Informal Methods

8409 characters in 2041 words on 277 lines

Florian Moser

January 12, 2021

1 basics

1.1 proof

convince other person of something that is true
way of communication

1.2 flow charts

rectangles for statements
diamond for conditions
lines to visualize control flow
triange with annotiation (predicate over state property)

show infinite loop
if path exist without state change
if path is possible (joined conditions fulfilable)

predicate
true at every execution (not just single one)
follows from previous predicate & loop condition
T (true / top) as most general (conveys no information)
$\perp$ (false, bottom) as most restrictive (unreachable)

1.3 program trace

statements actually executed by program
(can omit comparison checks)

1.4 predicate rules

conditions
if true, add condition to predicate
if false, add negation of condition

assignment (like x = y)
for before P(y), then afterwards P(x)
for afterwards Q(x), then before Q(y)

joins (two paths join)
for before P (one path) and Q (other path)
then afterwards P or Q

1.5 loop

structure
input (from init or after loop)
both supply invariant of loop
condition; if fulfilled execute body else exit
before body, have invariant + loop guard
before exit, have invariant + not loop guard

invariant
the creative step to prove property
choose as strong as needed to show property
choose as weak as able to proof
cannot be generated (reduces to halting problem)
show guard & invariant + loop execution $\Rightarrow$ invariant
show (not guard) & invariant $\Rightarrow$ property

1.6 hoare rules

predicate combinations

composition
if P $\rightarrow$ Q and Q $\rightarrow$ R
then P $\rightarrow$ R

if-rule
for P, G, R, S predices; A, B statements
(1) given P ˆ G & execute A $\rightarrow$ then R
(2) given P ˆ (not G) & execute B $\rightarrow$ then S
show that (1) and (2) hold, then (3) holds
(3) given P & execute if G $\rightarrow$ A else B $\rightarrow$ then R or S

loop rule
for P pre-condition, I loop invariant, G loop guard, Q post-condition
(1) P $\rightarrow$ I
(2) given G ˆ I & execute loop body $\rightarrow$ then I
(3) (not G) ˆ I $\rightarrow$ Q
show that (1), (2), (3) hold, then (4) holds
(4) given P & execute "do G $\rightarrow$ body" $\rightarrow$ then Q

1.7 frame conditions

conditions needed to establish real proof
like "array content is never modified"
need also to be proved; automatic verification can help

1.8 construction of algorithms

describe algorithm in terms of "general snapshot"
(hence describe when and to what single values change)
include any variables needed to be specific enough for algorithm
for all variables note its purpose and limits
then proof the required property indeed holds
particularly ensure each variable is initialized, changed and used

1.9 invariant structure

$>$ I
statement
$> I_2$

implications
I_2 must follow out of I and statement directly
no other references permitted (like previous loop iteration)

concurrency
for each process, combine each statement with each other process
combinatorial explosion (linear in size, exponential in #process)
show preconditions contradict
or show postconditions preserved

core invariant ideas
for loops, use "up to i, condition holds"
for sort, ensure multiset (elements) never changed
for dijkstra, ensure unexplored paths in set
for concurrency, use ghost lock variables

2 zune bug

broke devices on 1. Jan 2009
after leap year 2008
bc infinite loop (d=366, Lead(y) == true)

```
y := 1980
do d < 365 ->
if Leap(y) ->
if d > 366 ->
d,y := d-366,y+1
else
d,y := d-365, y+1
```

3 sample concurrency proof

show all data is passed from input over buffer to output
for three concurrent threads
S_x interpretations
S_0 = O[0, y)
S_{B2O} = O[y, h)
S_{B2} = B1[h upto n)

S_{B1B2} = B1[n upto m]
S_{B1} = B2[m upto t)
S_{IB1} = B2[t, x)
S_I = I[x, N)
invariant
> S_0 + S_{B2O} + S_{B2} + S_{B1B2} + S_{B1} + S_{IB1} + S_I = I[0, N) ^
h $\leq$ n ^ n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t ^ t-m $\leq$ N_B
main
> let I_ = S_0 + S_{B2O} + S_{B2} + S_{B1B2} + S_{B1} + S_{IB1} + S_I = I[0, N) ^
h $\leq$ n ^ n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t ^ t-m $\leq$ N_B
x, y, t, h, n, m := 0, 0, 0, 0, 0, 0
> I_ ^ x $\leq$ N ^ y $\leq$ N
> y = h
> x = t
> n = m
pbegin
S: do x != N $\rightarrow$
> I_ ^ x < N
> x = t
push()
> I_ ^ x $\leq$ N
> x = t
od
> I_ ^ x = N
//
R: do y != N $\rightarrow$
> I_ ^ y < N
> y = h
pop()
> I_ ^ y $\leq$ N
> y = h
od
> I_ ^ y = N
//
Q: do y != N $\rightarrow$
> I_
> m = n
transfer()
> I_
> m = n
od
> I_
pend
> I_ ^ y = N
push()
> let S_ = S_0 + S_{B2O} + S_{B2} + S_{B1B2}
> let I_ = h $\leq$ n ^ n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t
> S_ + S_{B1} + S_{IB1} + S_I = K ^
x < N ^ t-m $\leq$ N_B ^ I_
> t = x
wait until t - m = N_B
> S_ + B1[m upto t) + [] + (I[x] + I[x+1, N)) = K ^
x < N ^ t-m < N_B ^ I_
> t = x
B1[t mod N_B] := I[X]
> S_ + B1[m upto t) + [] + (B[t mod N_B] + I[x+1, N)) = K ^
x < N ^ t-m < N_B ^ I_
> t = x
x := x + 1
> S_ + B1[m upto t) + B[t mod N_B] + I[x, N) = K ^
x < N ^ t-m < N_B ^ I_
> t + 1 = x
t := t + 1
> S_ + B1[m upto t) + [] + I[x, N) = K ^
x < N ^ t-m $\leq$ N_B ^ I_
> t = x
for push, note that
- every first invariant is at least as strong as the overall invariant
- t = x ^ x < N are preconditions
- t = x is a postcondition / loop invariant of S
pop()
> let S_ = S_{B1B2} + S_{B1} + S_{IB1} + S_I
> let I_ = n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t ^ t-m $\leq$ N_B
> S_0 + S_{B2O} + S_{B2} + S_ = K ^
x < N ^ h $\leq$ n ^ I_
> y = h
wait until h < n
> O[0, y) + [] + (B2[h mod N_B] + B2[h+1 upto n)) + S_ = K ^
x < N ^ h < n ^ I_
> y = h
O[y] := B2[h mod N_B]
> O[0, y) + [] + (O[y] + B2[h+1 upto n)) + S_ = K ^
x < N ^ h < n ^ I_
> y = h
h := h + 1
> O[0, y) + O[y] + B2[h upto n) + S_ = K ^
x < N ^ h $\leq$ n ^ I_
> y+1 = h
y := y + 1
> O[0, y) + [] + B2[h upto n) + S_ = K ^
x < N ^ h $\leq$ n ^ I_
> y = h
for pop, note that
- every first invariant is at least as strong as the overall invariant
- y = h ^ y < N are preconditions
- y = h is a postcondition / loop invariant of R
transfer()
> let S_ = S_0 + S_{B2O} + S_{IB1} + S_I
> let I_ = h $\leq$ n ^ t-m $\leq$ N_B
> S_{B2} + S_{B1B2} + S_{B1} + S_ = K ^
n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t ^ I_
> n = m
wait until m < t
> B2[h upto n) + [] + (B1[m mod N_B] + B1[m+1 upto t)) + S_ = K ^
n-h $\leq$ N_B ^ n $\leq$ m ^ m < t ^ I_
> n = m
wait until n - h < N_B
> B2[h upto n) + [] + (B1[m mod N_B] + B1[m+1 upto t)) + S_ = K ^
n-h < N_B ^ n $\leq$ m ^ m < t ^ I_
> n = m
B2[n mod N_B] = B1[m mod N_B]
> B2[h upto n) + [] + (B2[n mod N_B] + B1[m+1 upto t)) + S_ = K ^
n-h < N_B ^ n $\leq$ m ^ m < t ^ I_
> n = m
m := m + 1
> B2[h upto n) + B2[n mod N_B] + B1[m upto t) + S_ = K ^
n-h < N_B ^ n < m ^ m $\leq$ t ^ I_
> n+1 = m
n := n + 1
> B2[h upto n) + [] + B1[m upto t) + S_ = K ^
n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t ^ I_
> n = m
for transfer, note that
- every first invariant is at least as strong as the overall invariant
- m = n is the precondition
- m = n is a postcondition / loop invariant of Q
non-interference argument
like in the lecture notes we look at all possible forms of invariants
then argue why no interference is possible (... that breaks these invariants)
> S_0 + S_{B2O} + S_{B2} + S_{B1B2} + S_{B1} + S_{IB1} + S_I = K ^ h $\leq$ n ^ n-h $\leq$ N_B ^ n $\leq$ m ^ m $\leq$ t ^ t-m $\leq$ N_B
we have shown at every point in the local correctness proof that the invariant was preserved
(at each first line of the invariants, there is some argument that is at least as strong as the invariant)
hence no interference possible
> x < N, x $\leq$ N
only local variable x and constant N is involved
hence no interference possible
> y < N, y $\leq$ N
only local variable y and constant N is involved
hence no interference possible
> t = x, t+1 = x
involves the shared variable t and the local variable x
but both t and x are only updated by the process S using the invariant
> y = h, y+1 = h
involves the shared variable h and the local variable y
but both h and y are only updated by the process R using the invariant
> t - m < N_B
only occurs in S (push)
the only statement which could invalidate it is in Q (transfer)
but Q only increments m (which can't break invariant)
> m < t
only occurs in Q (transfer)
the only statement which could invalidate it is in S (push)
but S only increments t (which can't break invariant)
> n - h < N_B
only occurs in Q (transfer)
the only statement which could invalidate it is in R (pop)
but R only increments h (which can't break invariant)
> h < n

only occurs in R (pop)
the only statement which could invalidate it is in Q (transfer)
but Q only increments n (which can't break invariant)
$>$ n $<$ m
only occurs in Q (transfer)
no other process changes either n or m

partial correctness argument
the invariant we ended up with was
$>$ S_0 + S_{B2O} + S_{B2} + S_{B1B2} + S_{B1} + S_{IB1} + S_I = I[0, N) ˆ
y = N
putting in the definition for S_0 yields
$>$ O[0, y) + S_{B2O} + S_{B2} + S_{B1B2} + S_{B1} + S_{IB1} + S_I = I[0, N) ˆ y =
N
out of O[0, y) and y = N we conclude that |O[0, y)| = |I[0, N)| = N
as no less-than-empty collections can exist, the length of the other
collections must be 0
replacing all other collections with the empty collection yields
$>$ O[0, y) + [] + [] + [] + [] + [] + [] = I[0, N)
hence it follows that
$>$ O[0, y) = I[0, N)