The Alloyed Joys of Software Engineering Research

Daniel Jackson

Keynote · ICSE 2017 · Buenos Aires
Alloy
alloy’s cultural origins

Oxford, home of Z

Pittsburgh, home of SMV
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
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 Timeline

<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, Davis Putnam</td>
<td>intentional naming</td>
</tr>
<tr>
<td>Alloy 2 (2001)</td>
<td>+ non-binary relations signatures</td>
<td>Chaff, Berkmin symmetry, sharing</td>
<td>Unison filesync</td>
</tr>
<tr>
<td>Alloy 4 (2007)</td>
<td>+ meta, sequences arithmetic</td>
<td>bounds better sharing</td>
<td>flash filesystem</td>
</tr>
</tbody>
</table>
the alloy constraint analyzer

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

Re: Notice of Trademark Infringement

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 :-).

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,

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

\[
\text{sig Call \{members: set User\}}
\]

\[
\begin{align*}
\text{Call} & = \mathbb{P} (id: \text{CallId} \times \text{members: } \mathbb{P} \text{User}) \\
\text{User} & = \mathbb{P} (id: \text{UserId} \times \text{talking: } \mathbb{P} \text{User})
\end{align*}
\]

traditional interpretation

\[
\begin{align*}
\forall c, c' \in \mathbb{P} (id: \text{CallId} \times \text{members: } \mathbb{P} \text{User}) \mid \ldots
\end{align*}
\]

higher order quantification: ouch!

\[
\exists \text{members: Call } \leftrightarrow \text{User} \mid \forall c, c' \in \text{Call} \mid \ldots
\]

first order quantification: solve with SAT

\[
\begin{align*}
\text{all } c, c' : \text{Call \{no c.members \& c'.members\}}
\end{align*}
\]

Alloy interpretation
everything’s a relation

```plaintext
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

\[\text{sig User \{talking: set User\}}\]

\[\text{check \{no u: User | u in u.talking\}}\]

\[
\begin{array}{c|c|c}
U0 & U1 & U2 \\
\hline
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
U0 & U1 & U2 \\
\hline
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
U0 & U1 & U2 \\
\hline
0 & 0 & 0 \\
1 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
(U0) & (U0, U1), (U1, U0) \\
\hline
(U0) & (U0, (U1)) \\
\end{array}
\]
getting satisfaction

```
sig User {talking: set User}

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

```
add symmetry breaking predicates too
```

```

<table>
<thead>
<tr>
<th></th>
<th>U0</th>
<th>U1</th>
<th>U2</th>
</tr>
</thead>
<tbody>
<tr>
<td>U0</td>
<td>u0</td>
<td>U0</td>
<td>U0</td>
</tr>
<tr>
<td>U1</td>
<td>u1</td>
<td>U1</td>
<td>U1</td>
</tr>
<tr>
<td>U2</td>
<td>u2</td>
<td>U2</td>
<td>U2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>U0</th>
<th>U1</th>
<th>U2</th>
</tr>
</thead>
<tbody>
<tr>
<td>U0</td>
<td>t00</td>
<td>t01</td>
<td>t02</td>
</tr>
<tr>
<td>U1</td>
<td>t10</td>
<td>t11</td>
<td>t12</td>
</tr>
<tr>
<td>U2</td>
<td>t20</td>
<td>t21</td>
<td>t22</td>
</tr>
</tbody>
</table>

```

u0 ⇒ (u0 ∧ t00) ∨ (u1 ∧ t10) ∨ (u2 ∧ t20) ∧

u1 ⇒ (u1 ∧ t01) ∨ (u1 ∧ t11) ∨ (u2 ∧ t21) ∧

u2 ⇒ (u0 ∧ t02) ∨ (u1 ∧ t12) ∨ (u2 ∧ t22)

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 }
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 $\leq 25$ pairs
can check an operation on $2^{25}$ pre-states
but if initially empty, 25 steps to populate?
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

\[ \exists p: \text{Program} \mid \forall s: \text{State} \mid S(p,s) \]

this motivated Alloy* [Milicevic+]

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

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

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?

```plaintext
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 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

can be maddening
harder to learn than I knew
even harder to debug
unsat core not enough
20

projects
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+]
ar-chitectural 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]
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

in all cases, it's more than finding bugs
3 lessons
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."

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!
thoughts
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?
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?

**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
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!

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