CS 5/6110, Software Correctness Analysis, Spring 2021

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

Overview of lectures before/after break

- Today, 3/3
  - Show how FOL encodings can be made in Alloy
  - What tools exist and we can learn from
    - Gordon’s verifier
    - Hoare-verifier in Python + Z3
    - Dafny
    - Verifast (does also separation logic)
    - Left tutorials online
- After the break
  - Can we see where the state-of-the-art is in concurrent program verification?
    - Perhaps involves separation logic
      - Also perhaps involves the Rust language
  - This means we must study
    - First-Order Logic
    - Hoare Logic
    - Separation Logic
      - Would be good to read the PhD dissertation
        - “Understanding and evolving the Rust programming language”

Overview of lectures before/after break

- Assignment 6
  - Question 1:
    - Use the Alloy simulation of FOL and check validity (see next slide)
  - Question 2:
    - Read Mike-Gordon-Slides.pdf
      - Covers Hoare Logic, First-Order Logic, Separation Logic
    - Try the Gordon Prover
  - Question 3:
    - Project Selection
      - A 2-page proposal
      - Details TBD
      - John has been briefed

FOL
Work
book

proofs for

* (a) $P(b) \vdash \forall x\,(x = b \to P(x))$
  (b) $P(b), \forall x \forall y\,(P(x) \wedge P(y) \to x = y) \vdash \forall x\,(P(x) \leftrightarrow x = b)$
* (c) $\exists x\, \exists y\,(H(x, y) \vee H(y, x)), \neg \exists x\, H(x, x) \vdash \exists x \exists y\, \neg(x = y)$
  (d) $\forall x\,(P(x) \leftrightarrow x = b) \vdash P(b) \wedge \forall x \forall y\,(P(x) \wedge P(y) \to x = y)$.

* 12. Prove the validity of $S \to \forall x\, Q(x) \vdash \forall x\,(S \to Q(x))$, where S has arity 0 (a 'propositional atom').

13. By natural deduction, show the validity of
* (a) $\forall x\, P(a, x, x), \forall x\, \forall y\, \forall z\,(P(x, y, z) \to P(f(x), y, f(z)))$
  $\vdash P(f(a), a, f(a))$
* (b) $\forall x\, P(a, x, x), \forall x\, \forall y\, \forall z\,(P(x, y, z) \to P(f(x), y, f(z)))$
  $\vdash \exists z\, P(f(a), z, f(f(a)))$
* (c) $\forall y\, Q(b, y), \forall x\, \forall y\,(Q(x, y) \to Q(s(x), s(y)))$
  $\vdash \exists z\,(Q(b, z) \wedge Q(z, s(s(b))))$
  (d) $\forall x\, \forall y\, \forall z\,(S(x, y) \wedge S(y, z) \to S(x, z)), \forall x\, \neg S(x, x)$
  $\vdash \forall x\, \forall y\,(S(x, y) \to \neg S(y, x))$
  (e) $\forall x\,(P(x) \vee Q(x)), \exists x\, \neg Q(x), \forall x\,(R(x) \to \neg P(x)) \vdash \exists x\, \neg R(x)$
  (f) $\forall x\,(P(x) \to (Q(x) \vee R(x))), \neg \exists x\,(P(x) \wedge R(x)) \vdash \forall x\,(P(x) \to Q(x))$
  (g) $\exists x\, \exists y\,(S(x, y) \vee S(y, x)) \vdash \exists x\, \exists y\, S(x, y)$
  (h) $\exists x\,(P(x) \wedge Q(x)), \forall y\,(P(x) \to R(x)) \vdash \exists x\,(R(x) \wedge Q(x))$.

14. Translate the following argument into a sequent in predicate logic using a suitable set of predicate symbols:

If there are any tax payers, then all politicians are ta...

Material on Dafny (after the break)

Dafny Exercises: Loop invariant?

```
method g23(x:nat, n:nat) returns (P:nat)
ensures P == power(x,n)
{
  var X, N := x, n;
  P := 1;
  while (N != 0)
   decreases N
   {
        N := N - 1;
        P := P * X;
   }
}
```

Dafny Exercises: specificiation of power?

```
method g23(x:nat, n:nat) returns (P:nat)
ensures P == power(x,n)
{
  var X, N := x, n;
  P := 1;
  while (N != 0)
   decreases N
   invariant P * power(X,N) == power(x,n)
   {
         N := N - 1;
         P := P * X;
   }
}
```

Dafny Exercises

```
function power(x:nat, n:nat) : nat
{ if (n==0) then 1 else x * power(x, n-1) }

method g23(x:nat, n:nat) returns (P:nat)
ensures P == power(x,n)
{
  var X, N := x, n;
  P := 1;
  while (N != 0)
   decreases N
   invariant P * power(X,N) == power(x,n)
   {
         N := N - 1;
         P := P * X;
   }
}
```

Dafny Exercises: Ranking function (“decreases”)?

```
method gcdm(x:nat, y:nat) returns (G:nat)
requires x >= 0
requires y >= 0
ensures G == gcd(x,y)
{var X, Y := x, y;
  while (X != Y && X > 0 && Y > 0)
   {
    if (X > Y)
     { X := X – Y; }
    else
     { Y := Y – X; }
   }
   if (X == 0)
    { G := Y; }
   else
    { G := X; }
}
```

Dafny Exercises: Loop invariant?

```
method gcdm(x:nat, y:nat) returns (G:nat)
requires x >= 0
requires y >= 0
ensures G == gcd(x,y)
{var X, Y := x, y;
  while (X != Y && X > 0 && Y > 0)
  decreases X+Y
   {
    if (X > Y)
     { X := X - Y; }
    else
     { Y := Y - X; }
   }
   if (X == 0)
    { G := Y; }
   else
    { G := X; }
}
```

Dafny Exercises: Specification?

```
method gcdm(x:nat, y:nat) returns (G:nat)
requires x >= 0
requires y >= 0
ensures G == gcd(x,y)
{var X, Y := x, y;
  while (X != Y && X > 0 && Y > 0)
  decreases X+Y
  invariant gcd(X,Y) == gcd(x,y)
   {
    if (X > Y)
     { X := X - Y; }
    else
     { Y := Y - X; }
   }
   if (X == 0)
    { G := Y; }
   else
    { G := X; }
}
```

Dafny Exercises: Help prove termination!

```
// Not good enough! See errors later!

function gcd(x:nat, y:nat) : nat
requires x >= 0 && y >= 0
{ if (x==y) then y
  else if (x > y) then gcd(x-y, y)
  else gcd(x, y-x)
}

method gcdm(x:nat, y:nat) returns (G:nat)
requires x >= 0
requires y >= 0
ensures G == gcd(x,y)
{var X, Y := x, y;
  while (X != Y && X > 0 && Y > 0)
  decreases X+Y
  invariant gcd(X,Y) == gcd(x,y)
   {
    if (X > Y)
     { X := X - Y; }
    else
     { Y := Y - X; }
   }
   if (X == 0)
    { G := Y; }
   else
    { G := X; }
}

/*
dafny/dafny gcd4.dfy
/Users/ganesh/repos/software_correctness/Dafny/gcd4.dfy(5,23): Error: cannot prove termination;
pplying a decreases clause
Execution trace:
    (0,0): anon0
    (0,0): anon9_Else
    (0,0): anon10_Else
    (0,0): anon11_Then
/Users/ganesh/repos/software_correctness/Dafny/gcd4.dfy(6,7): Error: cannot prove termination;
```

Dafny Exercises: Finally!

```
//Verified!

function gcd(x:nat, y:nat) : nat
requires x >= 0 && y >= 0
decreases x+y
{ if      (x==0) then y
  else if (y==0) then x
  else if (x==y) then y
  else if (x > y) then gcd(x-y, y)
  else gcd(x, y-x)
}

method gcdm(x:nat, y:nat) returns (G:nat)
requires x >= 0
requires y >= 0
ensures G == gcd(x,y)
{var X, Y := x, y;
  while (X != Y && X > 0 && Y > 0)
  decreases X+Y
  invariant gcd(X,Y) == gcd(x,y)
   {
    if (X > Y)
     { X := X - Y; }
    else
     { Y := Y - X; }
   }
   if (X == 0)
    { G := Y; }
   else
    { G := X; }
}
```

Dafny Exercises:
Optimized exp
(unfinished..)

plz try with me!
Two approaches:
1) more simple
invariants
2) lemma

```
// Does not verify
function power(x:nat, n:nat) : nat
{ if (n==0) then 1 else x * power(x, n-1) }

method g23(x:nat, n:nat) returns (P:nat)
ensures P == power(x,n)
{ var X, N := x, n;
  P := 1;
  while (N != 0)
   decreases N
   invariant P * power(X,N) == power(x,n)
   { //--1
     if (N % 2 != 0)
       { P := P * X; }
     N := N / 2; //--2
     X := X * X; //--3
   }
}
//7^7 -> (X,N,P)=(7,7,1)->((7,7,7))->(7^2,3,7)
//     ->((7^2,3,7^3))->(7^4,1,7^3)
//     -> ((7^4,1,7^7))  ->(7^8,0,7^7)
//7^4 -> (X,N,P)=(7,4,1)->(7^2,2,1)->(7^4,1,1)
//     ->((7^4,1,7^4))->(7^8,0,7^4)->7^4
// P * power(X,N) == power(x,n)
// 3: P * power(X*X,N)    == power(x,n)
// 2: P * power(X*X,N/2) == power(x,n)
// 1: (N%2 != 0) ->
//        P*X * power(X*X,N/2) == power(x,n)
//      | P * power(X*X,N/2) == power(x,n)
// even(N) =>
//     power(X*X,N/2) == power(X,N) : yes
// odd(N) =>
//     P*power(X,N)
//     P*X*power(X^2,(N-1)/2)   : yes
```

Dafny Exercises: Lin search

```
method linsearch(a: array?<int>, key: int)
  returns (index: int)
requires a != null;
ensures 0 <= index ==> index < a.Length && a[index] == key;
// Incomplete - what else would you say?
{
  index := 0;
  while (index < a.Length)
  invariant 0 <= index <= a.Length;
  invariant ?
  {
   if a[index]==key { return; }
   index := index + 1;
  }
  index := -1;
}
```

Dafny Exercises:

```
method linsearch(a: array?<int>, key: int)
 returns (index: int) // low=0, high=a.Length
requires a != null;
ensures 0 <= index ==> index < a.Length && a[index] == key;
ensures index < 0  ==> forall i :: 0 <= i < a.Length ==> a[i] != key;
{
  index := 0;
  while (index < a.Length)
  invariant forall i :: 0 <= i < index ==> a[i] != key
  {
   if a[index]==key { return; }
   index := index + 1;
  }
  index := -1;
}
```

Dafny Exercises:

See other failures and successes – see files kept online

An intro to Hoare Logic and its rules

Follow Gordon’s prover code kept online

Dafny Exercises

- We will continue with more Dafny in Asgs
- Wed: Finish up Hoare Logic
- Then Static Analysis