# recent advances in Alloy

Daniel Jackson · MIT

Integrated Formal Methods · Oxford · July 4, 2007

CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

# outline

**three advances in Alloy**

- a new modelling idiom
- a new engine/API
- a new analysis feature

# what is Alloy?

# Alloy

**a software modelling language**

- small and uniform ASCII syntax for first-order logic with relations
- binding by parameters (with funs/preds) and free variables (with sigs)
- subtype and parametric polymorphism without casts
- dependent declarations, multiplicity constraints

**an analysis tool**

- exhaustive analysis and simulation with bounded 'scope'
- based on off-the-shelf SAT solvers
- customizable visualization
- available as API for use in other tools

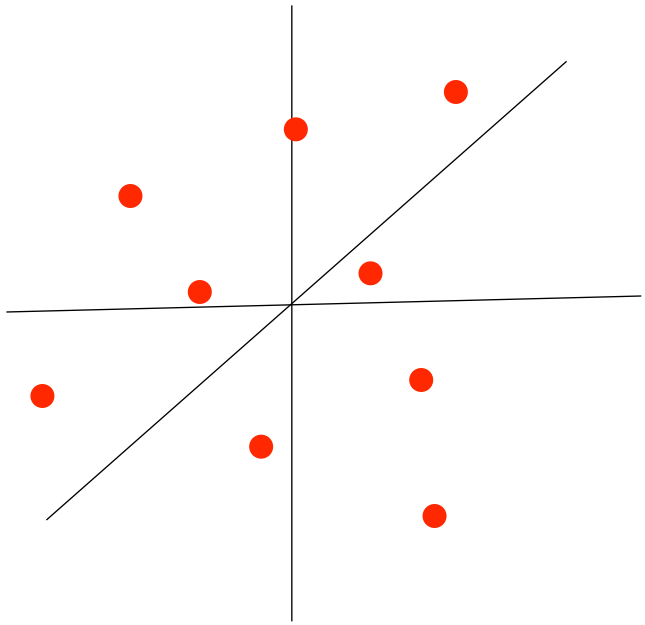
**influences**

- Z, VDM, Larch, Syntropy, etc; model checking

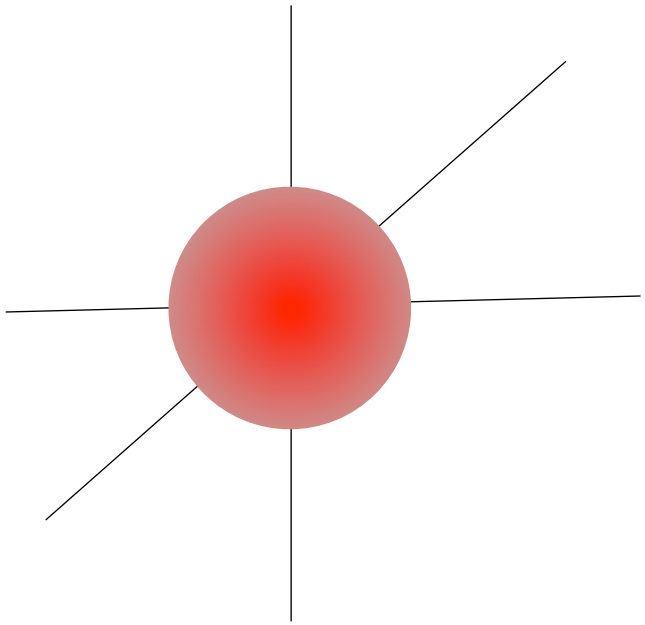
# scope-complete analysis

**observations about analyzing designs**

- most assertions are wrong
- most flaws have small counterexamples



testing:
a few cases of arbitrary size



scope-complete:
all cases within small scope

# applications of Alloy

**in teaching**

- taught in over 30 courses worldwide
- not just FMs and modelling; maths too (eg, Huth&Ryan, Trinity)

**in research and industry**

- First Alloy Workshop, September 2006 with FSE
- Airbus, AT&T, Galois, IBM, NASA, Navteq, Northrop Grumman, Telcordia

**applications to date**

- design analysis
- test case generation
- code verification
- automatic configuration

# elements of alloy

# alloy in 3 slides

**signatures**

- provide classification hierarchy for sets
- composite structure of objects
- local name space for relations
- incremental development

**relational logic**

- simple and uniform
- generalized join

**facts, predicates and assertions**

- simple packaging of constraints

# signatures & fields

```
sig A {}
```

-- introduces a set of atoms called A

```
sig B extends A {}
```

-- introduces a subset B of A

```
sig C extends A {}
```

-- introduces a subset C of A disjoint from B

```
sig A {f: B}
```

-- introduces a binary relation from A to B called f

```
sig A {f: B->C}
```

-- introduces a ternary relation from A to B to C called f

# relational operators

| | |
|---|---|
| p + q | $\{t \mid t \in p \vee t \in q\}$ |
| p - q | $\{t \mid t \in p \wedge t \notin q\}$ |
| p & q | $\{t \mid t \in p \wedge t \in q\}$ |
| p -> q | $\{(p_1, \ldots p_n, q_1, \ldots q_m) \mid (p_1, \ldots p_n) \in p \wedge (q_1, \ldots q_m) \in q$ |
| p . q | $\{(p_1, \ldots p_{n-1}, q_2, \ldots q_m) \mid (p_1, \ldots p_n) \in p \wedge (p_n, q_2, \ldots q_m) \in q\}$ |
| p in q | $\{(p_1, \ldots p_n) \in p\} \subseteq \{(q_1, \ldots q_n) \in q\}$ |
| p = q | $\{(p_1, \ldots p_n) \in p\} = \{(q_1, \ldots q_n) \in q\}$ |

eg, given **sig** A {f: B->C}, a: A, b: B, c: C
some expressions and their types:

a.f: B->C

f.c: A->B

b.(a.f): set C

# constraints & commands

**fact** {F}

-- establishes formula F, as an assumption

**pred** P () {Fp}

-- declares predicate P; invocation equivalent to inlining Fp

**assert** A () {Fa}

-- declares assertion A, claiming that formula Fa is valid

**run** P

-- instructs analyzer to find instance satisfying facts and Fp

**check** A

-- instructs analyzer to find instance satisfying facts and **not** Fa

# favorite example: hotel locks

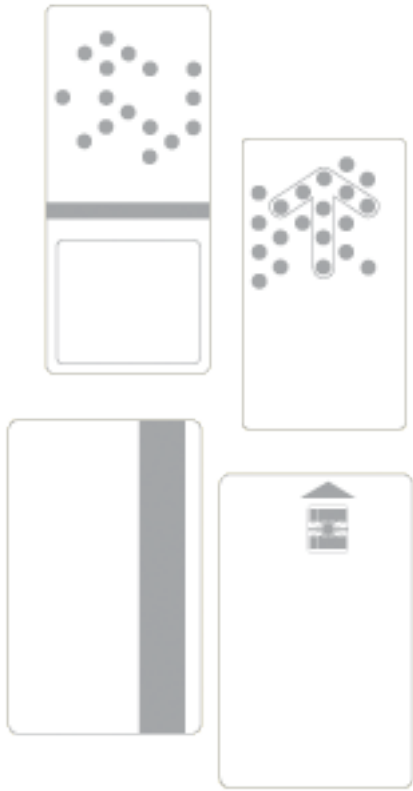
# hotel locking

**recodable locks (since 1980)**

- new guest gets a different key
- lock is ‘recoded’ to new key
- last guest can no longer enter

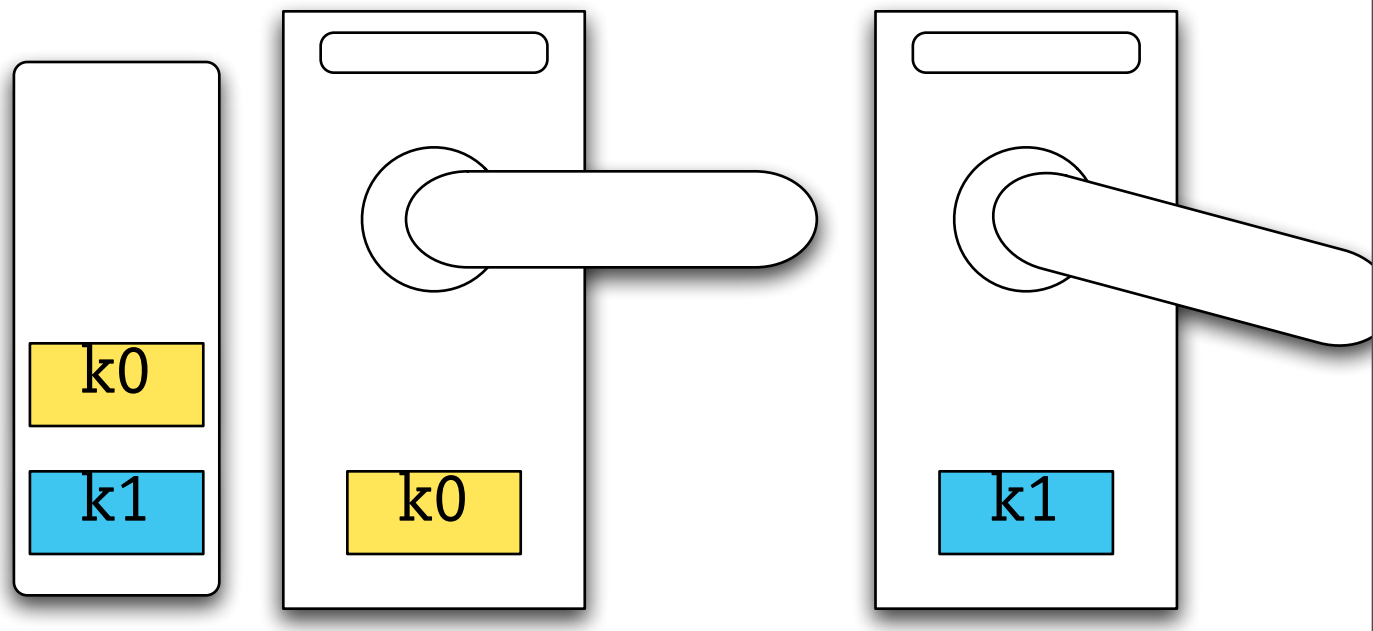
**how does it work?**

- locks are standalone, not wired

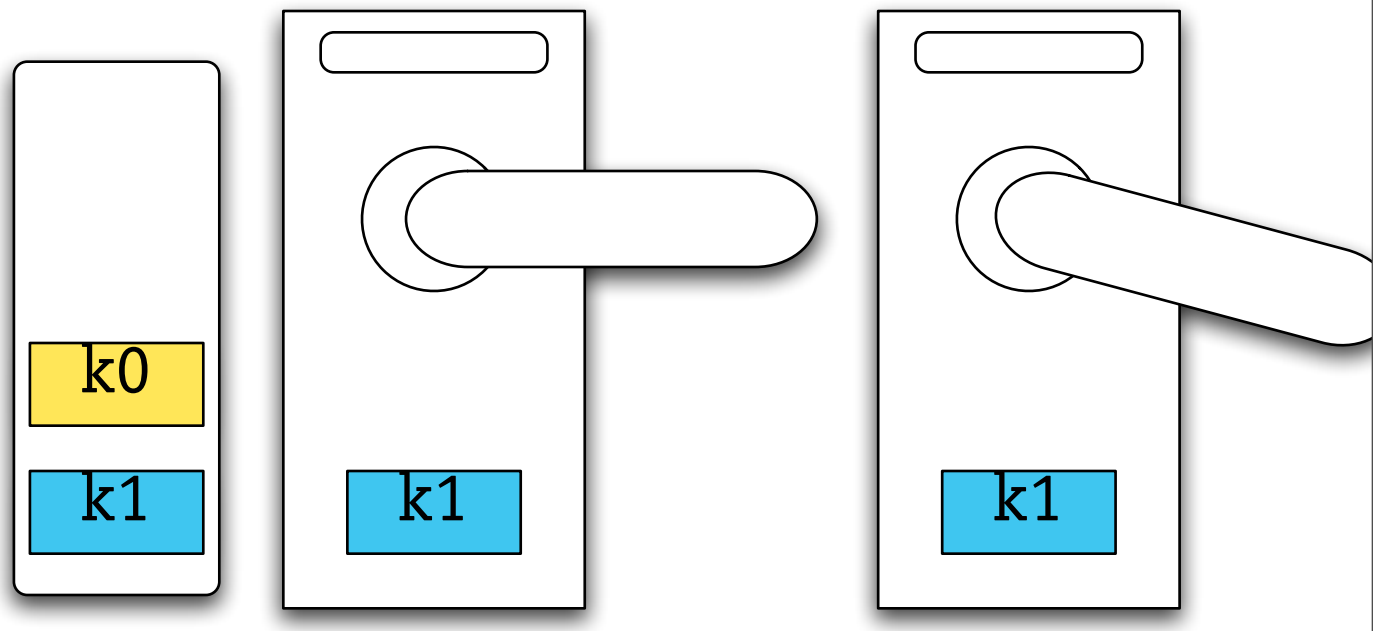


# a recodable locking scheme

card has two keys
if first matches lock,
recode with second



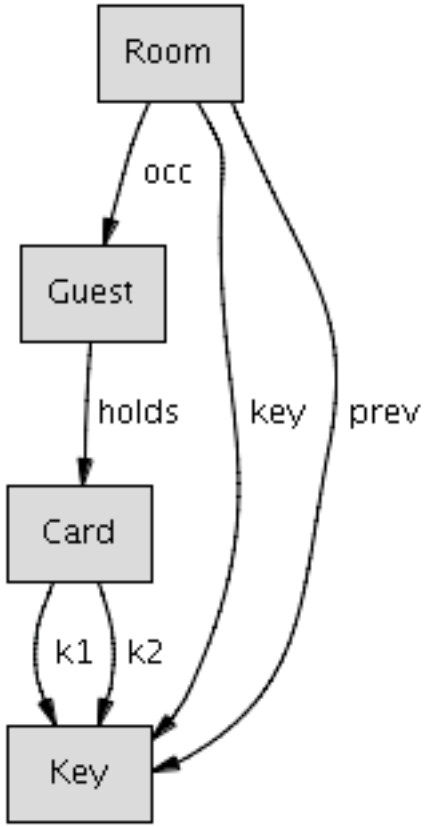

if second matches,
just open



# a new idiom

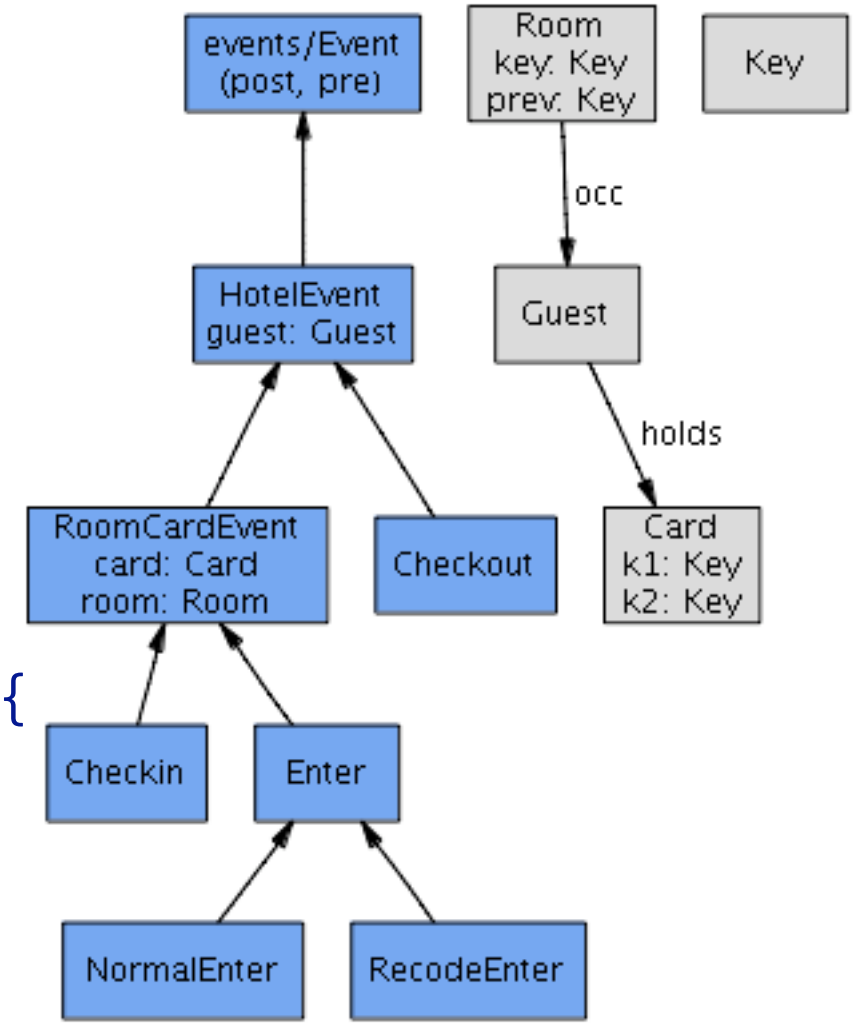
# local state

```
sig Key {}

sig Card {k1, k2: Key}
-- c.k1 is first key of card c
-- k1.k is set of cards with k as first key

sig Guest {
   holds: Card -> Time
   }
-- g.holds.t is set of cards g holds at time t

sig Room {
   key: Key one -> Time,
   prev:  Key lone -> Time,
   occ: Guest -> Time
   }
-- r.key.t is key of room r at time t
```



# events as objects

```
abstract sig Event {
   pre, post: Time }
   }

abstract sig HotelEvent extends Event {
   guest: Guest
   }

sig Checkout extends HotelEvent { }

abstract sig RoomCardEvent extends HotelEvent {
   room: Room,
   card: Card
   }

sig Checkin extends RoomCardEvent { }

abstract sig Enter extends RoomCardEvent { }
sig NormalEnter extends Enter { }
sig RecodeEnter extends Enter { }
```



# constraining events

```
abstract sig Enter extends RoomCardEvent { }
   {
   card in guest.holds.pre
   }

sig RecodeEnter extends Enter { }
   {
   card.k1 = room.key.pre
   key.post = key.pre ++ room -> card.k2
   }
```

# frame conditions

```
sig RecodeEnter extends Enter { }
   {
   card.k1 = room.key.pre
   key.post = key.pre ++ room -> card.k2

   prev.unchanged
   holds.unchanged
   occ.unchanged
   }

pred Event.unchanged (field: univ -> Time) {
   field.(this.pre) = field.(this.post)
   }

pred Event.unchanged (field: univ -> univ -> Time) {
   field.(this.pre) = field.(this.post)
   }
```

# frame conditions, Reiter-style

```
sig Room {
   key: Key one -> Time,
   prev:  Key lone -> Time,
   occ: Guest -> Time
   }
   {
   Checkin.modifies [prev]
   (Checkin + Checkout).modifies [occ]
   RecodeEnter.modifies [key]
   }

pred modifies (es: set Event, field: univ -> Time) {
   all e: Event - es | field.(e.pre) = field.(e.post)
   }
```

# generating traces

```
open util/ordering[Time] as time

sig Time {}

abstract sig Event {
   pre, post: Time
   }

fact Traces {
   all t: Time - last | one e: Event | e.pre = t and e.post = t.next
   }
```

# is it safe?

```
assert NoBadEntry {
   all e: Enter | let occs = occ.(e.pre) [e.room] |
         some occs => e.guest in occs
   }

check NoBadEntry for 5
```

# summary

**to use idiom**

- open library module providing Event, unchanged, traces
- add Time column to time-varying fields
- declare events with pre/post conditions

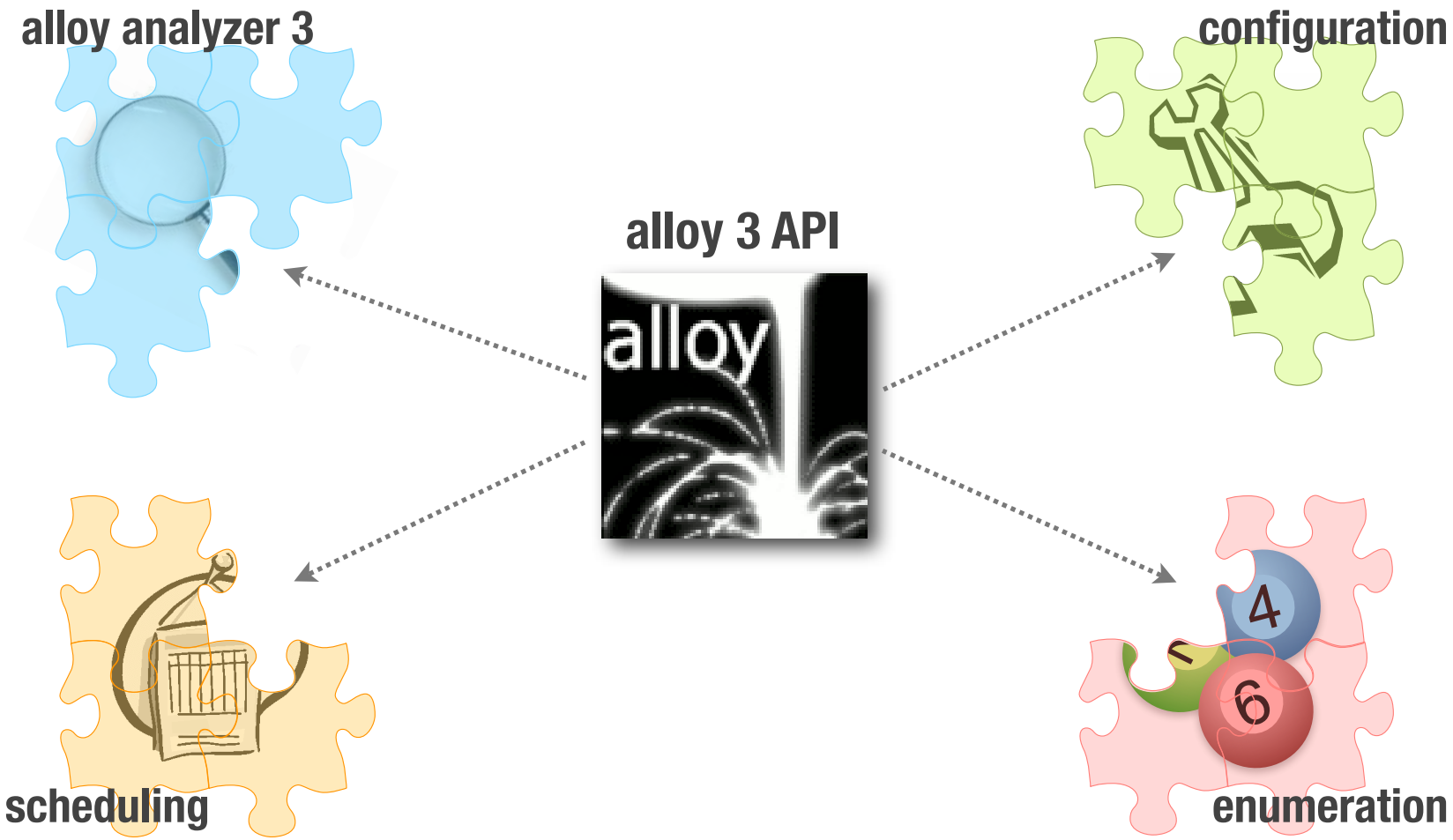
**advantages**

- dynamic aspect doesn't interfere with subtyping of domain objects
- classification of events factors out common elements
- can express LTL properties and more

# a new engine

[ Emina Torlak ]

# why a new engine?



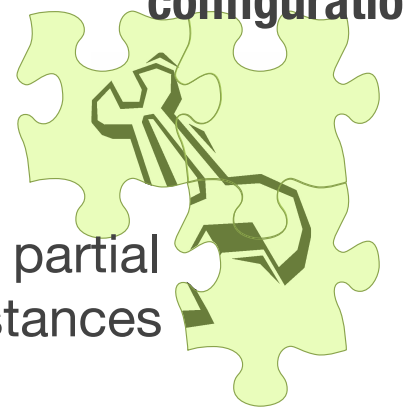
# why a new engine?

**alloy analyzer 3**

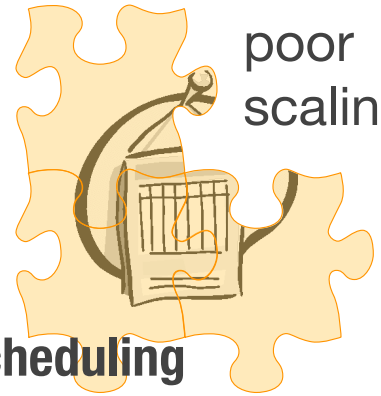
no sharing

**alloy 3 API**


**configuration**



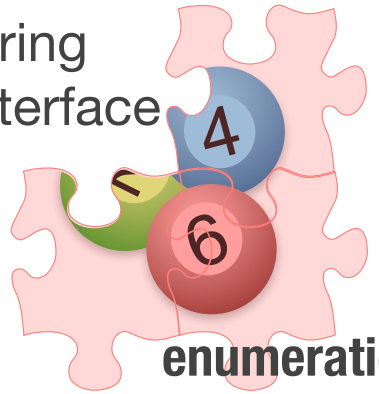
no partial instances



poor scaling

**scheduling**

string interface



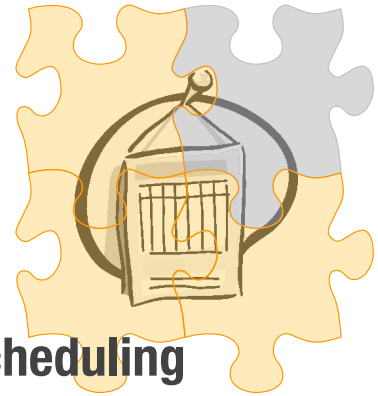

**enumeration**

# kodkod

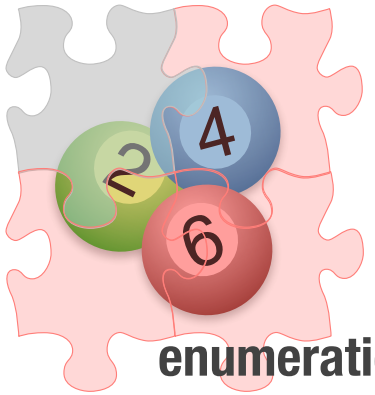
alloy analyzer 4.0

kodkod

configuration

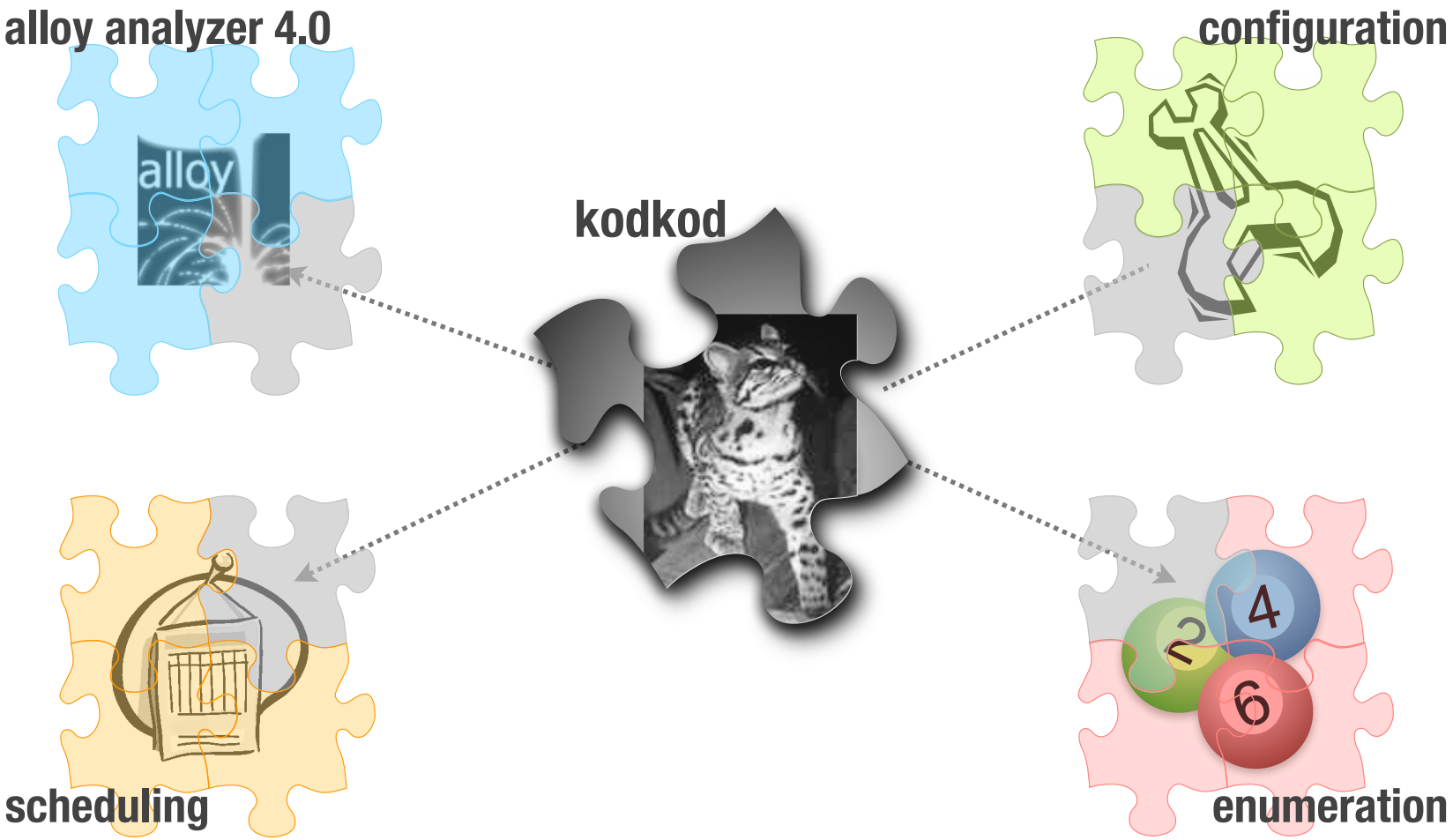


scheduling

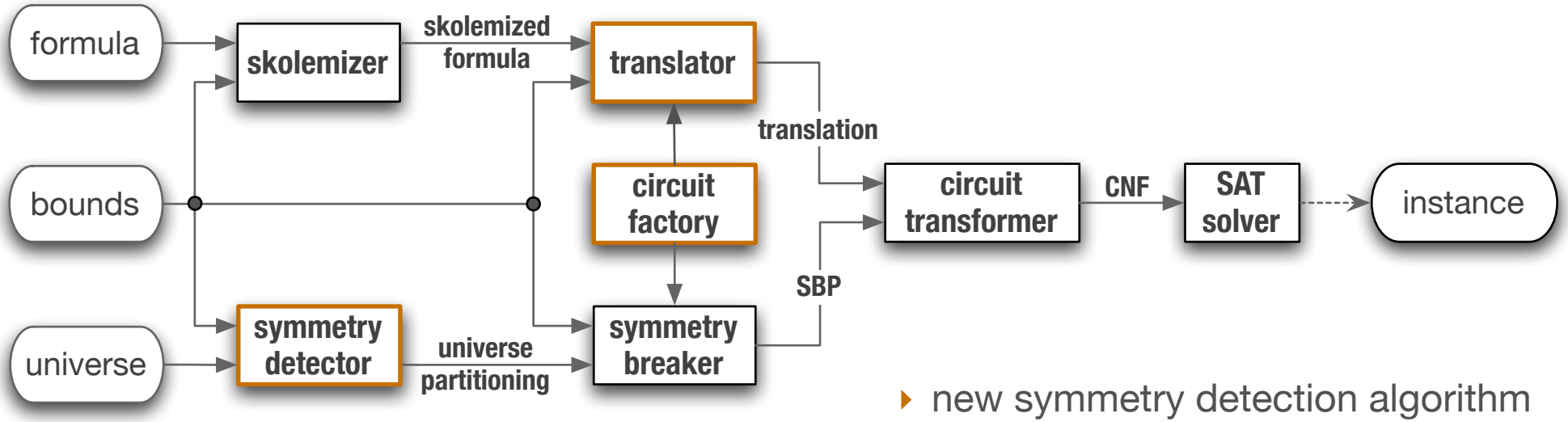



enumeration

# kodkod


alloy analyzer 4.0
alloy
kodkod
configuration
scheduling
enumeration

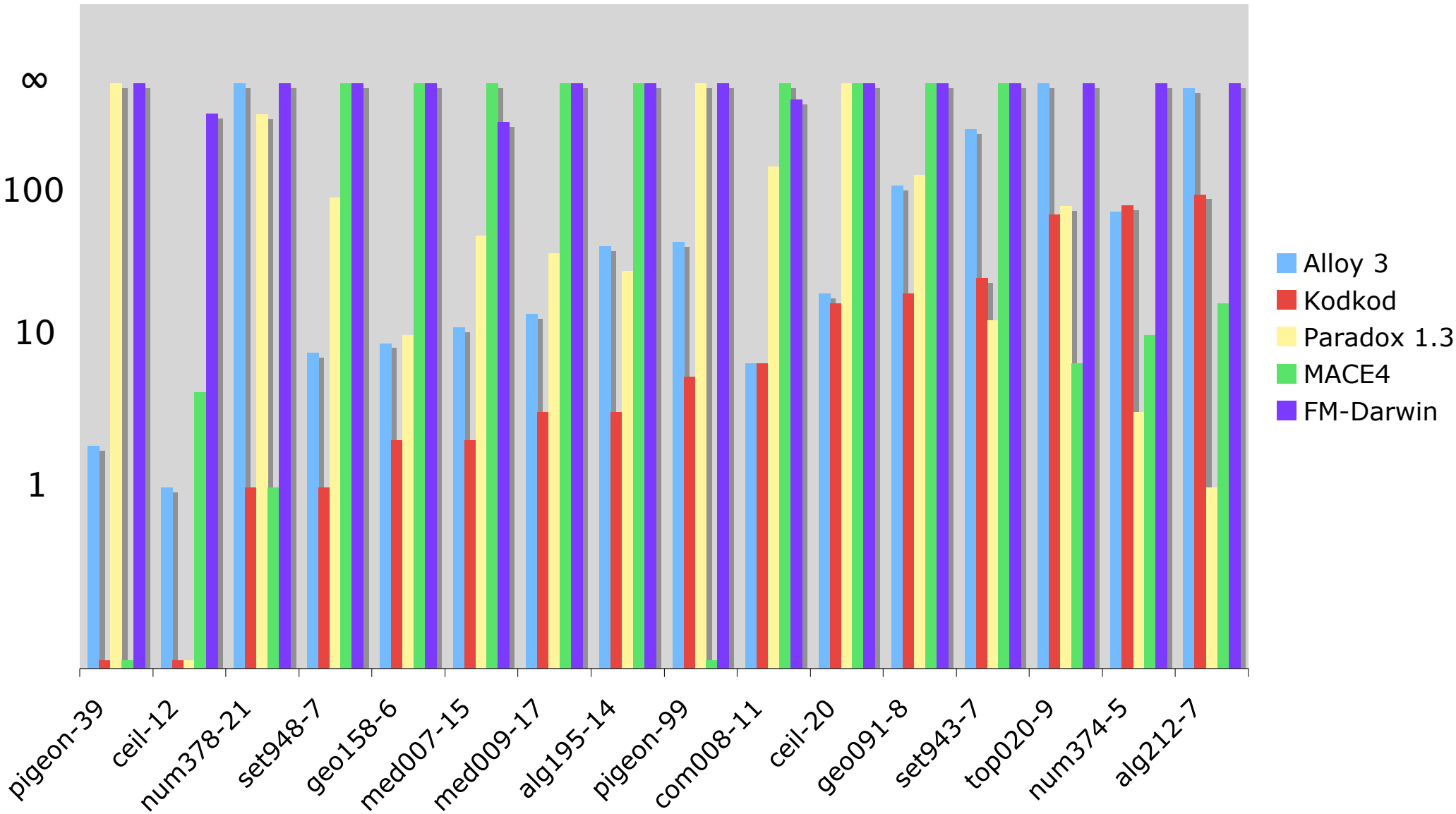
# kodkod structure




- new symmetry detection algorithm that works for arbitrary bounds
- new translation to propositional logic based on sparse matrices
- new sharing detection based on *compact boolean circuits*

# performance

analysis time, seconds



# a new analysis

[ Emina Torlak & Felix Chang ]

# multiple choice

**1. suppose analyzer finds a counterexample to an assertion**

A. assertion is **wrong**

B. analyzer is **broken**

**2. suppose analyzer does *not* find a counterexample**

A. assertion is **valid** and **design is correct**

B. assertion is valid but **design is too strong**

C. assertion is valid but **assertion is too weak**

D. assertion is **invalid**, has counterexample in larger scope

**research question**

- can we detect 2B, 2C, 2D?

# exposing the proof

**when no counterexample is found**

- SAT solver creates a **proof of unsatisfiability**
- just a resolution graph over the clauses of the CNF

**sometimes, not all clauses are used in the proof**

- unused clauses are irrelevant
- Alloy constraints that translate to unused clauses are also!

# scope to small

```
Executing "Check NoBadEntry for 2"
   Solver=minisatprover(jni) Bitwidth=4 MaxSeq=2 Symmetry=20
   1647 vars. 86 primary vars. 3117 clauses. 32ms.
   No counterexample found.  Assertion may be valid. 88ms.
```

**with only two time instants**

- no Enter event can happen (since Checkin must precede it)

**highlighting shows**

- Event postcondition irrelevant

```
sig Checkin extends RoomCardEvent { }
    {
    no room.occ.pre
    card.k1 = room.prev.pre
    holds.post = holds.pre + guest -> card
    prev.post = prev.pre ++ room -> card.k2
    occ.post = occ.pre + room -> guest

    key.unchanged
    }

abstract sig Enter extends RoomCardEvent { }
    {
    card in guest.holds.pre
    }

sig NormalEnter extends Enter { }
    {
    card.k2 = room.key.pre

    prev.unchanged
    holds.unchanged
    occ.unchanged
    key.unchanged
    }

sig RecodeEnter extends Enter { }
    {
    card.k1 = room.key.pre
    key.post = key.pre ++ room -> card.k2

    prev.unchanged
    holds.unchanged
    occ.unchanged
    }

sig Checkout extends HotelEvent { }
    {
    some occ.pre.guest
```

# model too strong

```
Executing "Check NoBadEntry for 5"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=5 Symmetry=20
  22023 vars. 755 primary vars. 55638 clauses. 257ms.
  No counterexample found.  Assertion may be valid. 1136ms.
```

**to prevent double issue, wrote**

```
all e1, e2: Checkin |
    e1.card.k2 != e2.card.k2
```

**highlighting shows**

- Checkin is irrelevant
- can never happen!

**meant**

```
all disj e1, e2: Checkin |
    e1.card.k2 != e2.card.k2
```

```
sig Checkin extends RoomCardEvent { }
    {
    no room.occ.pre
    card.k1 = room.prev.pre
    holds.post = holds.pre + guest -> card
    prev.post = prev.pre ++ room -> card.k2
    occ.post = occ.pre + room -> guest

    key.unchanged
    }

abstract sig Enter extends RoomCardEvent { }
    {
    card in guest.holds.pre
    }

sig Checkout extends HotelEvent { }
    {
    some occ.pre.guest
    occ.post = occ.pre - Room -> guest

    prev.unchanged
    holds.unchanged
    key.unchanged
    }

run {some Checkout}

run {some Enter} for 2


fact NoDoubleIssue {
    all  e1, e2: Checkin | e1.card.k2 != e2.card.k2  -- dor
    all e: Checkin | e.card.k2 !in Room.key.first -- don't iss
    }
```

# assertion too weak

```
Executing "Check MustHoldKey for 3"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=3 Symmetry=20
  4873 vars. 213 primary vars. 10471 clauses. 306ms.
  No counterexample found. Assertion may be valid. 143ms.
```

```
assert MustHoldKey {
    all e: Enter | e.card in e.guest.holds.(e.pre)
    }
check MustHoldKey for 3
```

**assertion is just a restatement**

- of precondition of Enter

**highlighting shows**

- other Event constraints irrelevant

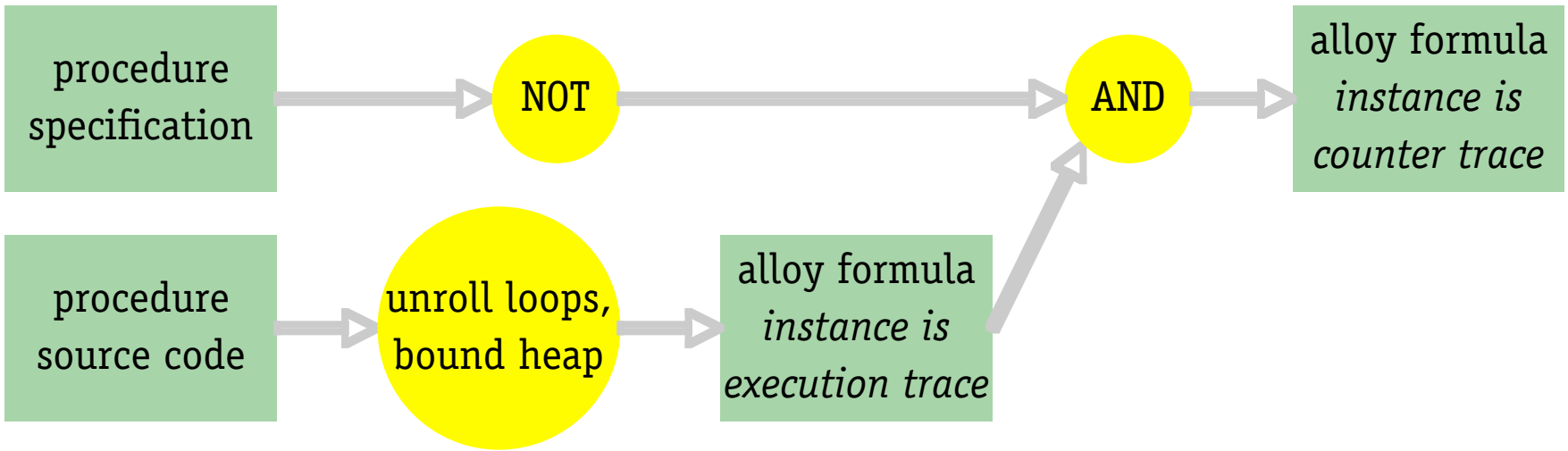
```
abstract sig Enter extends RoomCardEvent { }
    {
    card in guest.holds.pre
    }

sig NormalEnter extends Enter { }
    {
    card.k2 = room.key.pre

    prev.unchanged
    holds.unchanged
    occ.unchanged
    key.unchanged
    }

sig RecodeEnter extends Enter { }
    {
    card.k1 = room.key.pre
    key.post = key.pre ++ room -> card.k2

    prev.unchanged
    holds.unchanged
    occ.unchanged
    }

sig Checkout extends HotelEvent { }
    {
    some occ.pre.guest
    occ.post = occ.pre - Room -> guest

    prev.unchanged
    holds.unchanged
    key.unchanged
    }
```

# other ongoing projects

# code verification

**a new tradeoff**

- fully automatic, deep analysis of rich specifications
- counterexample traces generated



# code verification progress

**two aspects of work**

- demand-driven specification extraction for called code (Mana Taghdiri)
- efficient translation of code into Alloy (Greg Dennis)

**current case studies**

- libraries (OpenJGraph, Sun Java Collections)
- Quartz job scheduler
- KOA electronic voting system

**experience**

- discovered subtle bugs missed by testing
- typical performance:
  - few thousand kloc
  - scope of 5 (types, loop unwindings)

# proton therapy project

**collaboration with BPTC**

- Burr Proton Therapy Center at MGH
- methods for safer software
- dependability & flexibility

**projects to date**

- beam scheduler design analysis
- emergency stop analysis (code dependences)
- end-to-end dependability case

# infrastructure development

**recent news**

- awarded $800k by NSF (June 2007)

**plans**

- tool improvements
- educational materials
- user community website
- case study repository

# for more information

**about Alloy**

- http://alloy.mit.edu

**about Kodkod**

- http://web.mit.edu/~emina/www/kodkod.html

**about the Software Design Group**

- http://sdg.csail.mit.edu

**contact me!**

- dnj@mit.edu