

More linear search with invariants

CS 5010 Program Design Paradigms
“Bootcamp”
Lesson 8.6



Lesson Introduction

- In this lesson, we'll show an example of how invariants can be used to improve a linear search.
- The transformation we'll use is called “reduction in strength”, and is a well-known algorithm that compilers use to improve the code in a loop.

Another example: Integer Square Root

int-sqrt : Nat -> Nat

GIVEN: n,

RETURNS: z such that $z^2 \leq n < (z+1)^2$

examples:

(int-sqrt 25) = 5

(int-sqrt 26) = 5 ...

(int-sqrt 35) = 5

(int-sqrt 36) = 6

This is one of my favorite examples.

Video Demonstration

- Watch the video demonstration at <http://www.youtube.com/watch?v=EW66F-vUApE>
- Note: the video is a little out of date:
 - it talks about accumulators instead of context arguments
 - the purpose statements are not always up to our current coding standards
 - sorry about that.
- Below are the slides from the video, slightly updated.

int-sqrt.v0

```
;; STRATEGY: Call more general function
(define (int-sqrt.v0 n)
  (linear-search 0 n
    (lambda (z)
      (< n (sqr (+ z 1))))))
```

int-sqrt.v1

```
(define (int-sqrt.v1 n)
  (local
    ((define (inner-loop z)
      ;; PURPOSE: Returns int-sqrt(n)
      ;; WHERE  $z^2 \leq n$ 
      ;; HALTING MEASURE  $(- n z)$ 
      (cond
        [(< n (sqr (+ z 1))) z]
        [else (inner-loop (+ z 1))]))
      (inner-loop 0)))
```

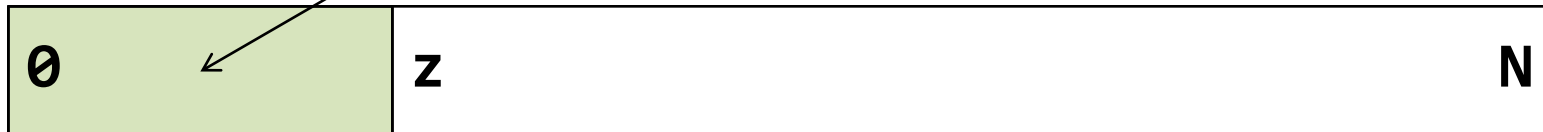
invariant guarantees that the halting measure is non-negative

we just checked that $(z+1)^2 \leq n$, so calling inner-loop with $z+1$ satisfies the invariant.

A picture of this invariant

INVARIANT: $z^2 \leq n$

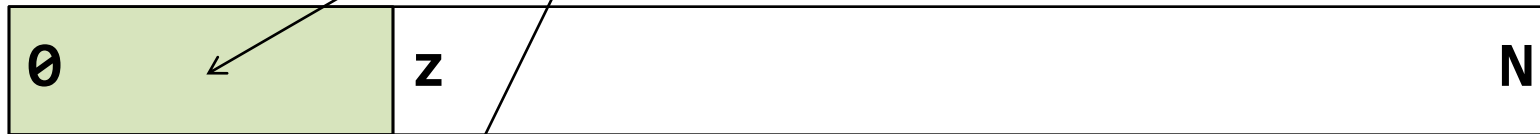
all these numbers also have squares that are $\leq n$



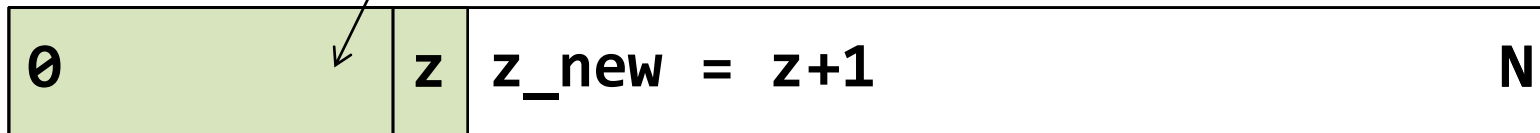
What happens at the recursive call?

INVARIANT: $z^2 \leq n$

all these numbers also have squares that are $\leq n$



CONDITION: $(z+1)^2 \leq n$



INVARIANT $z^2 \leq n$: true again for the new value of z.

Improving this code

Don't like to do **sqr** at every step, so let's keep the value of

(sqr (+ z 1))

in a context argument, which we'll call **u**.

Compute new value of **u** as follows:

$$z' = (z+1)$$

$$u' = (z'+1)*(z'+1)$$

$$= ((z+1)+1)*((z+1)+1)$$

$$= (z+1)^2 + 2(z+1) + 1$$

$$= u + 2z + 3$$

Function Definition

```
(define (int-sqrt.v2 n)
  (local
    ((define (inner-loop z u)
      ;; PURPOSE: Returns int-sqrt(n)
      ;; WHERE  $z^2 \leq n$ 
      ;; AND  $u = (z+1)^2$ 
      ;; HALTING MEASURE (- n z)
      (cond
        [(< n u) z]
        [else (inner-loop
              (+ 1 z)
              (+ u (* 2 z) 3))]))))
    (inner-loop 0 1)))
```

the inner loop finds the answer for the whole function

$$u = (z+1)^2$$

update context argument to maintain the invariant

initialize context argument to make the invariant true

Let's do it one more time

Add invariant: $v = 2*z+3$

$$z' = z+1$$

$$v' = 2*z' + 3$$

$$= 2*(z+1) + 3$$

$$= 2*z + 2 + 3$$

$$= (2*z + 3) + 2$$

$$= v + 2$$

Function Definition

```
(define (int-sqrt.v3 n)
  (local
    ((define (inner-loop z u v)
      ;; PURPOSE: Returns int-sqrt(n)
      ;; WHERE  $z^2 \leq n$ 
      ;; AND  $u = (z+1)^2$ 
      ;; AND  $v = 2*z+3$ 
      ;; HALTING MEASURE  $(- n z)$ 
      (cond
        [(< n u) z]
        [else (inner-loop
                (+ 1 z)
                (+ u v)
                (+ v 2))]))))
    (inner-loop 0 1 3)))
```

$$u = (z+1)^2$$

$$v = 2z+3$$

You could never understand this program if I hadn't written down the invariants!

Lesson Summary

- We've seen how invariants can be used to improve the code of a linear search or loop.
- We've seen how invariants can be used to explain the “reduction-in-strength” optimization.
- We've seen how invariants can be used to explain an otherwise-obscure piece of code.

Next Steps

- Study the file 08-8-square-roots.rkt in the Examples folder.
- Do the Guided Practices
- If you have questions about this lesson, ask them on the Discussion Board
- Go on to the next lesson