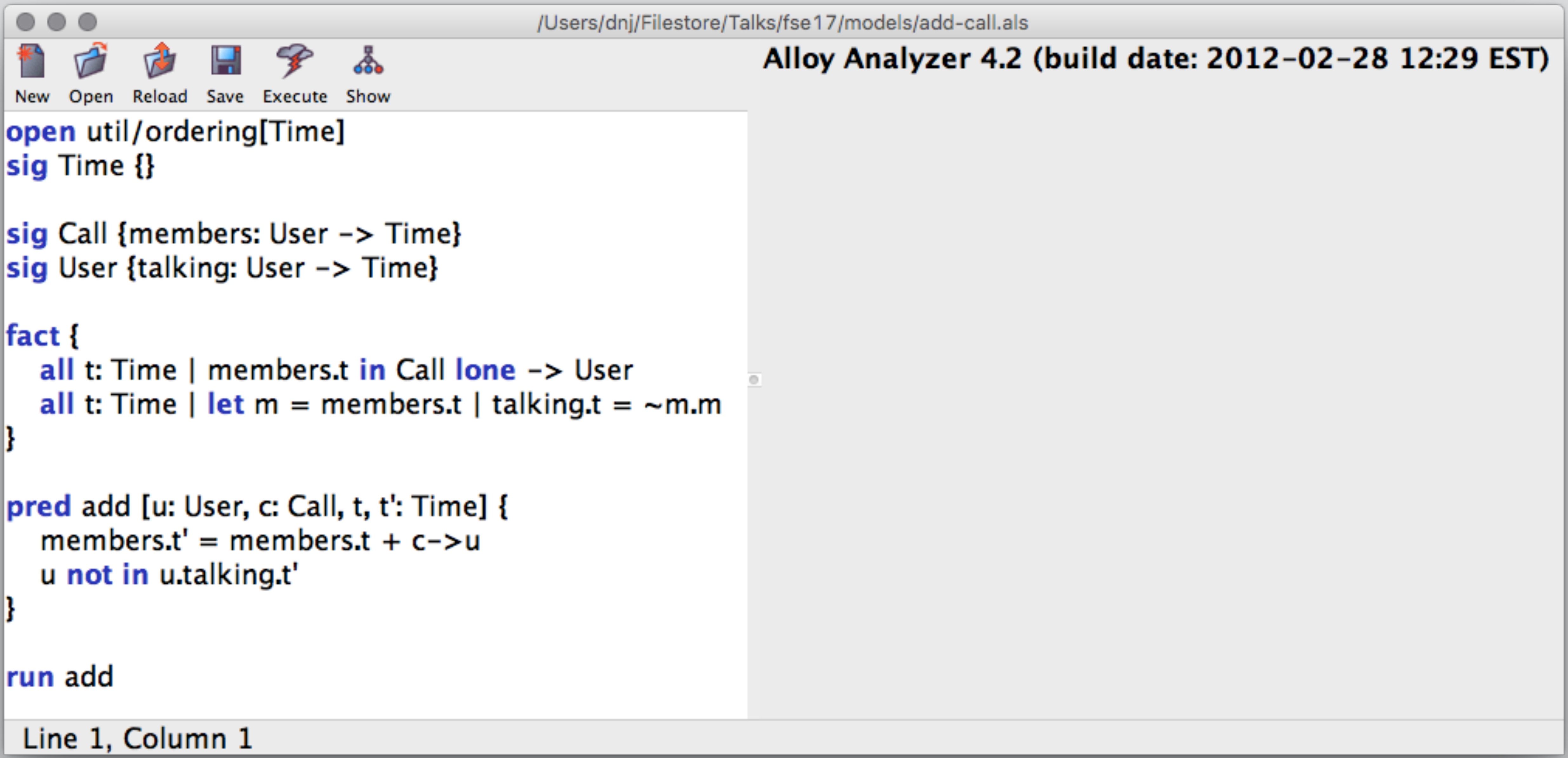
```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact {
  all t: Time | members.t in Call lone -> User
  all t: Time | let m = members.t | talking.t = ~m.m
}

pred add [u: User, c: Call, t, t’: Time] {
  members.t’ = members.t + c->u
  u not in u.talking.t’
}

run add
```

is declarative spec easy?

```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact {
 all t: Time | members.t in Call lone -> User
 all t: Time | let m = members.t | talking.t = ~m.m
}

pred add [u: User, c: Call, t, t’: Time] {
 members.t’ = members.t + c->u
 u not in u.talking.t’
}

run add
```

don’t end up talking to yourself

let's see what happens

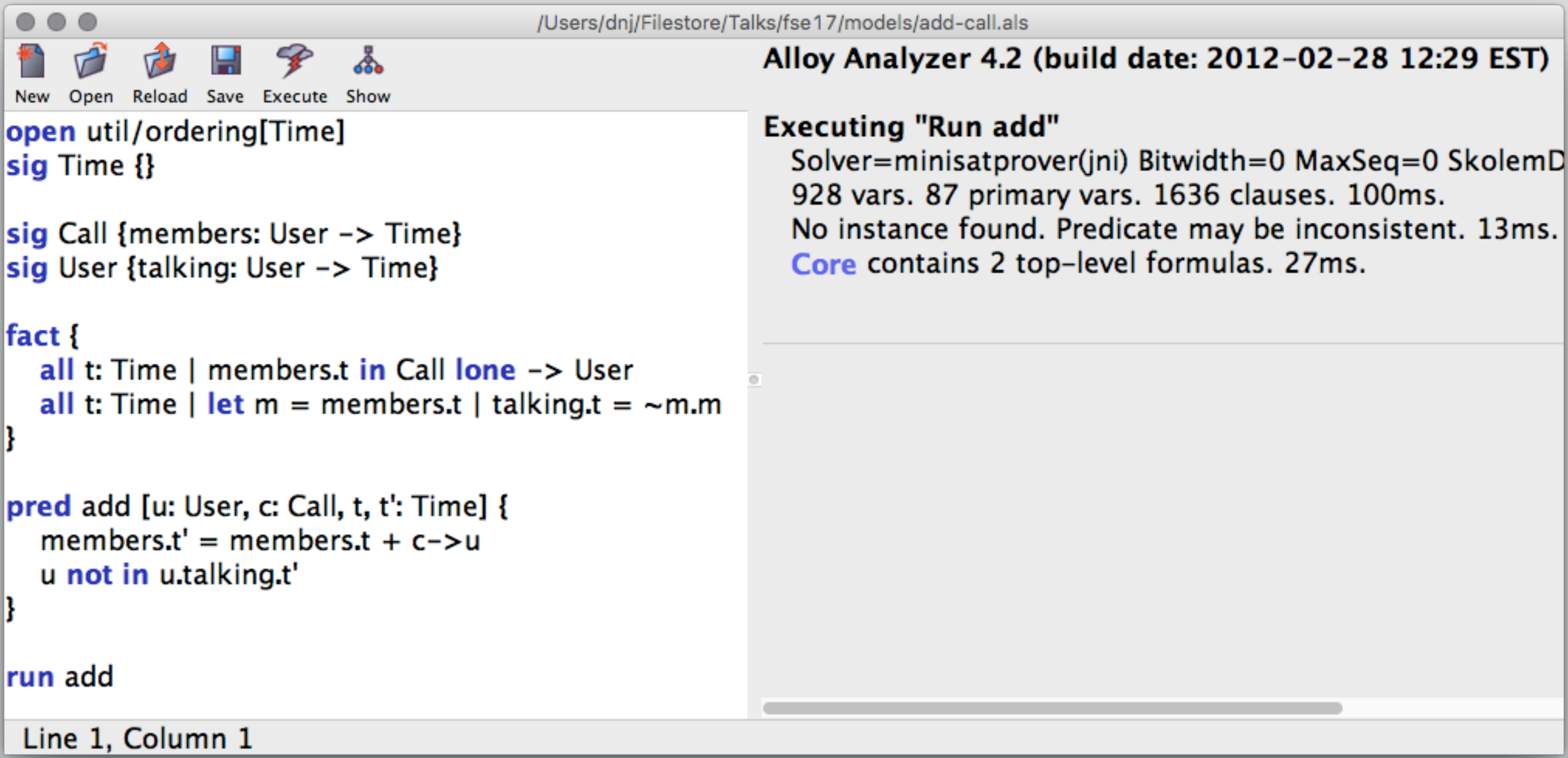


Alloy Analyzer 4.2 (build date: 2012-02-28 12:29 EST)

```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact {
   all t: Time | members.t in Call lone -> User
   all t: Time | let m = members.t | talking.t = ~m.m
}

pred add [u: User, c: Call, t, t': Time] {
   members.t' = members.t + c->u
   u not in u.talking.t'
}

run add
```

Line 1, Column 1

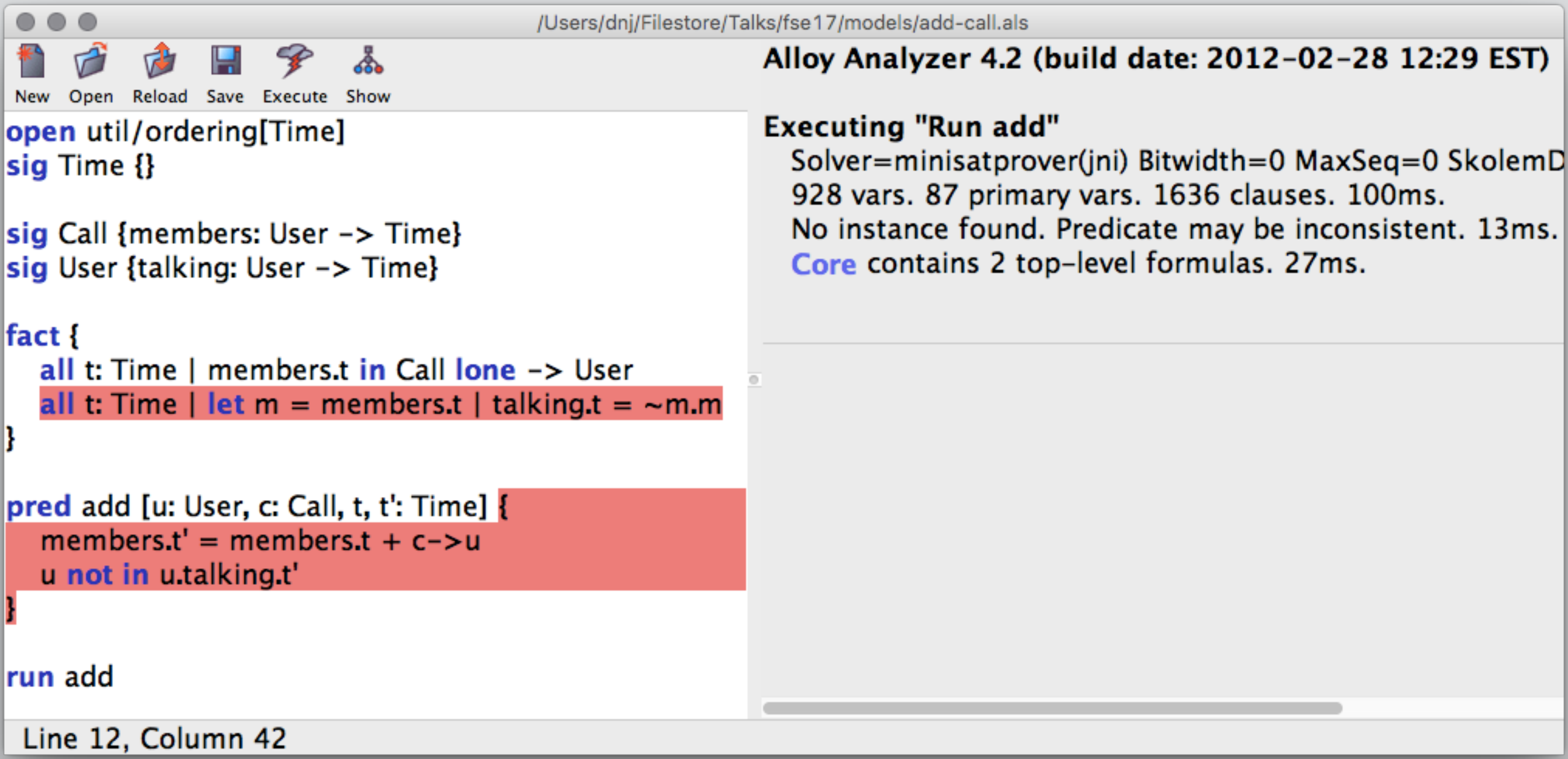
let's see what happens



```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact {
   all t: Time | members.t in Call lone -> User
   all t: Time | let m = members.t | talking.t = ~m.m
}

pred add [u: User, c: Call, t, t': Time] {
   members.t' = members.t + c->u
   u not in u.talking.t'
}

run add
```

```
Alloy Analyzer 4.2 (build date: 2012-02-28 12:29 EST)

Executing "Run add"
   Solver=minisatprover(jni) Bitwidth=0 MaxSeq=0 SkolemD
   928 vars. 87 primary vars. 1636 clauses. 100ms.
   No instance found. Predicate may be inconsistent. 13ms.
   Core contains 2 top-level formulas. 27ms.
```

let's see what happens

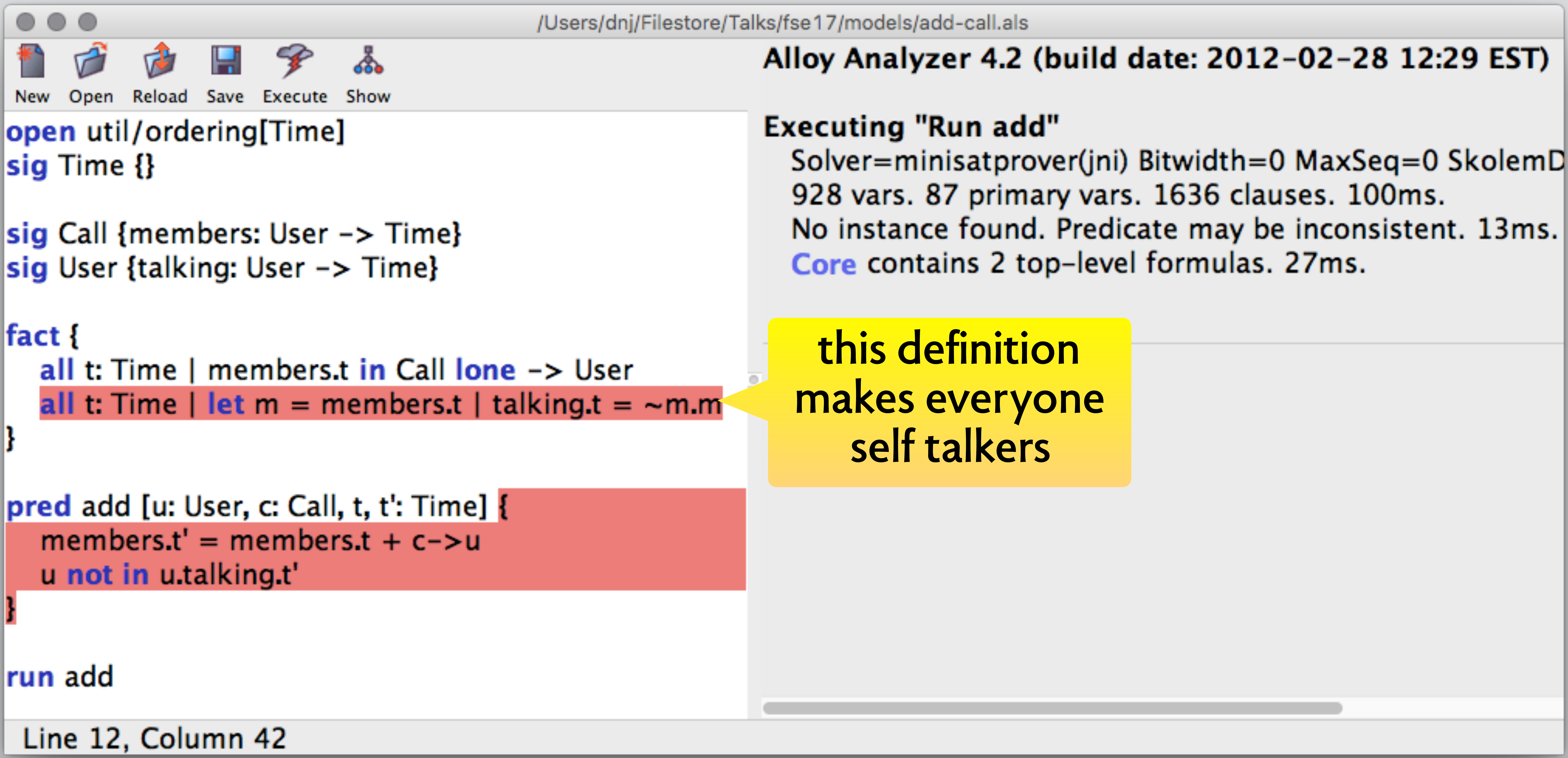


```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact {
   all t: Time | members.t in Call lone -> User
   all t: Time | let m = members.t | talking.t = ~m.m
}

pred add [u: User, c: Call, t, t': Time] {
   members.t' = members.t + c->u
   u not in u.talking.t'
}

run add
```

Line 12, Column 42

```
Alloy Analyzer 4.2 (build date: 2012-02-28 12:29 EST)

Executing "Run add"
   Solver=minisatprover(jni) Bitwidth=0 MaxSeq=0 SkolemD
   928 vars. 87 primary vars. 1636 clauses. 100ms.
   No instance found. Predicate may be inconsistent. 13ms.
   Core contains 2 top-level formulas. 27ms.
```

let's see what happens



```
open util/ordering[Time]
sig Time {}

sig Call {members: User -> Time}
sig User {talking: User -> Time}

fact {
   all t: Time | members.t in Call lone -> User
   all t: Time | let m = members.t | talking.t = ~m.m
}

pred add [u: User, c: Call, t, t': Time] {
   members.t' = members.t + c->u
   u not in u.talking.t'
}

run add
```

```
Alloy Analyzer 4.2 (build date: 2012-02-28 12:29 EST)

Executing "Run add"
   Solver=minisatprover(jni) Bitwidth=0 MaxSeq=0 SkolemD
   928 vars. 87 primary vars. 1636 clauses. 100ms.
   No instance found. Predicate may be inconsistent. 13ms.
   Core contains 2 top-level formulas. 27ms.
```

Line 12, Column 42

this definition makes everyone self talkers

so what's the story?

declarative specification

can be magical
often very succinct
nice separation of concerns

so what's the story?

declarative specification

can be magical
often very succinct
nice separation of concerns

can be maddening
harder to learn than I knew
even harder to debug
unsat core not enough

20

projects

extending Alloy

expressiveness
Alloy*: higher-order quantifiers [Milicevic+]

temporal constructs
DynAlloy [Frias+], [Macedo+]

better scenarios
target instances [Cunha+]
Aluminum: minimal instances [Nelson+]

performance
separating configurations [Macedo+]
exploit previous analyses: Titanium [Bagheri+]
translation optimizations [Marinov+]

platforms
Eclipse [LeBerre], web client [Cunha+]

tools built on Alloy

code analysis
Forge [Dennis+], TACO [Galeotti+]

architecture
design space exploration [Bagheri+]
architectural styles [Garlan+]

security
Margrave: policy analysis [Fisler+]
Poirot: vulnerabilities due to platform choice [Kang+]

software defined networking
Flowlog [Nelson+]

checking theorems
Nitpick for Isabelle [Blanchette]

tools built on Alloy

code analysis
Forge [Dennis+], TACO [Galeotti+]

architecture
design space exploration [Bagheri+]
architectural styles [Garlan+]

security
Margrave: policy analysis [Fisler+]
Poirot: vulnerabilities due to platform choice [Kang+]

software defined networking
Flowlog [Nelson+]

checking theorems
Nitpick for Isabelle [Blanchette]

a small sample of amazing tools people have built

some favorite applications of Alloy

some favorite applications of Alloy

web security [Akhawe+]
reusable model of web platform
found 2 known and 3 new vulnerabilities

some favorite applications of Alloy

web security [Akhawe+]
reusable model of web platform
found 2 known and 3 new vulnerabilities

networking [Zave]
showed Chord violates all its invariants
designed a new version + invariant

some favorite applications of Alloy

web security [Akhawe+]
reusable model of web platform
found 2 known and 3 new vulnerabilities

networking [Zave]
showed Chord violates all its invariants
designed a new version + invariant

dependability cases [UW PLSE]
end-to-end analysis of neutron therapy

some favorite applications of Alloy

web security [Akhawe+]
reusable model of web platform
found 2 known and 3 new vulnerabilities

networking [Zave]
showed Chord violates all its invariants
designed a new version + invariant

dependability cases [UW PLSE]
end-to-end analysis of neutron therapy

memory models [Torlak+; Wickerson+, Dodds+, Lustig+]
validate and develop new memory models

some favorite applications of Alloy

web security [Akhawe+]
reusable model of web platform
found 2 known and 3 new vulnerabilities

in all cases, it's more than finding bugs

networking [Zave]
showed Chord violates all its invariants
designed a new version + invariant

dependability cases [UW PLSE]
end-to-end analysis of neutron therapy

memory models [Torlak+; Wickerson+, Dodds+, Lustig+]
validate and develop new memory models

3
lessons

invest in your tool

```
sig User {device: Device, calls: set Call}{
  no device implies no calls
  this in calls.users
}
```

invest in your tool

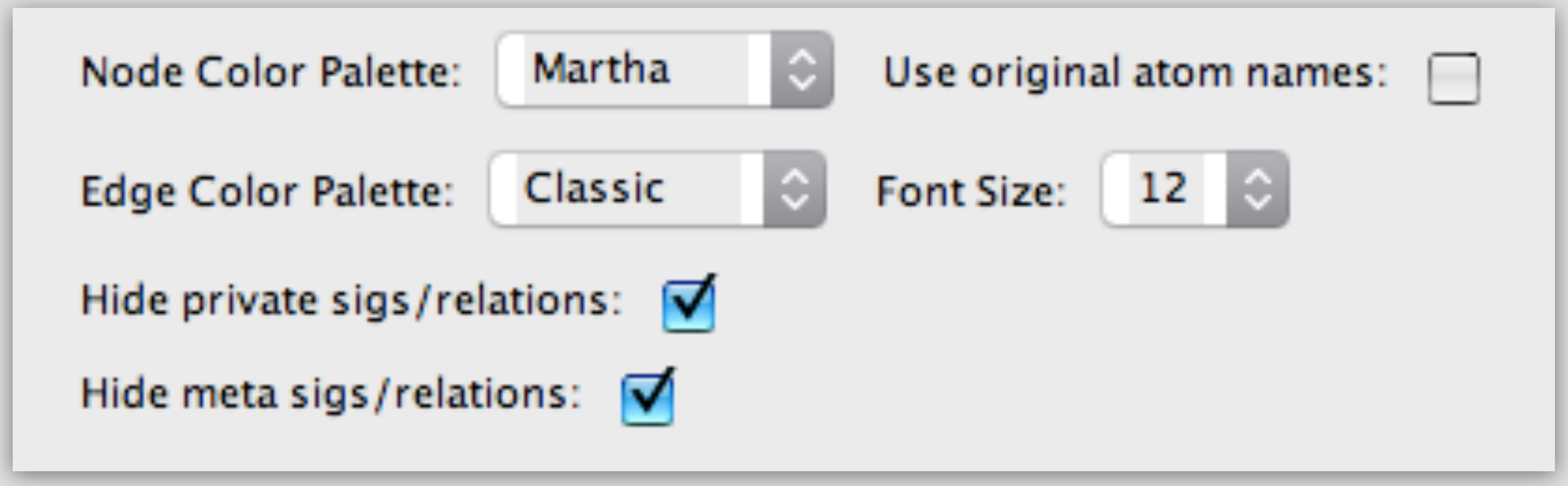
```
sig User {device: Device, calls: set Call}{
  no device implies no calls
  this in calls.users
}
```

look Ma, no semicolons!

invest in your tool

```
sig User {device: Device, calls: set Call}{
  no device implies no calls
  this in calls.users
}
```

look Ma, no semicolons!



Node Color Palette: Martha

Use original atom names: ☐

Edge Color Palette: Classic

Font Size: 12

Hide private sigs/relations: ☑

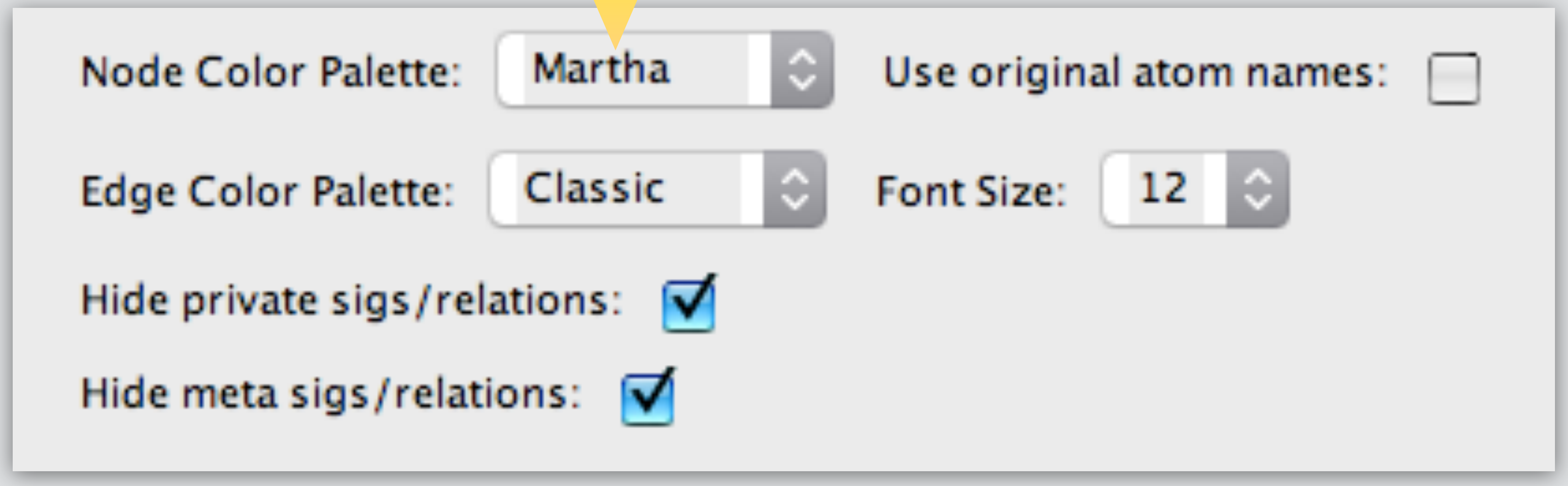
Hide meta sigs/relations: ☑

invest in your tool

```
sig User {device: Device, calls: set Call}{
  no device implies no calls
  this in calls.users
}
```

look Ma, no semicolons!

before she went to jail



Node Color Palette: Martha

Use original atom names: ☐

Edge Color Palette: Classic

Font Size: 12

Hide private sigs/relations: ☑

Hide meta sigs/relations: ☑

be nice (and objective)

a stupid thing I wrote:

"[In Z,] since declared sets cannot be used in subsequent declarations, simple multiplicity constraints must be written as additional textual formulas. The resulting specification is cluttered and unnatural."

be nice (and objective)

a stupid thing I wrote:

```
"[In Z,] since declared sets cannot be used in
subsequent declarations, simple multiplicity
constraints must be written as additional textual
formulas. The resulting specification is cluttered
and unnatural."
```

understandably aggrieved reviewer:

```
I suppose that I shouldn't be irritated by the
final sentence in this quote, but I am: what is
the measure of what is natural?  Anyway, the whole
thing is complete tosh…
```

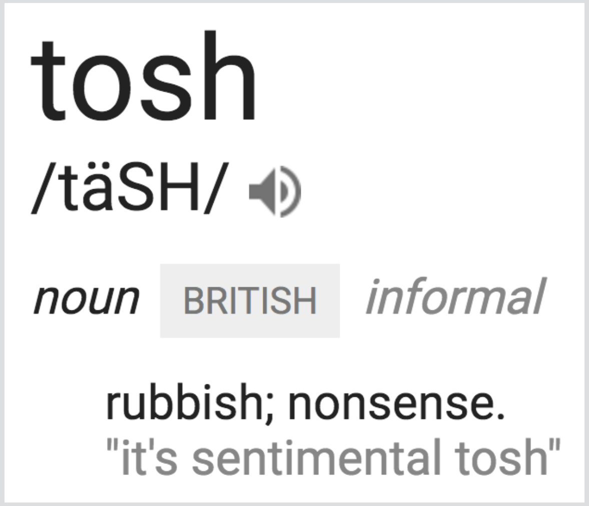
be nice (and objective)

a stupid thing I wrote:

"[In Z,] since declared sets cannot be used in subsequent declarations, simple multiplicity constraints must be written as additional textual formulas. The resulting specification is cluttered and unnatural."

understandably aggrieved reviewer:

I suppose that I shouldn't be irritated by the final sentence in this quote, but I am: what is the measure of what is natural? Anyway, the whole thing is complete tosh…

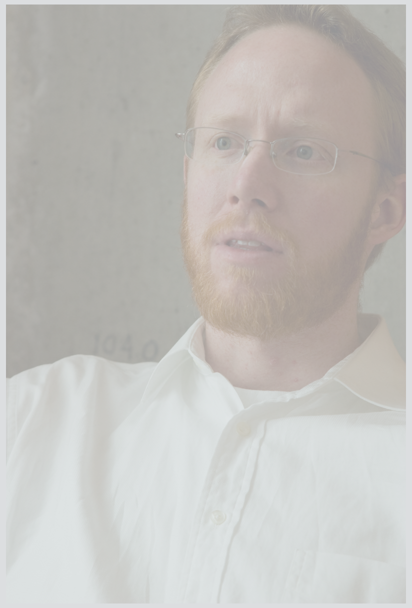

tosh
/täSH/

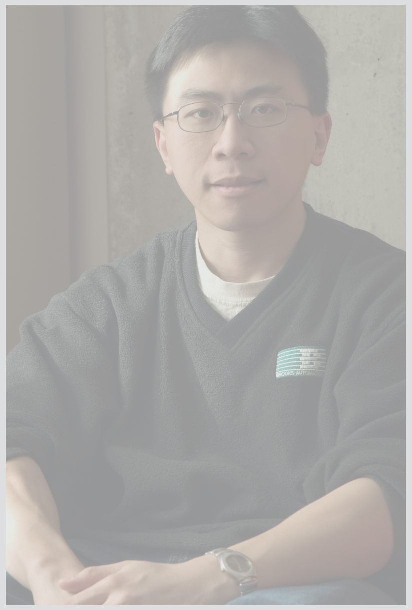
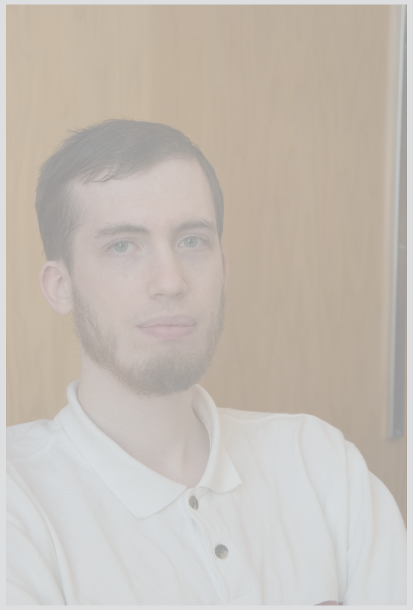
noun BRITISH *informal*

rubbish; nonsense.
"it's sentimental tosh"

get lucky!

get lucky!



3 thoughts

human factors

human factors

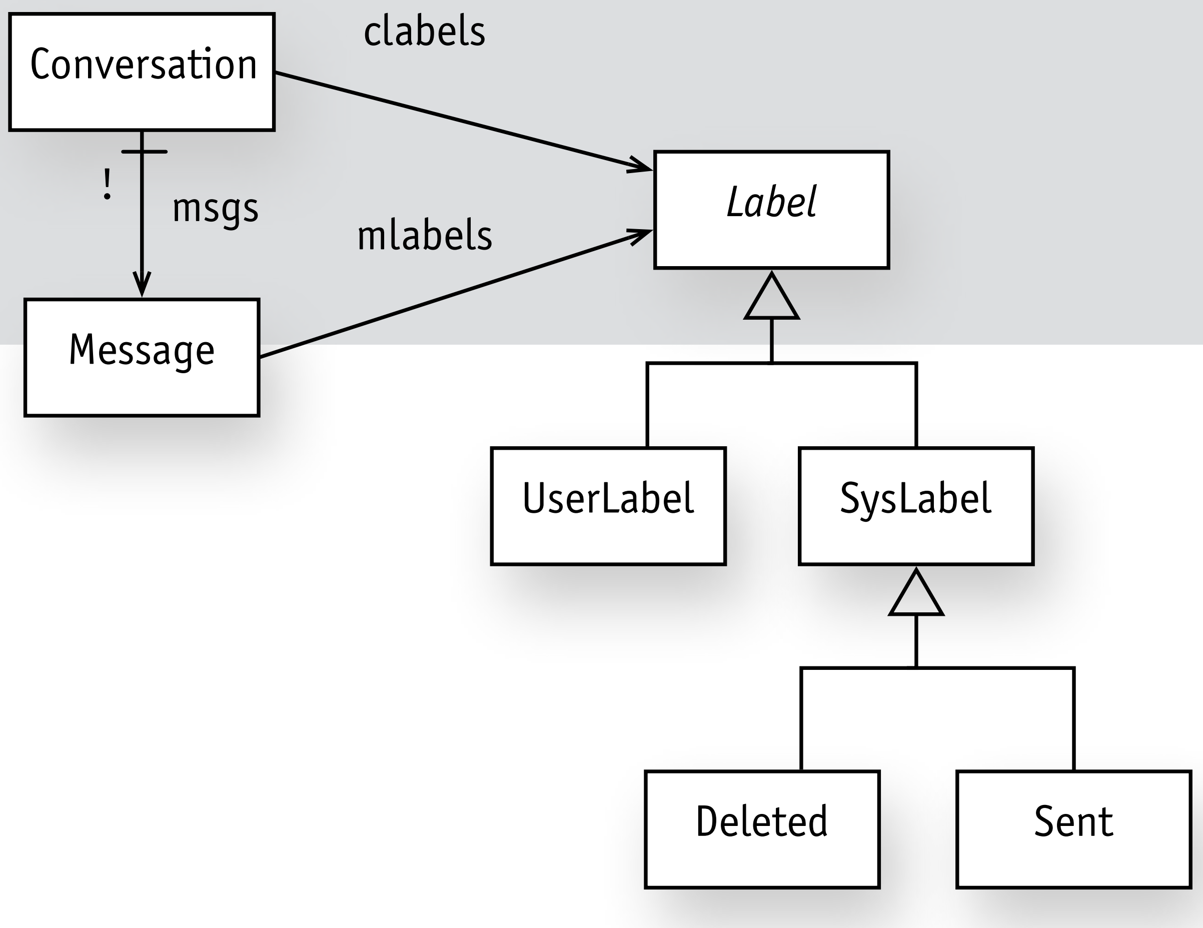
more emphasis needed
especially in formal methods

human factors

more emphasis needed
especially in formal methods

what I eventually figured out
abstraction is really hard
most programmers can’t draw an ER diagram
usual educational approaches don’t work

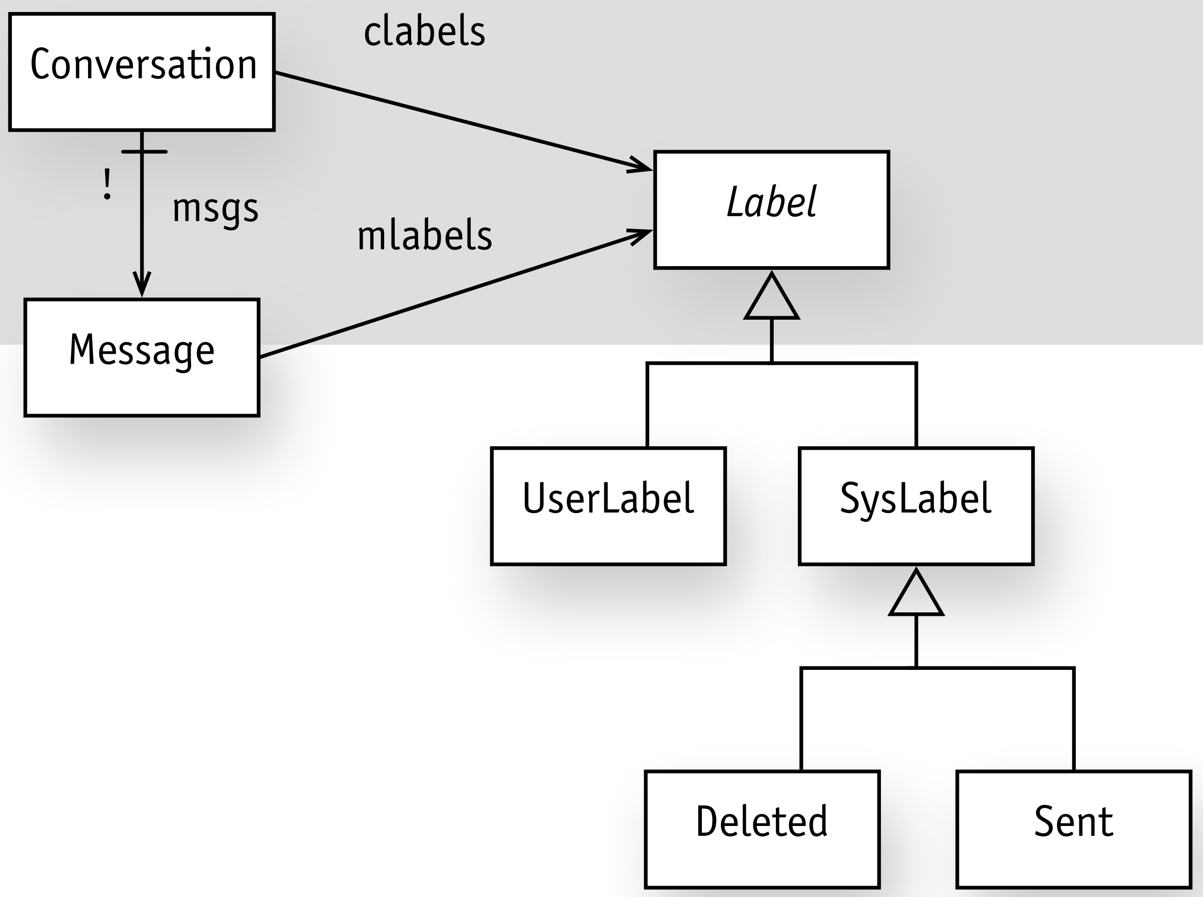
human factors



more emphasis needed
especially in formal methods

what I eventually figured out
abstraction is really hard
most programmers can’t draw an ER diagram
usual educational approaches don’t work

human factors

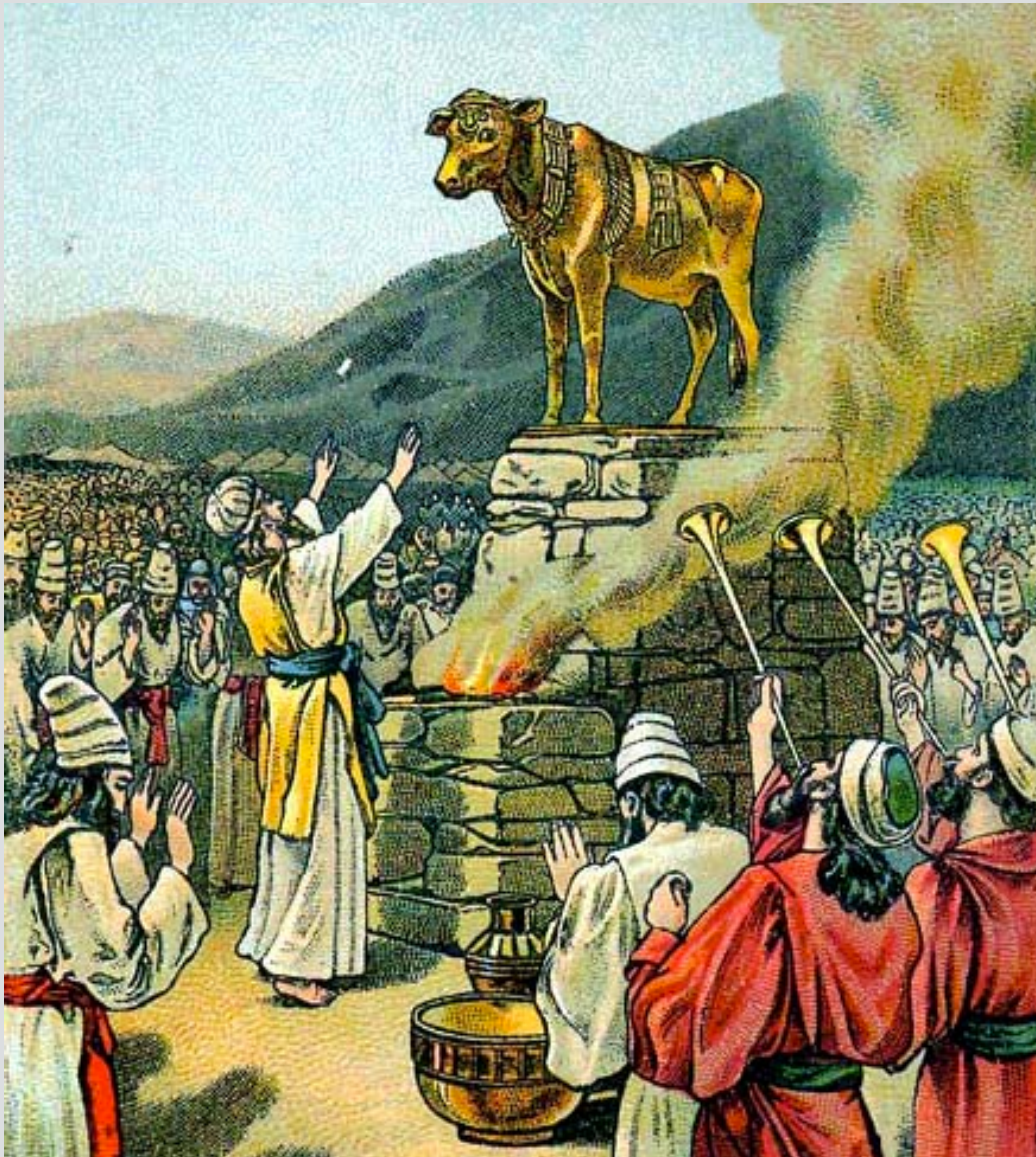


more emphasis needed
especially in formal methods

what I eventually figured out
abstraction is really hard
most programmers can't draw an ER diagram
usual educational approaches don't work

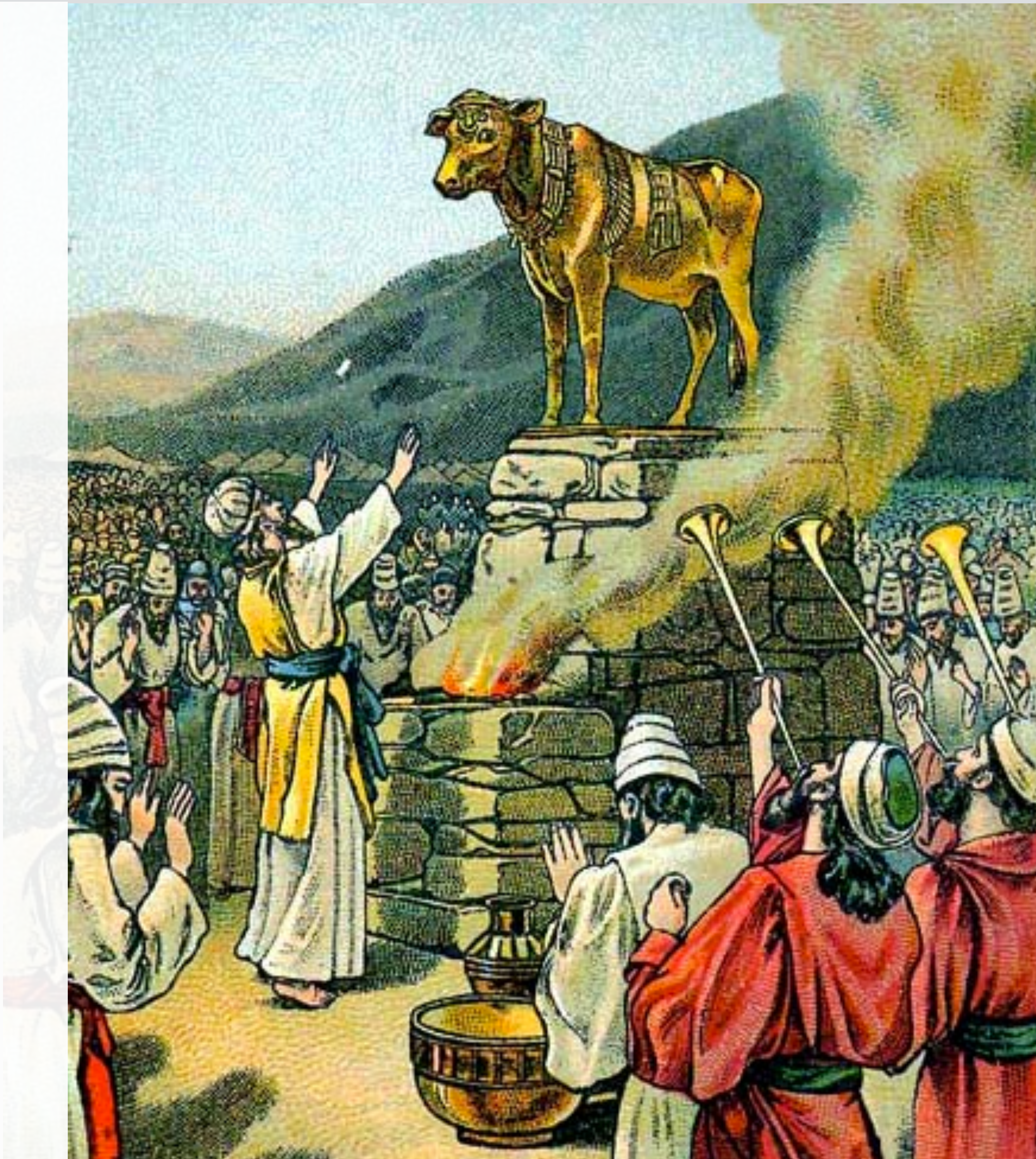
what if I'd studied this 20 years ago?
might not have changed Alloy
but might have changed my research direction?

on empiricism



on empiricism

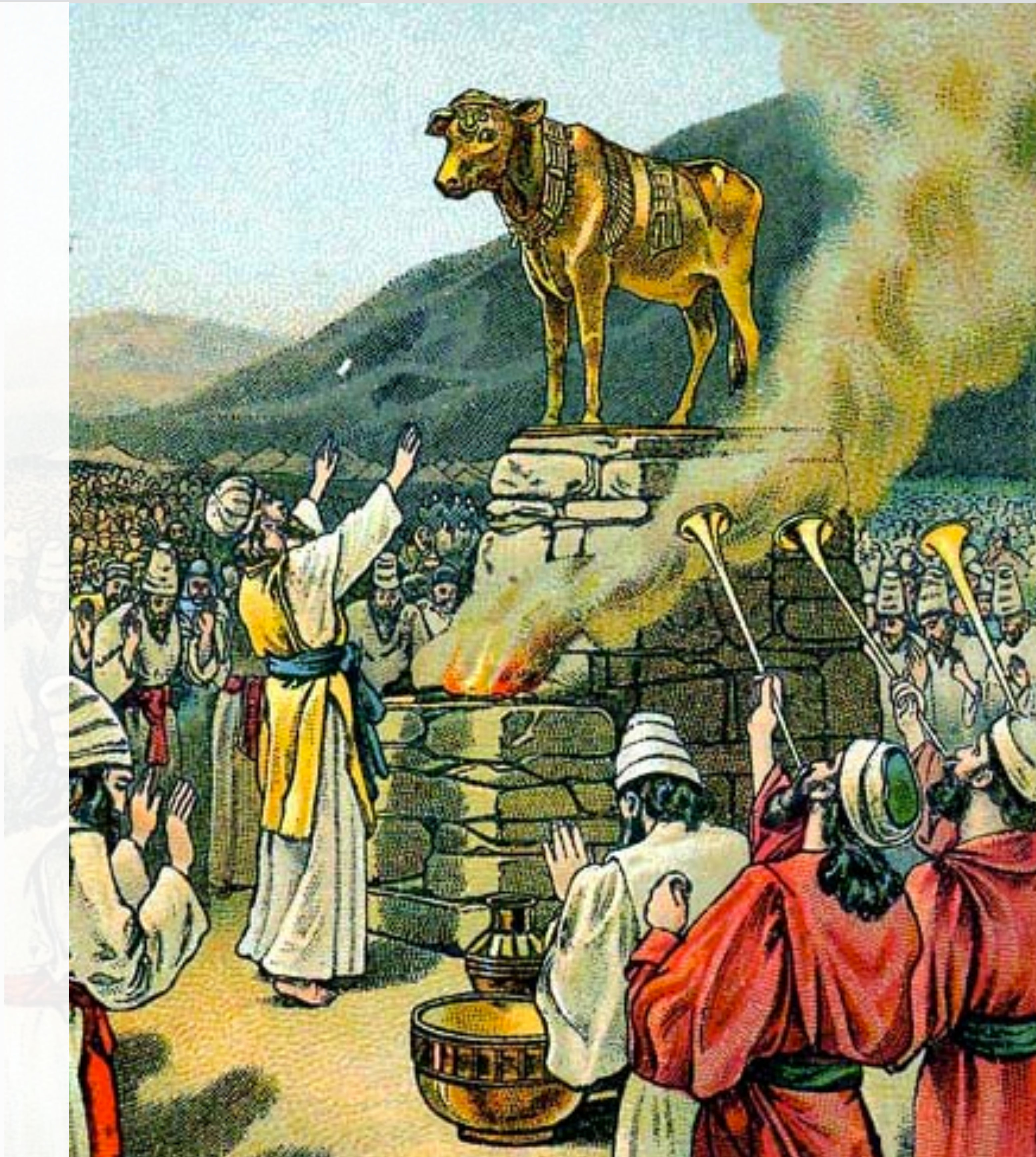
empirical research
exciting & powerful



on empiricism

empirical research
exciting & powerful

empirical validation
as sole arbiter: a mistake

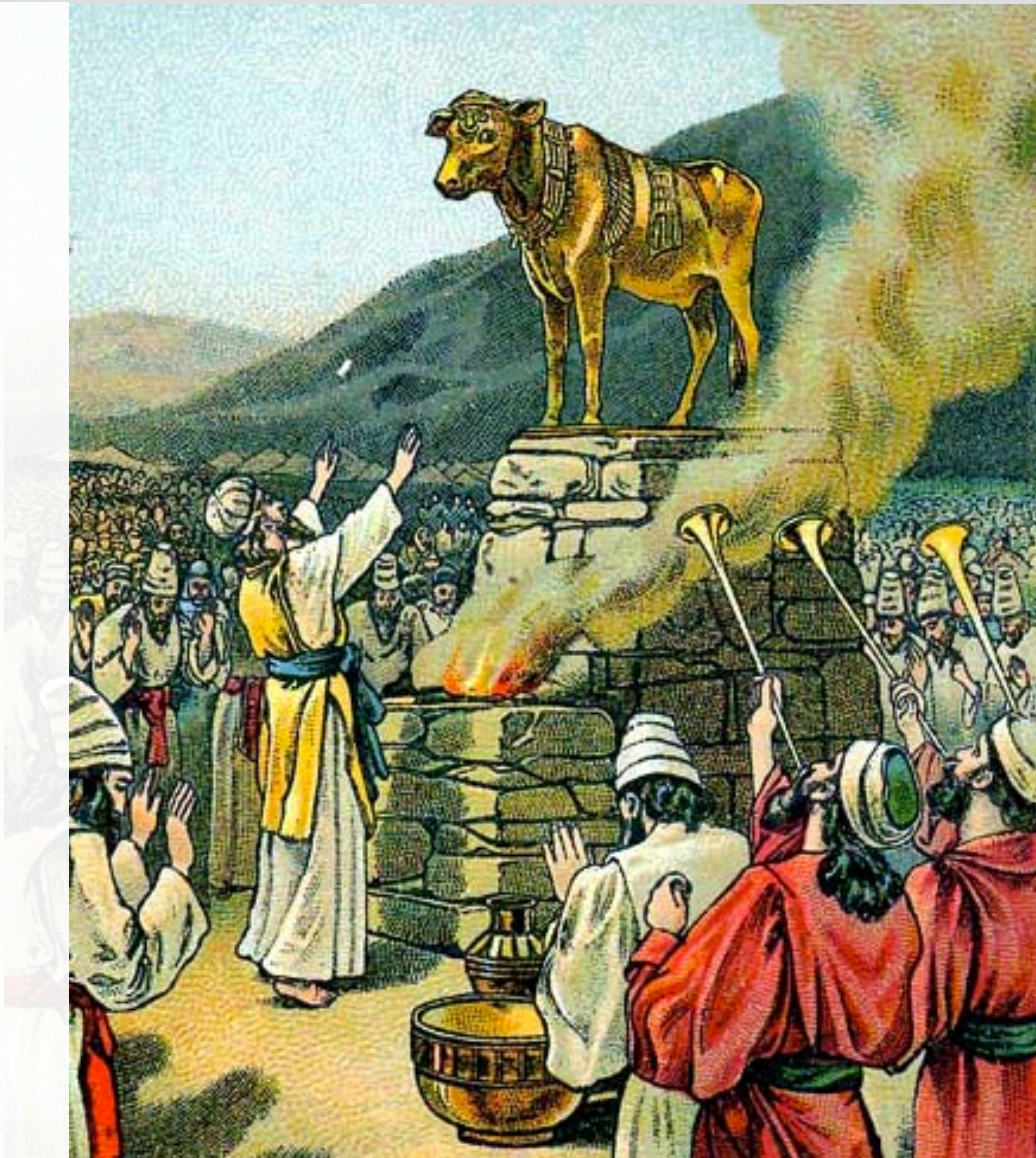


on empiricism

empirical research
exciting & powerful

empirical validation
as sole arbiter: a mistake

has not
upped field's reputation
resolved old disputes
made papers compelling



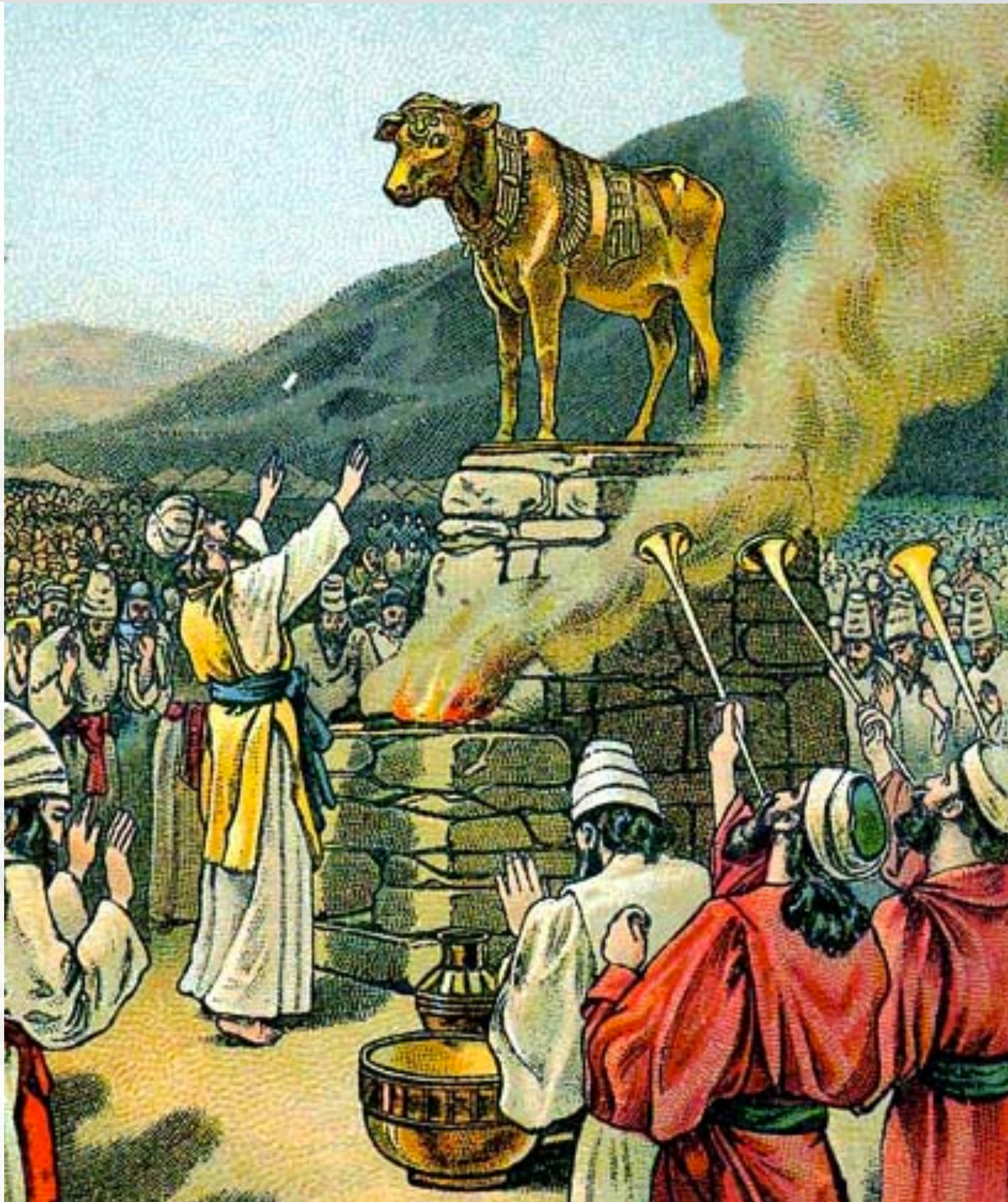
on empiricism

empirical research
exciting & powerful

empirical validation
as sole arbiter: a mistake

has not
upped field's reputation
resolved old disputes
made papers compelling

but has
inhibited novel work
devalued design research



serving industry?

serving industry?

industrial collaborations provide
source of new problems
deeper understanding of old problems
new approaches (XP, agile, etc)
opportunity to try research ideas

serving industry?

industrial collaborations provide
source of new problems
deeper understanding of old problems
new approaches (XP, agile, etc)
opportunity to try research ideas

but increasingly seems that
SE researchers see their role as serving industry
addressing immediate problems

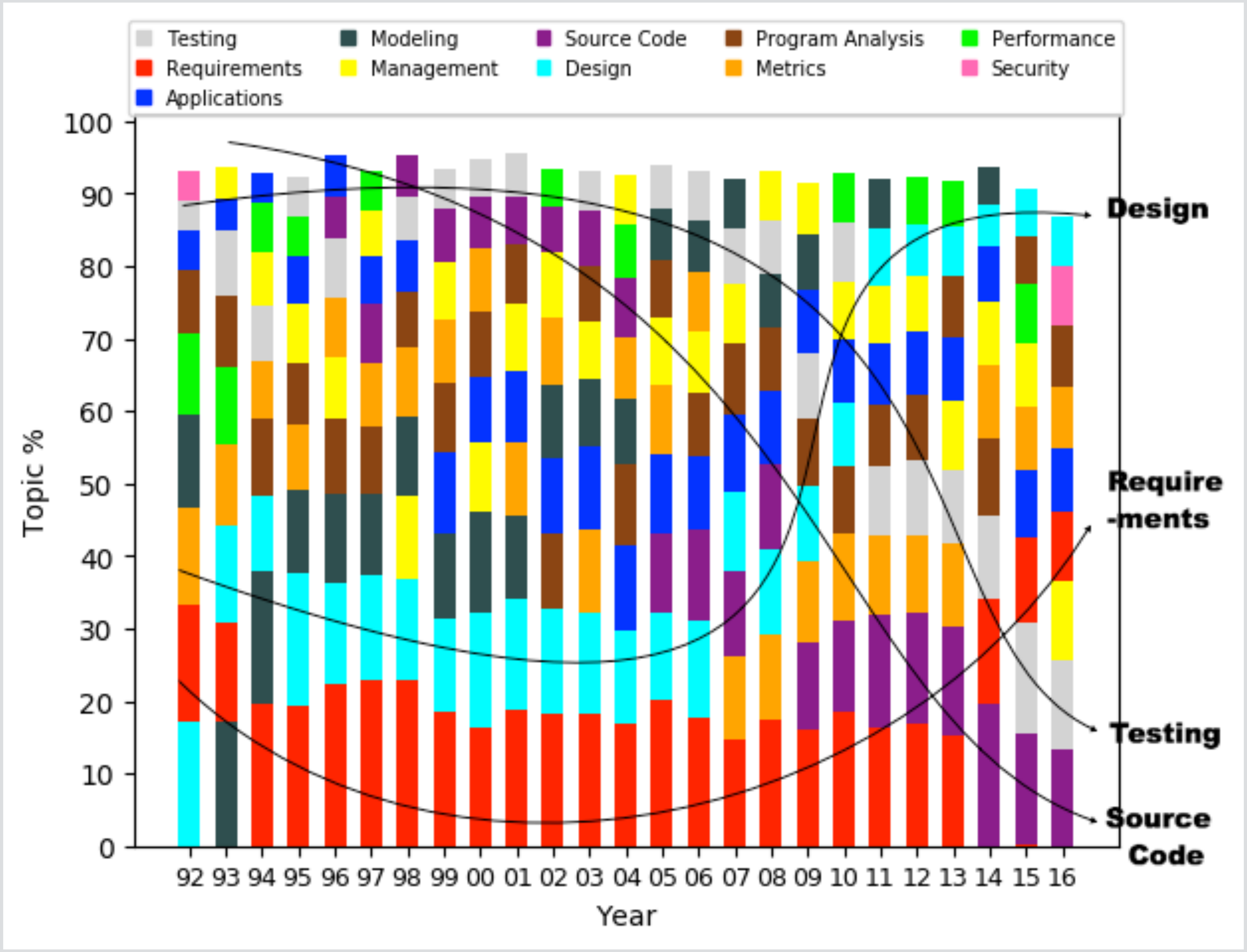
serving industry?

industrial collaborations provide
source of new problems
deeper understanding of old problems
new approaches (XP, agile, etc)
opportunity to try research ideas

but increasingly seems that
SE researchers see their role as serving industry
addressing immediate problems

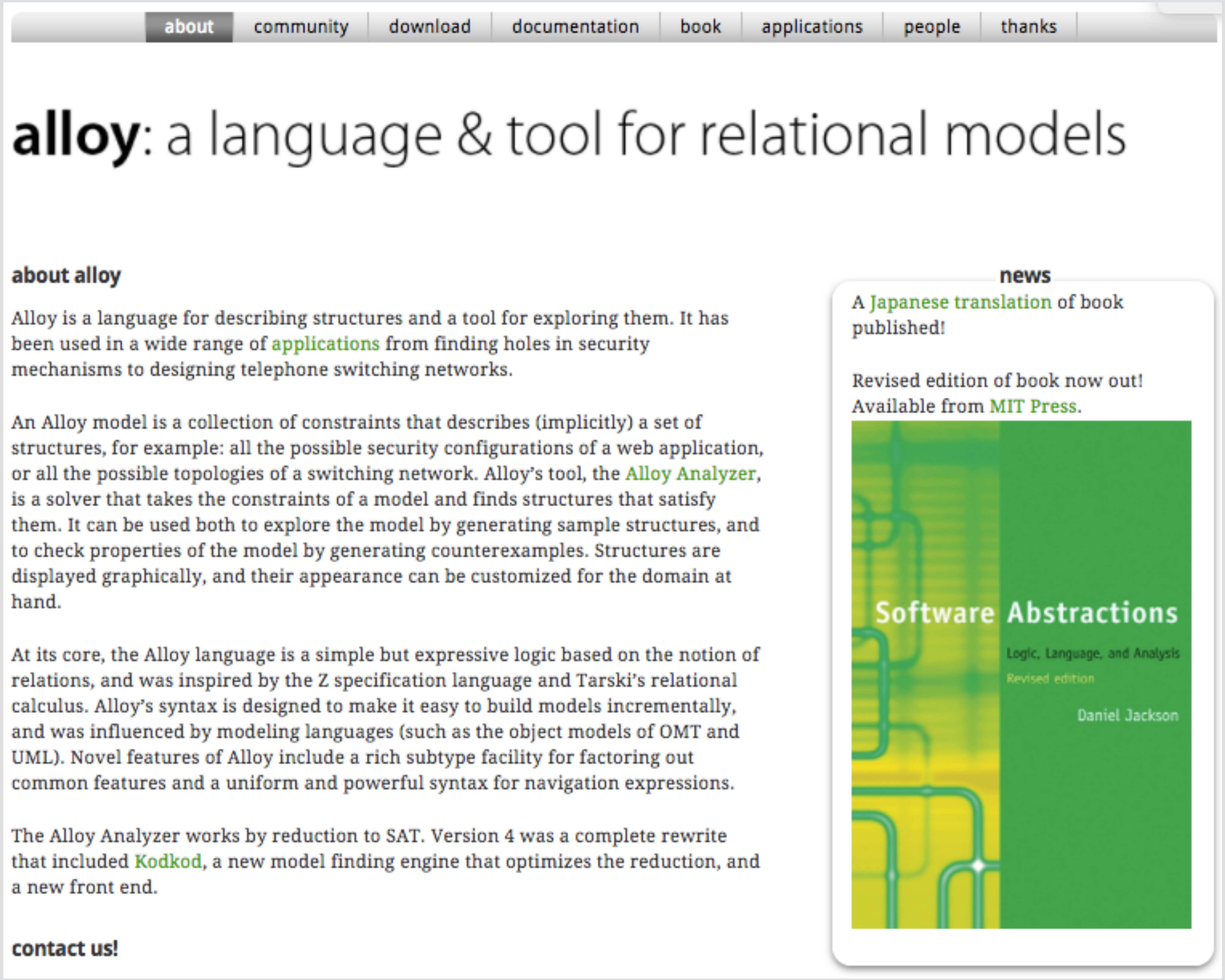
this leads to
overemphasis on code & test
lack of long-term thinking

a consequence



from Mathew, Agrawal & Menzies

more info at http://alloy.mit.edu



alloy: a language & tool for relational models

about alloy

Alloy is a language for describing structures and a tool for exploring them. It has been used in a wide range of applications from finding holes in security mechanisms to designing telephone switching networks.

An Alloy model is a collection of constraints that describes (implicitly) a set of structures, for example: all the possible security configurations of a web application, or all the possible topologies of a switching network. Alloy's tool, the Alloy Analyzer, is a solver that takes the constraints of a model and finds structures that satisfy them. It can be used both to explore the model by generating sample structures, and to check properties of the model by generating counterexamples. Structures are displayed graphically, and their appearance can be customized for the domain at hand.

At its core, the Alloy language is a simple but expressive logic based on the notion of relations, and was inspired by the Z specification language and Tarski's relational calculus. Alloy's syntax is designed to make it easy to build models incrementally, and was influenced by modeling languages (such as the object models of OMT and UML). Novel features of Alloy include a rich subtype facility for factoring out common features and a uniform and powerful syntax for navigation expressions.

The Alloy Analyzer works by reduction to SAT. Version 4 was a complete rewrite that included Kodkod, a new model finding engine that optimizes the reduction, and a new front end.

contact us!

news

A Japanese translation of book published!

Revised edition of book now out! Available from MIT Press.

Software Abstractions

Logic, Language, and Analysis

Revised edition

Daniel Jackson