micromodels of software
declarative modelling and analysis with Alloy

lecture 3: analysis

Daniel Jackson
MIT Lab for Computer Science
Marktoberdorf, August 2002

Alloy’s analysis

Alloy's analysis

only one kind of analysis

- given formula, find instance

Alloy’s analysis

only one kind of analysis

- given formula, find instance

instance

- assignment that makes formula true
- to free variables
- witnesses to existential by skolemizing

Alloy’s analysis

only one kind of analysis
- › given formula, find instance

instance
- › assignment that makes formula true
- › to free variables
- › witnesses to existential by skolemizing

kinds of analysis
- › simulation: instance of formula, or example
- › check: instance of negation, or counterexample

simulation

simulation

formula

```
Person: set PERSON
Date: set DATE
bb, bb′ : Person ->? Date
p: Person
d: Date
bb′ = bb ++ p->d
```

simulation

formula

```
Person: set PERSON
Date: set DATE
bb, bb′ : Person ->? Date
p: Person
d: Date
bb′ = bb ++ p->d
```

example

```
PERSON = {(P0),(P1),(P2)}
Person = {(P0),(P1)}
DATE = {(D0),(D1)}
Date = {(D0),(D1)}
p = {(P1)}
d = {(D1)}
bb = {(P0,D0),(P1,D0)}
bb′ = {(P0,D0),(P1,D1)}
```

checking

checking

formula

```
t: set T
r: t -> t
a, b: set t
not (a-b).r = a.r - b.r
```

checking

formula

```
t: set T
r: t -> t
a, b: set t
not (a-b).r = a.r - b.r
```

counterexample

```
T = {(T0),(T1)}
t = {(T0),(T1)}
r = {(T0,T0),(T1,T0)}
a = {(T0)}
b = {(T1)}
```

scope

scope

scope

- gives dimensions of space
- number of atoms in each basic type

    instance I in scope S iff for all types T, #I(T) = S(T)

scope

scope

- gives dimensions of space
- number of atoms in each basic type

    instance I in scope S iff for all types T, #I(T) = S(T)

explosion!

- suppose scope (T) = s for all T, m relations of arity k

    each k-relation has s^k possible edges, so 2^(s^k) values

    m relations, so #space = (2^(s^k))^m = m(s^k) bits
- typical example

    s = 5, m = 20, k = 2, #space = 500 bits

scope for example

formula

```
Person: set PERSON
Date: set DATE
bb, bb′ : Person ->? Date
p: Person
d: Date
bb′ = bb ++ p->d
```

example

```
PERSON = {(P0),(P1),(P2)}
Person = {(P0),(P1)}
DATE = {(D0),(D1)}
Date = {(D0),(D1)}
p = {(P1)}
d = {(D1)}
bb = {(P0,D0),(P1,D0)}
bb′ = {(P0,D0),(P1,D1)}
```

scope for example

scope is
3 PERSON, 2 DATE

formula

Person: set PERSON

Date: set DATE

bb, bb′ : Person ->? Date

p: Person

d: Date

bb′ = bb ++ p->d

example

PERSON = {(P0),(P1),(P2)}

Person = {(P0),(P1)}

DATE = {(D0),(D1)}

Date = {(D0),(D1)}

p = {(P1)}

d = {(D1)}

bb = {(P0,D0),(P1,D0)}

bb′ = {(P0,D0),(P1,D1)}

scope for counterexample

formula

```
t: set T
r: t -> t
a, b: set t
not (a-b).r = a.r - b.r
```

counterexample

```
T = {(T0),(T1)}
t = {(T0),(T1)}
r = {(T0,T0),(T1,T0)}
a = {(T0)}
b = {(T1)}
```

scope for counterexample

scope is 2 for T

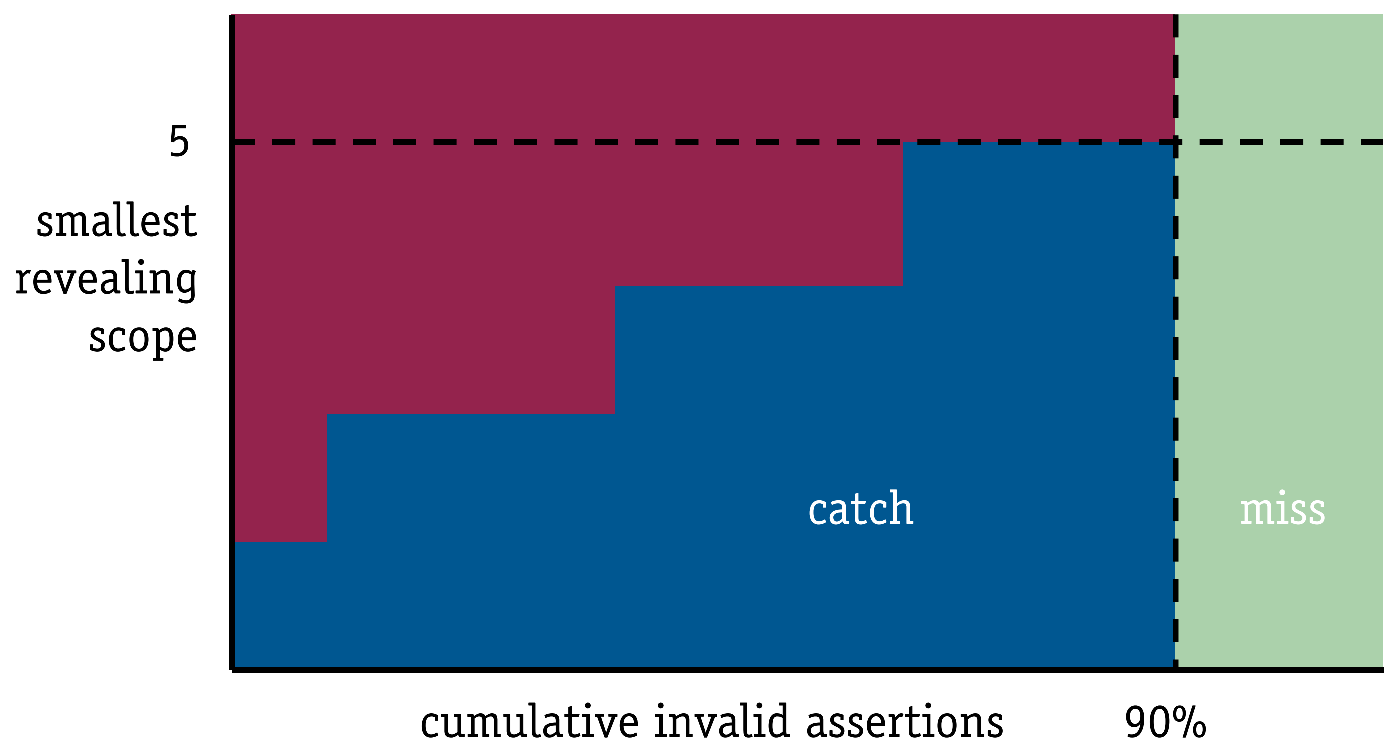
formula

```
t: set T
r: t -> t
a, b: set t
not (a-b).r = a.r - b.r
```

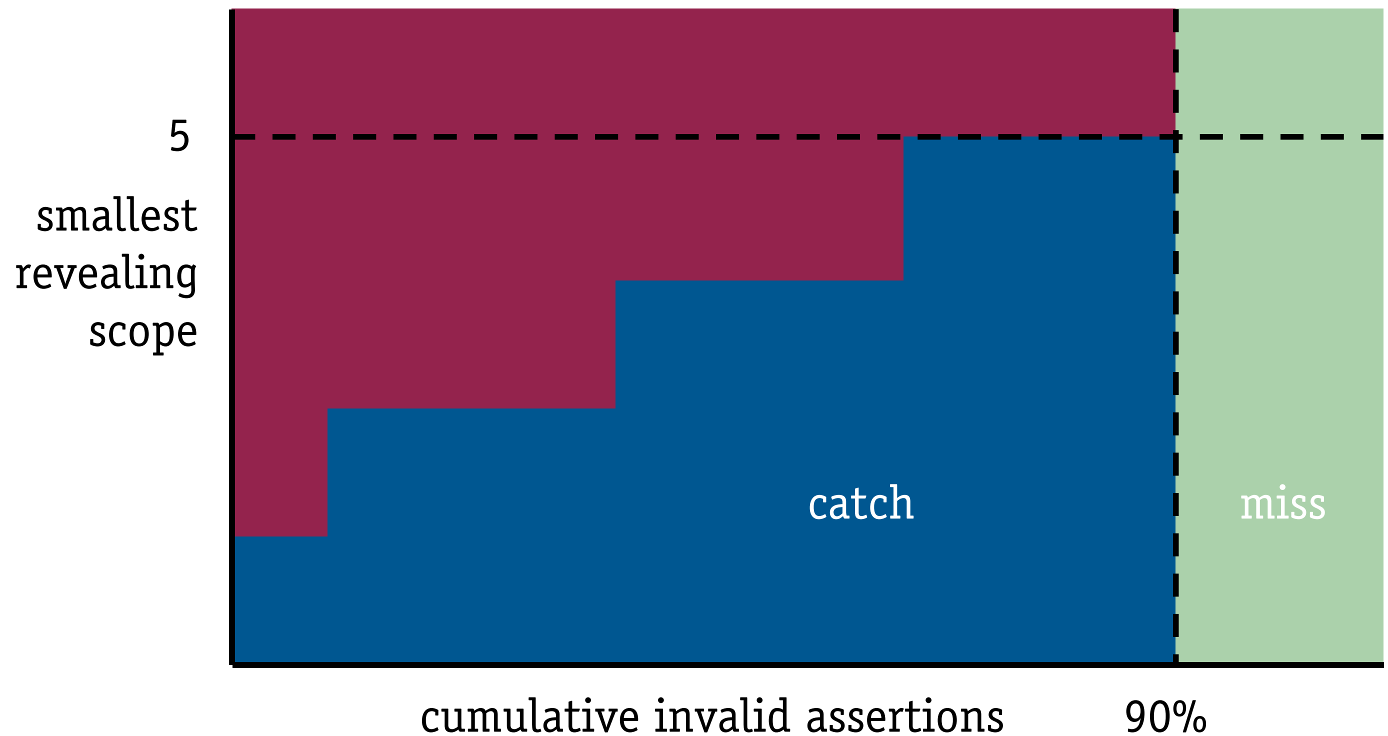
counterexample

```
T = {(T0),(T1)}
t = {(T0),(T1)}
r = {(T0,T0),(T1,T0)}
a = {(T0)}
b = {(T1)}
```

small scope hypothesis



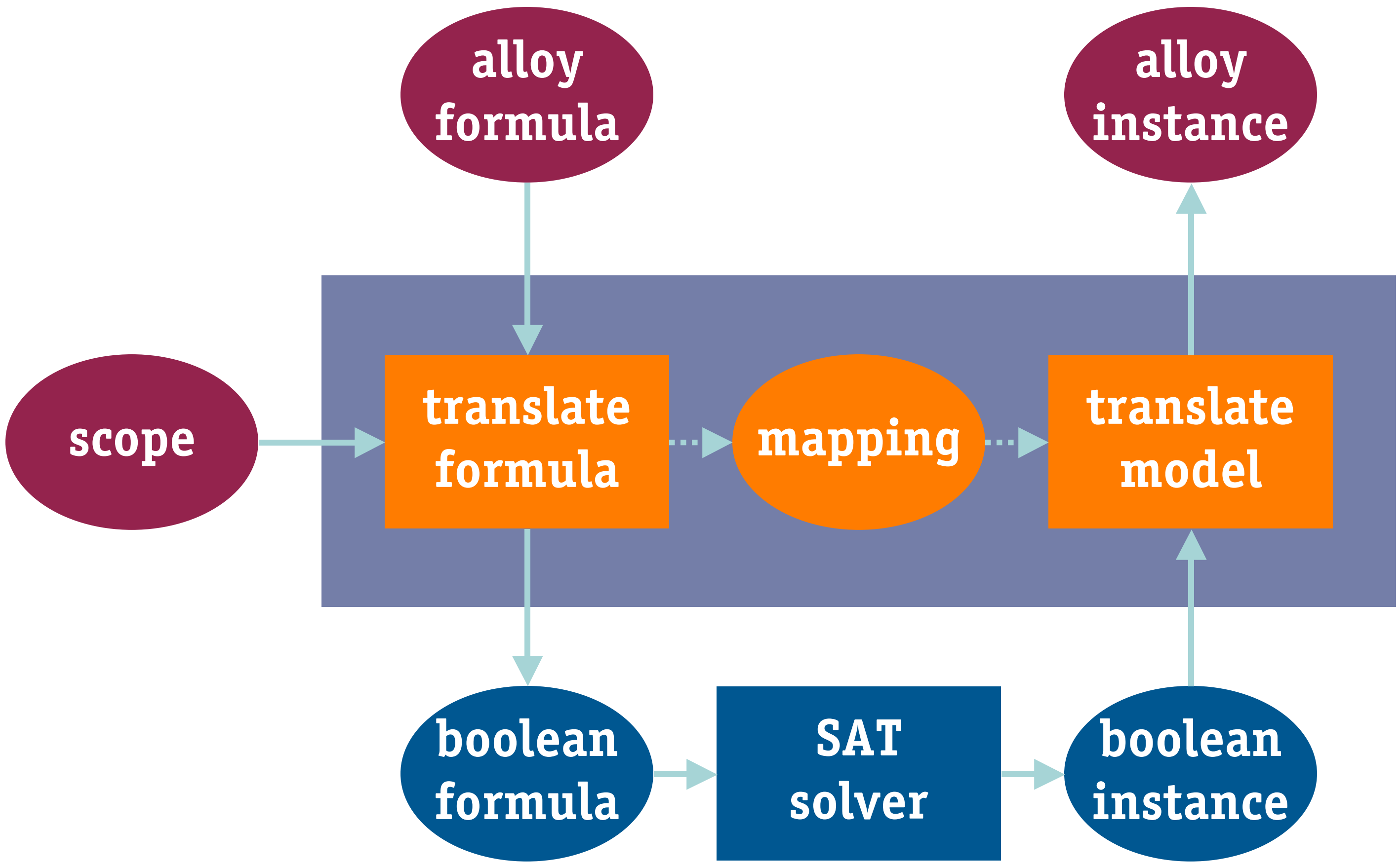
small scope hypothesis



search within finite scope

- sound: examples are correct
- incomplete: may miss an example
- but in practice, small scopes are enough

analyzer architecture



which instance?

which instance?

no guarantees

- if instance in scope, Alloy will find it
- but may not be smallest, nicest, etc
- not even deterministic

which instance?

no guarantees

› if instance in scope, Alloy will find it
› but may not be smallest, nicest, etc
› not even deterministic

in practice

› instances usually small
› can ask for another
› can make deterministic

translation to SAT

translation to SAT

idea

- relation’s value can be represented as adjacency matrix
- so space of values represented as matrix of boolean vars
- translate expr to matrix of boolean formulas
- translate formula to boolean formula

example

example

```
t: set T ; r: t -> t; a, b: set t
not (a-b).r = a.r - b.r
t, a, b: (T), r: (T,T)
```

example

t: set T ; r: t -> t; a, b: set t
not (a-b).r = a.r - b.r
t, a, b: (T), r: (T,T)

scope of 2, T = {(t0),(t1)}

example

t: set T ; r: t -> t; a, b: set t
not (a-b).r = a.r - b.r
t, a, b: (T), r: (T,T)

scope of 2, T = {(t0),(t1)}

t(i) true means tuple (T_i) in t
r(i,j) true means tuple (T_i,T_j) in r
a(i) true means tuple (T_i) in a
b(i) true means tuple (T_i) in b

example

t: set T ; r: t -> t; a, b: set t

not (a-b).r = a.r - b.r

t, a, b: (T), r: (T,T)

scope of 2, T = {(t0),(t1)}

t(i) true means tuple (T_i) in t

r(i,j) true means tuple (T_i,T_j) in r

a(i) true means tuple (T_i) in a

b(i) true means tuple (T_i) in b

(a-b) (i) □ a(i) AND NOT b(i)

(a-b).r (i) □ OR_j r(i) AND (a(i) AND NOT b(i)) call this F_i

(a.r-b.r) (i) □ (OR_j r(i) AND a(i)) AND NOT (OR_j r(i) AND b(i)) G_i

not (a-b).r = a.r - b.r □ NOT AND_i (F_i IFF G_i)

example

t: set T ; r: t -> t; a, b: set t
not (a-b).r = a.r - b.r
t, a, b: (T), r: (T,T)

scope of 2, T = {(t0),(t1)}

t(i) true means tuple (T_i) in t
r(i,j) true means tuple (T_i,T_j) in r
a(i) true means tuple (T_i) in a
b(i) true means tuple (T_i) in b

(a-b) (i) □ a(i) AND NOT b(i)
(a-b).r (i) □ OR_j r(i) AND (a(i) AND NOT b(i)) call this F_i
(a.r-b.r) (i) □ (OR_j r(i) AND a(i)) AND NOT (OR_j r(i) AND b(i)) G_i
not (a-b).r = a.r - b.r □ NOT AND_i (F_i IFF G_i)

solution: t_0 t_1 $r_{0,0}$ $r_{1,0}$ a_0 b_1
t = {(T0),(T1)}, r = {(T0,T0),(T1,T0)}, a = {(T0)}, b = {(T1)}

translation issues

translation issues

quantifiers

- ground them out
- skolemize when possible

translation issues

quantifiers

- ground them out
- skolemize when possible

conversion to CNF

- use standard techniques

translation issues

quantifiers
- ground them out
- skolemize when possible

conversion to CNF
- use standard techniques

exploiting repeated subformulas
- detect sharing and make formula a DAG, not a tree
- avoid repeated translations

translation issues

quantifiers
- ground them out
- skolemize when possible

conversion to CNF
- use standard techniques

exploiting repeated subformulas
- detect sharing and make formula a DAG, not a tree
- avoid repeated translations

symmetry
- formula is invariant on permutations of atoms in a type
- add symmetry-breaking predicates

solver experience

solver experience

which solvers?

› wrapped by backend; user selects

› Chaff (Malik) and Berkmin (Goldberg) work best

solver experience

which solvers?

- wrapped by backend; user selects
- Chaff (Malik) and Berkmin (Goldberg) work best

performance

- no systematic studies yet
- for <1kbit of declared state, terminates in seconds
- grounding out often a bottleneck

software engineering experience

software engineering experience

where effort goes

- CNF construction most tricky & error prone
- now in Java rather than C++, should be easier
- most code in front-end (type inference, static semantics)

software engineering experience

where effort goes

- CNF construction most tricky & error prone
- now in Java rather than C++, should be easier
- most code in front-end (type inference, static semantics)

display of results

- crucial aspect of tool
- visualization in 3rd version
- tree/text sync hard to do well