CS 5/6110, Software Correctness Analysis, Spring 2022

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

Lecture 14 : First-Order Logic aka predicate logic

- Read from Bradley and Manna, Chapter 2
- Practice using the files checked into this directory
- Having good implementations of SAT-checking for decidable fragments of FOL is why formal methods and rigorous software testing have suddenly become practical!
- Alloy introduces a relational logic
  - FOL expression allowed, maybe even HOL
  - But all “proofs” are finite-model proofs
- Moral: Know relations, functions, FOL, proofs
  - To do well in software testing, specification, and correctness verification
  - “Noble bugs” a direct consequence of computability theory
    - “Stupid bugs” arise due to the Towers of the Babel

An example of FOL (“classical”)

- Some patients like every doctor
- No patient likes a quack
- Prove that no doctor is a quack

An example of FOL (“classical”)

- Let “P” be people containing x,y,z, …
- Let “p” be a patient-predicate – e.g. p(x)
- Let “d” be a doctor-predicate – e.g. d(y)
- Let “q” be a quack-predicate – e.g. q(z)
- Let “l” be the likes-predicate – e.g. l(x,y) [T/F], !l(x,z) [T/F] etc..
- Let’s now see how to model this problem
- Forall = “all” and ThereExists = ”exists”
- In Alloy it is all / some

An example of FOL (“classical”)

- Reminder: P, p, d, q, l are the symbols we have so far
- All quantification is over “P” or people

- Some patients like every doctor
  - exists x : p(x) and [all y : d(y) -> l(x,y)]

- No patient likes a quack
  - … give it a shot …
- Prove that no doctor is a quack
  - … state the proof-goal in FOL – give it a shot …

An example of FOL (“classical”)

- Reminder: P, p, d, q, l are the symbols we have so far
- All quantification is over “P” or people

- Some patients like every doctor
  - exists x : p(x) and [all y : d(y) -> l(x,y)]

- No patient likes a quack
  - all x,y : [p(x) & q(y) => ! l(x,y)]

- Prove that no doctor is a quack
  - Goal : prove that all x : d(x) => !q(x)

An example of FOL (“classical”)

- Reminder: P, p, d, q, l are the symbols we have so far
- All quantification is over “P” or people

- Some patients like every doctor
  - exists x : p(x) and [all y : d(y) -> l(x,y)]
    - question: Where else can we put “all y” ?

- No patient likes a quack
  - all x,y : [p(x) & q(y) => ! l(x,y)]

- Prove that no doctor is a quack
  - Goal : prove that all x : d(x) => !q(x)

An example of FOL (“classical”)

- Reminder: P, p, d, q, l are the symbols we have so far
- All quantification is over “P” or people

- Some patients like every doctor
  - exists x : p(x) and [all y : d(y) -> l(x,y)]
    - question: Where else can we put “all y” ?
    - Which of these “makes sense”
      - exists x : all y: [p(x) and [d(y) -> l(x,y)]]
      - All y : exists x : [p(x) and [d(y) -> l(x,y)]]

- No patient likes a quack
  - all x,y : [p(x) & q(y) => ! l(x,y)]

- Prove that no doctor is a quack
  - Goal : prove that all x : d(x) => !q(x)

Example of FOL (“classical”): Proof by contra

- A1: exists x : p(x) and [all y : d(y) -> l(x,y)]
- A2: all x,y : [p(x) & q(y) => ! l(x,y)]
- G: all x : d(x) => !q(x)
- NG: …give it a shot…

- A1 skolemize x with p0
- A1sk: p(p0) and [all y : d(y) -> l(p0,y)]

Example of FOL (“classical”): Proof by contra

- A1: exists x : p(x) and [all y : d(y) -> l(x,y)]
- A2: all x,y : [p(x) & q(y) => ! l(x,y)]
- G: all x : d(x) => !q(x)
- NG: exists x : d(x) & q(x)

- A1 skolemize x with p0
- A1sk: p(p0) and [all y : d(y) -> l(p0,y)]
- NGsk: Skolemize NG : d(d0) & q(d0)

Example of FOL (“classical”): Proof by contra

- A1: exists x : p(x) and [all y : d(y) -> l(x,y)]
- A2: all x,y : [p(x) & q(y) => ! l(x,y)]
- G: all x : d(x) => !q(x)
- NG: exists x : d(x) & q(x)
- A1sk: p(p0) and [all y : d(y) -> l(p0,y)]
- NGsk: d(d0) & q(d0)
- A1sk with d0 specialization: p(p0) & d(d0) => l(p0,d0)
- Now what?
  - Hint : we have not used A2

Example of FOL (“classical”): Proof by contra

- A1: exists x : p(x) and [all y : d(y) -> l(x,y)]
- A2: all x,y : [p(x) & q(y) => ! l(x,y)]
- G: all x : d(x) => !q(x)
- NG: exists x : d(x) & q(x)
- A1sk: p(p0) and [all y : d(y) -> l(p0,y)]
- NGsk: d(d0) & q(d0)
- A1sk with d0 specialization: p(p0) & d(d0) => l(p0,d0)
- Now what?
  - Hint : we have not used A2
- A2sp with p0 and d0: p(p0) & q(d0) => !l(p0,d0)
  - Do you smell the contradiction?
  - How to finish?

Example of FOL (“classical”): Proof by contra

- A1: exists x : p(x) and [all y : d(y) -> l(x,y)]
- A2: all x,y : [p(x) & q(y) => ! l(x,y)]
- G: all x : d(x) => !q(x)
- NG: exists x : d(x) & q(x)
- A1sk: p(p0) and [all y : d(y) -> l(p0,y)]
- NGsk: d(d0) & q(d0)
- A1sk with d0 specialization: (A1sk’): p(p0) & [d(d0) => l(p0,d0)]
- Now what?
- A2sp with p0 and d0: p(p0) & q(d0) => !l(p0,d0)
- MP of A1sk’ and NGsk : l(p0,d0)
- Contra of !l(p0,d0) and l(p0,d0)

Lecture 14 : First-Order Logic aka predicate logic : Basics

- There exist consistent axiomatizations of FOL
- There exist complete axiomatizations of FOL
- FOL is only semi-decidable
  - Only procedures exist for deciding validity
    - No algorithms
      - No TMs that don't loop
  - Semi-algorithms exist because of completeness
    - If indeed valid (true), there is a proof
    - And we can discover the proof
      - via systematic enumeration of all legal inference sequences

Encoding the “doctor/quack theorem” in Alloy

- What all FOL concepts are involved?

- How to encode each of them in Alloy?

Encoding the “doctor/quack theorem” in Alloy

- What all FOL concepts are involved?
  - Propositional variables
    - Zero-ary predicates
  - Predicates of higher arity
    - One-ary predicates (or subsets of the universe in most cases)
    - Two-ary predicates (pairs over U)
  - Zero-ary functions
    - Or constants in various value domains
  - One-ary and higher-arity functions

- How to encode each of them in Alloy?
  - We will study how the encoding goes for the doctor/quack theorem
  - Then we will study how to encode general FOL concepts using Alloy
    - Again, for bounded-model proofs
  - Look at Alloy-Constructs-and-Usage.txt for a quick overview

Alloy doctors/quacks encoding

```
some sig  P             -- our little universe of people
     { l : set P }      -- the "likes" binary relation
```

Alloy doctors/quacks encoding

```
some sig  P           -- our little universe of people
     { l : set P }    -- the "likes" binary relation

sig  pat  in P {}     -- patients unary relation
sig  doc  in P {}     -- doctor unary relation ; could overlap with pat
sig  quk  in P {}     -- think of quacks as any subset of people
```

Alloy doctors/quacks encoding

```
some sig  P           -- our little universe of people
     { l : set P }    -- the "likes" binary relation

sig  pat  in P {}     -- patients unary relation
sig  doc  in P {}     -- doctor unary relation ; could overlap with pat
sig  quk  in P {}     -- think of quacks as any subset of people


pred p[x: P] { x in pat } -- patient predicate denotes unary reln pat
pred d[x: P] { x in doc } -- doctor predicate  denotes unary reln doc
pred q[x: P] { x in quk } -- quack predicate
```

Alloy doctors/quacks encoding

```
some sig  P           -- our little universe of people
     { l : set P }    -- the "likes" binary relation

sig  pat  in P {}     -- patients unary relation
sig  doc  in P {}     -- doctor unary relation ; could overlap with pat
sig  quk  in P {}     -- think of quacks as any subset of people


pred p[x: P] { x in pat } -- patient predicate denotes unary reln pat
pred d[x: P] { x in doc } -- doctor predicate  denotes unary reln doc
pred q[x: P] { x in quk } -- quack predicate

-- some patients like every doctor
fact { some x : P | p[x] and (all y : P | d[y] => x->y in l) }
```

Alloy doctors/quacks encoding

```
some sig  P           -- our little universe of people
     { l : set P }    -- the "likes" binary relation

sig  pat  in P {}     -- patients unary relation
sig  doc  in P {}     -- doctor unary relation ; could overlap with pat
sig  quk  in P {}     -- think of quacks as any subset of people


pred p[x: P] { x in pat } -- patient predicate denotes unary reln pat
pred d[x: P] { x in doc } -- doctor predicate  denotes unary reln doc
pred q[x: P] { x in quk } -- quack predicate

-- some patients like every doctor
fact { some x : P | p[x] and (all y : P | d[y] => x->y in l) }

-- no patient likes a quack
fact { all x, y : P | (p[x] and q[y]) => x->y not in l }
```

Alloy doctors/quacks encoding

```
some sig  P           -- our little universe of people
     { l : set P }    -- the "likes" binary relation

sig  pat  in P {}     -- patients unary relation
sig  doc  in P {}     -- doctor unary relation ; could overlap with pat
sig  quk  in P {}     -- think of quacks as any subset of people


pred p[x: P] { x in pat } -- patient predicate denotes unary reln pat
pred d[x: P] { x in doc } -- doctor predicate  denotes unary reln doc
pred q[x: P] { x in quk } -- quack predicate

-- some patients like every doctor
fact { some x : P | p[x] and (all y : P | d[y] => x->y in l) }

-- no patient likes a quack
fact { all x, y : P | (p[x] and q[y]) => x->y not in l }

-- check that no doctor is a quack
assert docthm { all x : P | d[x] => !q[x] }
```

Alloy checks asserts for validity
by negating them
and checking whether they are sat or not
If Sat then … ?
If Unsat then … ?

Remember that any Boolean formula is
* Valid
* A contradiction (its negation is valid)
* Neither – it and its negation are satisfiable

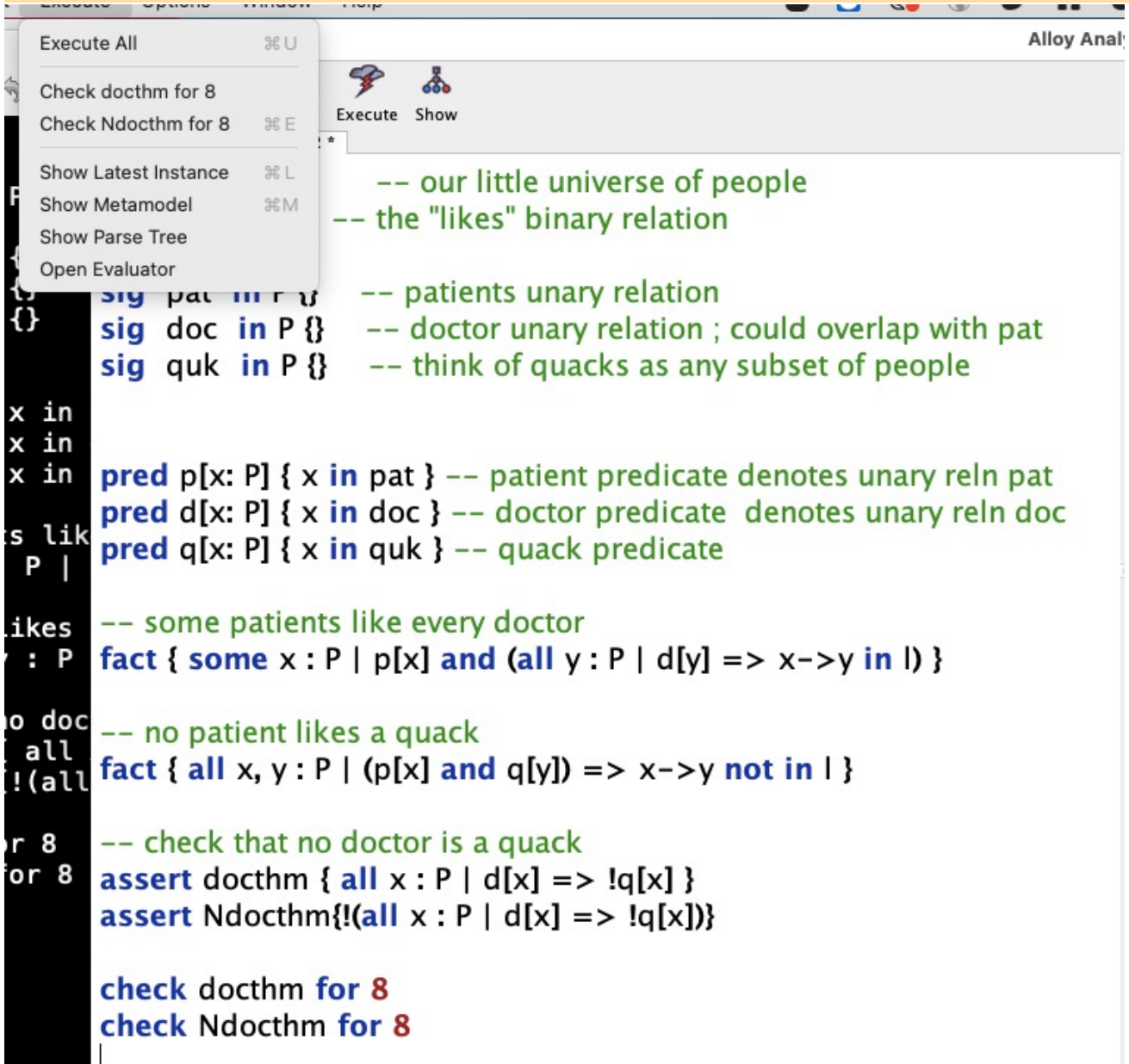
Alloy checks asserts for validity
by negating them
and checking whether they are sat or not
If Sat then the original assertion is invalid
If Unsat then the original assertion is valid
Just to be thorough, we will (later) do this:
* Write an assertion Assn
* Write negAssn = !Assn
* We will check both
* If Assn's "check" says "no counterexample, "Assn may be valid",
then ensure that !Assn is satisfiable (there is a counterexample for it)

BUT if both Assn and !Assn generate counterexamples, then they are
merely satisfiable but not valid AND also not contradictions !!

Alloy doctors/quacks encoding

```
some sig  P           -- our little universe of people
     { l : set P }    -- the "likes" binary relation

sig  pat  in P {}     -- patients unary relation
sig  doc  in P {}     -- doctor unary relation ; could overlap with pat
sig  quk  in P {}     -- think of quacks as any subset of people


pred p[x: P] { x in pat } -- patient predicate denotes unary reln pat
pred d[x: P] { x in doc } -- doctor predicate  denotes unary reln doc
pred q[x: P] { x in quk } -- quack predicate

-- some patients like every doctor
fact { some x : P | p[x] and (all y : P | d[y] => x->y in l) }

-- no patient likes a quack
fact { all x, y : P | (p[x] and q[y]) => x->y not in l }

-- check that no doctor is a quack
assert docthm { all x : P | d[x] => !q[x] }
assert Ndocthm{!(all x : P | d[x] => !q[x])}

check docthm for 8
check Ndocthm for 8
```

Alloy doctors/quacks encoding



Observe that
Via tbe pulldown

You can run any of the asserts
You've planted !!

Excerpts from Bradley and Manna
No slides

I'll scroll thru Chapter 2 to familiarize you

We will go thru how Alloy models these ideas, and let you read Chapter 2

We will then try and cover Hoare Logic before the Spring break – but no need to hurry if that does not happen!

Excerpts from Bradley and Manna: this is the kind of FOL formula we need to understand!

Example 2.3. In

$$\underbrace{\forall x.\ p(f(x),x)\ \rightarrow\ (\exists y.\ \underbrace{p(f(g(x,y)),g(x,y))}_{G})\ \wedge\ q(x,f(x))}_{F}\ ,$$

the scope of x is F, and the scope of y is G. This formula is read: "for all x, if $p(f(x),x)$ then there exists a y such that $p(f(g(x,y)),g(x,y))$ and $q(x,f(x))$". ■

Encoding FOL zero-ary predicates (Bools)

```
-- How to introduce FOL zero-ary functions or constants
one sig a,b extends U { } -- constants 'a' in U - guaranteed within U

some sig U {
 p1 : set U, -- p1 is an arbitrary unary predicate
 q1 : set U, -- q1 is an arbitrary unary predicate
 r0  : lone U, -- r0 is for the simulation of a zero-ary pred -- BOOL
 r1  : lone U, -- r1 is for .. another zero-ary pred -- BOOL
}

pred R0 { some a.r0 } -- Got one Bool Var! True if r0 is suitably assigned!
//-- sanity assert r0true { some a.r0 }
//-- may not be true - depends on how r0 is assigned!

pred R1 { some a.r1 } -- Got another Bool var!

assert r0a { R0 => (R1 => R0) } -- Now we can prove Boolean facts!
check r0a
```

Encoding higher arity predicates

```
some sig U {}
sig S1 in U {}
sig S2 in U {}
pred p1[x:U]   { x in S1 }              -- general-enough 1-ary pred
pred p2[x,y:U] { x in S1 and y in S2 } -- general-enough 2-ary pred

-- This is general-enough because
-- S1, S2 can grow from {} to the full U, and it allows x,y to be
-- either within S1 or S2
-- SANITY run { #U = 2 }

assert EA { some y:U | all x:U | p2[x,y]
            =>
            all  x:U | some y:U | p2[x,y]
          }
assert nEA { !(some y:U | all x:U | p2[x,y]
            =>
            all  x:U | some y:U | p2[x,y])
          }
check EA
check nEA
```

Functions encoded now

```
-- Now to simulate functions in Pred Logic (of arity higher)
some sig U {
  f1 : U,        -- one-ary fn
  f2 : U -> U,   -- two-ary fn
}
sig S1 in U {}
sig S2 in U {}
pred p1[x:U]   { x in S1 } -- general-enough 1-ary pred
pred p2[x,y:U] { x in S1 and y in S2 } -- general-enough 2-ary pred

assert EA { some y:U | all x:U | p2[x,f1[y] ]
            =>
            all  x:U | some y:U | p2[x,f1[y] ]
          }
assert nEA { !(some y:U | all x:U | p2[x, f2[y,y] ]
            =>
            all  x:U | some y:U | p2[x, f2[y,y]  ] )
          }
check EA
check nEA
```

Examples from Manna's original book about interpretation and validity

Example 1

Consider the formula

$$Fmla1 = \exists F.F(a) = b \\ \wedge(\forall x).[p(x) \Rightarrow F(x) = g(x, F(f(x)))]$$

We will now provide *three distinct* interpretations for it.

Interpretation 1.

$\mathrm{D} = \mathrm{Nat}$
$\mathrm{a} = 0$
$\mathrm{b} = 1$
$\mathrm{f} = \lambda x.(x = 0 \rightarrow 0, x - 1)$
$\mathrm{g} = *$
$\mathrm{p} = \lambda x.x > 0$

Interpretation 2.

$\mathrm{D} = \Sigma^*$
$\mathrm{a} = \varepsilon$
$\mathrm{b} = \varepsilon$
$\mathrm{f} = \lambda x.(tail(x))$
$\mathrm{g(x,y)} = \mathrm{concat(y,head(x))}$
$\mathrm{p} = \lambda x.x \neq \varepsilon$

Interpretation 3.

$\mathrm{D} = \mathrm{Nat}$
$\mathrm{a} = 0$
$\mathrm{b} = 1$
$\mathrm{f} = \lambda x.x$
$\mathrm{g(x,y)} = \mathrm{y+1}$
$\mathrm{p} = \lambda x.x > 0$

It is clear that under Interpretation 1, $Fmla1$ is true, because there indeed exists a function F, namely the factorial function, that makes the assertion true. It is also true under Interpretation 2, while it is false under Interpretation 3 (Exercise 18.4 asks for proofs). Hence, this formula is *not* valid — because it is not true under all interpretations.