Lists

The definition of a list

The definition of a list

- The empty list is a list.
- A nonempty list `lst` has two parts.
  - `(car lst)` — the first element of the list.
  - `(cdr lst)` — the rest of the list.

The definition of a list

- The empty list is a list.
- A nonempty list `lst` has two parts.
  - `(car lst)` — the first element of the list.
  - `(cdr lst)` — the rest of the list.

`car` is an element.

`cdr` is a list.

`interleave`

```
(interleave ‘(a b c) ‘(d e f))
```

`interleave`

```
(interleave ‘(a b c) ‘(d e f))

(car ‘(a b c))
        a
```

interleave

```
(interleave ‘(a b c) ‘(d e f))

(car ‘(a b c))          (cdr ‘(a b c))
      a                       (b c)
```

interleave

```
(interleave ‘(a b c) ‘(d e f))

(car ‘(a b c))        (cdr ‘(a b c))
      a                     (b c)

    (interleave ‘(d e f) ‘(b c))
            (d b e c f)
```

```
interleave

(interleave ‘(a b c) ‘(d e f))

(car ‘(a b c))          (cdr ‘(a b c))
        a                     (b c)


    (interleave ‘(d e f) ‘(b c))
            (d b e c f)
             ↑

             a
```

add-to-end

```
(add-to-end ‘(a b c d) ‘x)
```

```
add-to-end

(add-to-end ‘(a b c d) ‘x)

(car ‘(a b c d))
a
```

```
add-to-end
```

```
(add-to-end ‘(a b c d) ‘x)
```

```
(car ‘(a b c d))          (cdr ‘(a b c d))
      a                         (b c d)
```

add-to-end

```
(add-to-end ‘(a b c d) ‘x)
```

```
(car ‘(a b c d))        (cdr ‘(a b c d))
        a                        (b c d)
```

```
(add-to-end ‘(b c d) ‘x)
     (b c d x)
```

```
add-to-end
```

```
(add-to-end '(a b c d) 'x)
```

```
(car '(a b c d))        (cdr '(a b c d))
       a                       (b c d)
```

```
(add-to-end '(b c d) 'x)
(b c d x)
```

```
a
```

```
(define add-to-end
  (lambda (lst elt)
    (if (null? lst)
        (cons elt '())
        (cons (car lst)
              (add-to-end (cdr lst) elt)))))
```

What is the efficiency of `add-to-end`?

```
(define add-to-end
  (lambda (lst elt)
    (if (null? lst)
        (cons elt '())
        (cons (car lst)
              (add-to-end (cdr lst) elt)))))
```

What is the efficiency of `add-to-end`?

$$\Theta(n)$$

```
(define my-reverse
  (lambda (lst)
    (if (null? lst)
        '()
        (add-to-end (my-reverse (cdr lst)) (car lst)))))
```

What is the efficiency of `my-reverse`?

```
(define my-reverse
  (lambda (lst)
    (if (null? lst)
        '()
        (add-to-end (my-reverse (cdr lst)) (car lst)))))
```

What is the efficiency of `my-reverse`?

$$\Theta(n^2)$$

```
(reverse-onto '(a b c) '(1 2 3))
      (c b a 1 2 3)
```

```
(reverse-onto '(a b c) '(1 2 3))
              (c b a 1 2 3)
```

```
(car ‘(a b c))
      a
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(car ‘(a b c))
a
```

```
(cdr ‘(a b c))
(b c)
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(car ‘(a b c))
a
```

```
(cdr ‘(a b c))
(b c)
```

```
(cons ‘a ‘(1 2 3))
(a 1 2 3)
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(car ‘(a b c))
a
```

```
(cdr ‘(a b c))
(b c)
```

```
(cons ‘a ‘(1 2 3))
(a 1 2 3)
```

```
(reverse-onto '(b c) '(a 1 2 3))
(c b a 1 2 3)
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
                      (cons (car lst1) lst2)))))
```

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
                      (cons (car lst1) lst2)))))
```

What is the efficiency of `reverse-onto`?

```
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
```

```
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
                      (cons (car lst1) lst2)))))
```

What is the efficiency of `reverse-onto`?

$$\Theta(n)$$

```
;; Efficient version of reverse
(define your-reverse
```

```
;; Efficient version of reverse
(define your-reverse
  (lambda (lst)
```

```
;; Efficient version of reverse
(define your-reverse
  (lambda (lst)
    (define reverse-onto
      (lambda (lst1 lst2)
        (if (null? lst1)
            lst2
            (reverse-onto (cdr lst1)
                          (cons (car lst1) lst2)))))
```

```
;; Efficient version of reverse
(define your-reverse
  (lambda (lst)
    (define reverse-onto
      (lambda (lst1 lst2)
        (if (null? lst1)
            lst2
            (reverse-onto (cdr lst1)
                          (cons (car lst1) lst2)))))
    (reverse-onto lst '())))
```

```
(merge '(2 4 6 8) '(1 3 5 8 9))
       (1 2 3 4 5 6 8 9)
```

```
(merge '(2 4 6 8) '(1 3 5 8 9))
         (1 2 3 4 5 6 8 9)
```

```
(car ‘(2 4 6 8))
         2
```

```
(merge '(2 4 6 8) '(1 3 5 8 9))
       (1 2 3 4 5 6 8 9)
```

```
(car ‘(2 4 6 8))    (car ‘(1 3 5 8 9))
        2                    1
```

```
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)
```

```
(car ‘(2 4 6 8))
2
```

```
(car ‘(1 3 5 8 9))
1

(cdr ‘(1 3 5 8 9))
(3 5 8 9)
```

```
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)
```

```
(car ‘(2 4 6 8))
2
```

```
(car ‘(1 3 5 8 9))
1
```

```
(cdr ‘(1 3 5 8 9))
(3 5 8 9)
```

```
(merge '(2 4 6 8) '(3 5 8 9))
(2 3 4 5 6 8 9)
```

```
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)
```

```
(car ‘(2 4 6 8))
        2
```

```
(car ‘(1 3 5 8 9))
         1
```

```
(cdr ‘(1 3 5 8 9))
     (3 5 8 9)
```

```
(merge '(2 4 6 8) '(3 5 8 9))
(2 3 4 5 6 8 9)
 ↑
 1
```

```
(odd-part '(g i r a f f e))
          (g r f e)
```

```
(odd-part '(g i r a f f e))
(g r f e)
```

```
(car ‘(g i r a f f e))
g
```

```
(odd-part '(g i r a f f e))
(g r f e)
```

```
(car ‘(g i r a f f e))
g
```

```
(cdr ‘(g i r a f f e))
(i r a f f e)
```

```
(odd-part '(g i r a f f e))
(g r f e)

(car ‘(g i r a f f e))
g

(cdr ‘(g i r a f f e))
(i r a f f e)

(even-part ‘(i r a f f e))
(r f e)
```

```
(odd-part '(g i r a f f e))
(g r f e)

(car ‘(g i r a f f e))
g

(cdr ‘(g i r a f f e))
(i r a f f e)

(even-part ‘(i r a f f e))
(r f e)
```

g

A child at the county fair wins 5 tickets.

The redemption store carries the following items, priced in tickets:

a — apples, 3 tickets each
b — balls, 3 tickets each
c — cookies, 2 tickets each
d — dolls, 1 ticket each
e — ear muffs, 1 ticket each

A child at the county fair wins 5 tickets.

The redemption store carries the following items, priced in tickets:

a — apples, 3 tickets each
b — balls, 3 tickets each
c — cookies, 2 tickets each
d — dolls, 1 ticket each
e — ear muffs, 1 ticket each

In how many ways can the child spend her tickets?

```
 a b c d e
(3 3 2 1 1)
```

```
 a b c d e
(3 3 2 1 1)
```

```
ac
add
ade
aee
```

```
 a b c d e
(3 3 2 1 1)
```

```
ac     bc
add    bdd
ade    bde
aee    bee
```

```
 a  b  c  d  e
(3  3  2  1  1)

ac      bc      ccd
add     bdd     cce
ade     bde     cddd
aee     bee     cdde
                cdee
                ceee
```

```
 a b c d e
(3 3 2 1 1)
```

ac
add
ade
aee

bc
bdd
bde
bee

ccd
cce
cddd
cdde
cdee
ceee

ddddd
dddde
dddee
ddeee
deeee

```
 a b c d e
(3 3 2 1 1)
```

```
ac     bc     ccd     ddddd   eeeee
add    bdd    cce     dddde
ade    bde    cddd    dddee
aee    bee    cdde    ddeee
              cdee    deeee
              ceee
```

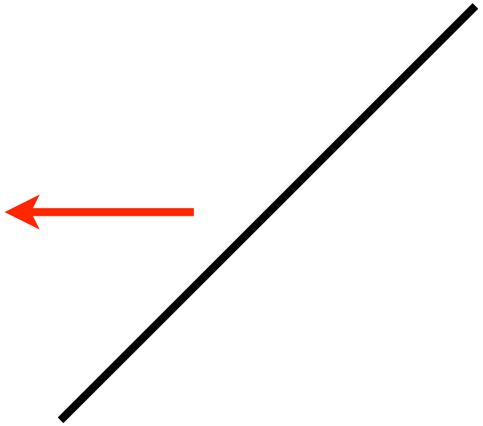
```
 a  b  c  d  e
(3  3  2  1  1)
```

```
ac       bc       ccd       ddddd    eeeee
add      bdd      cce       dddde
ade      bde      cddd      dddee
aee      bee      cdde      ddeee
                  cdee      deeee
                  ceee
```

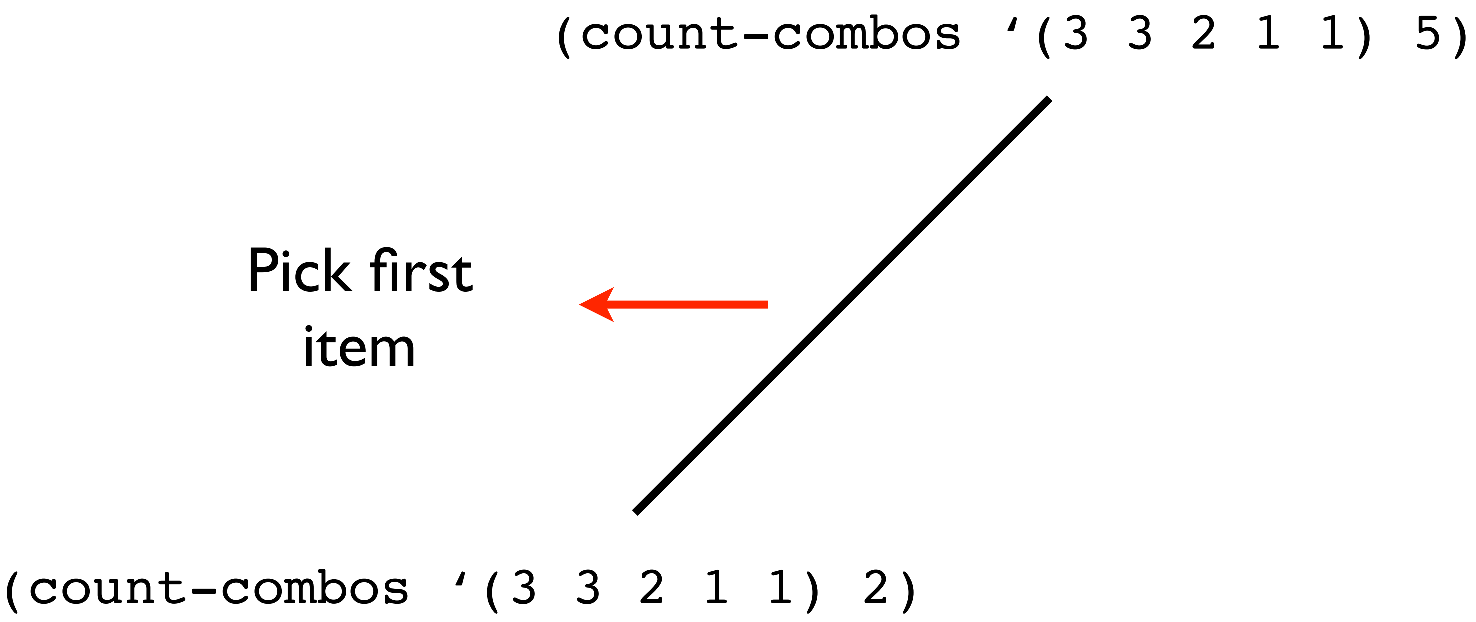
```
(count-combos '(3 3 2 1 1) 5)
20
```

```
(count-combos ‘(3 3 2 1 1) 5)
```

```
(count-combos ‘(3 3 2 1 1) 5)
```



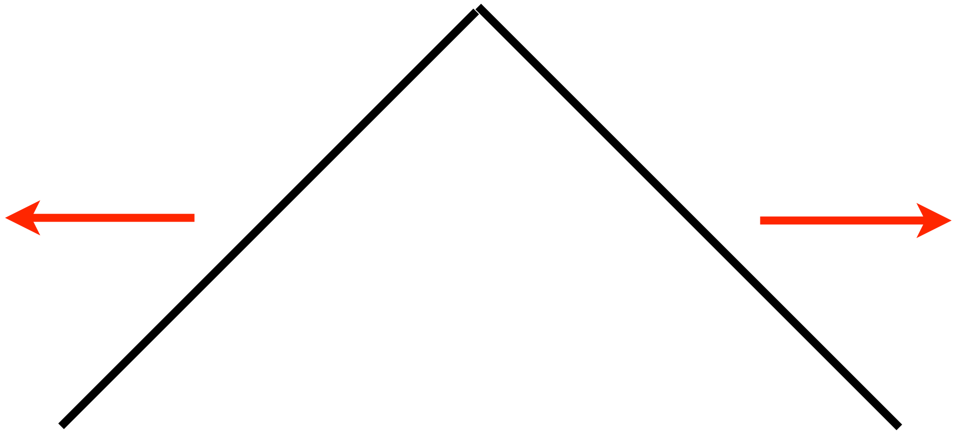
Pick first item



```
(count-combos ‘(3 3 2 1 1) 5)
```

Pick first item

```
(count-combos ‘(3 3 2 1 1) 2)
```

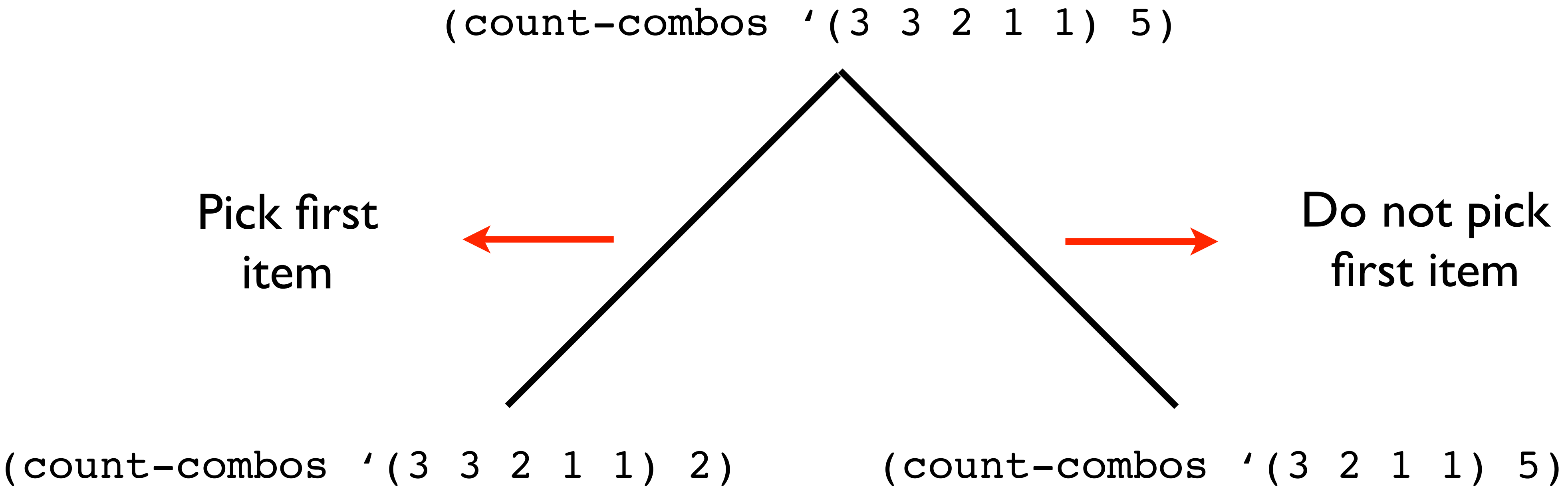


`(count-combos ‘(3 3 2 1 1) 5)`

Pick first item

Do not pick first item

`(count-combos ‘(3 3 2 1 1) 2)`



`(count-combos ‘(3 3 2 1 1) 5)`

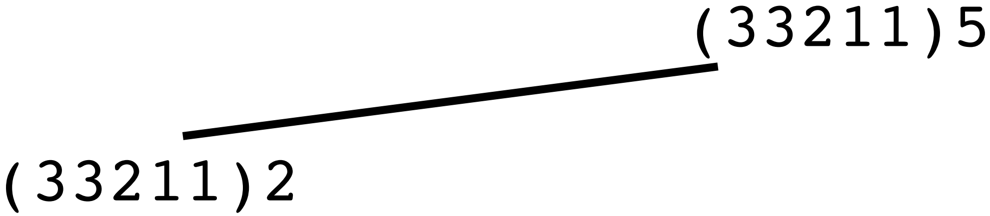
Pick first item

Do not pick first item

`(count-combos ‘(3 3 2 1 1) 2)`

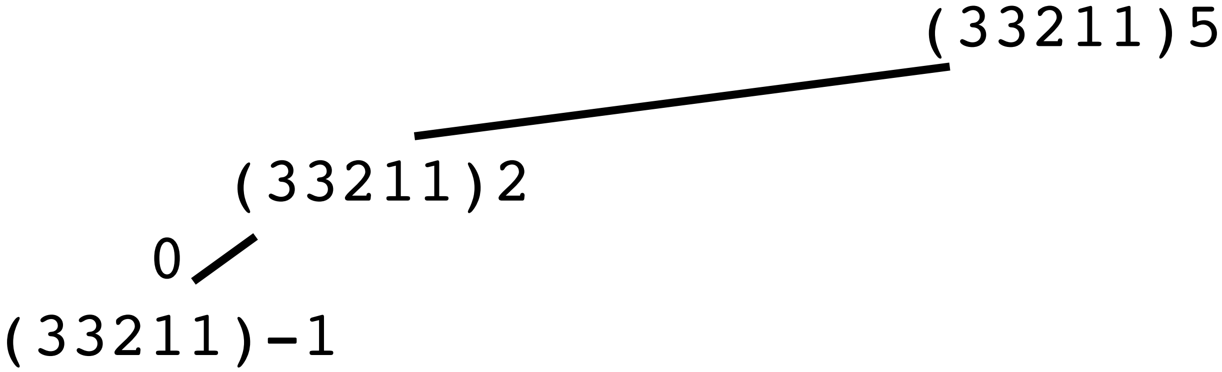
`(count-combos ‘(3 2 1 1) 5)`

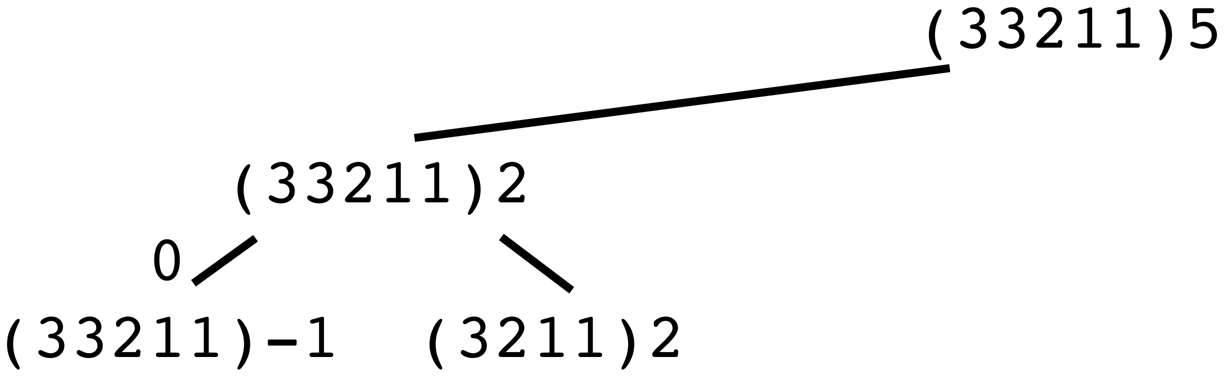
```
(33211)5
```

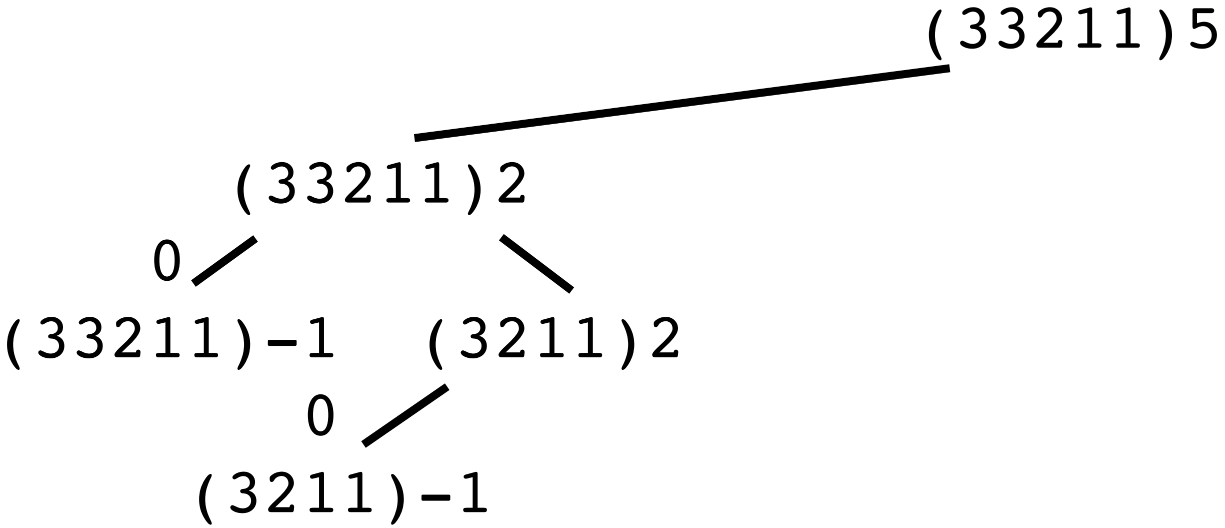


`(33211)5`

`(33211)2`


(33211)5
(33211)2
0
(33211)-1


(33211)5
(33211)2
0
(33211)-1
(3211)2



(33211)5

(33211)2

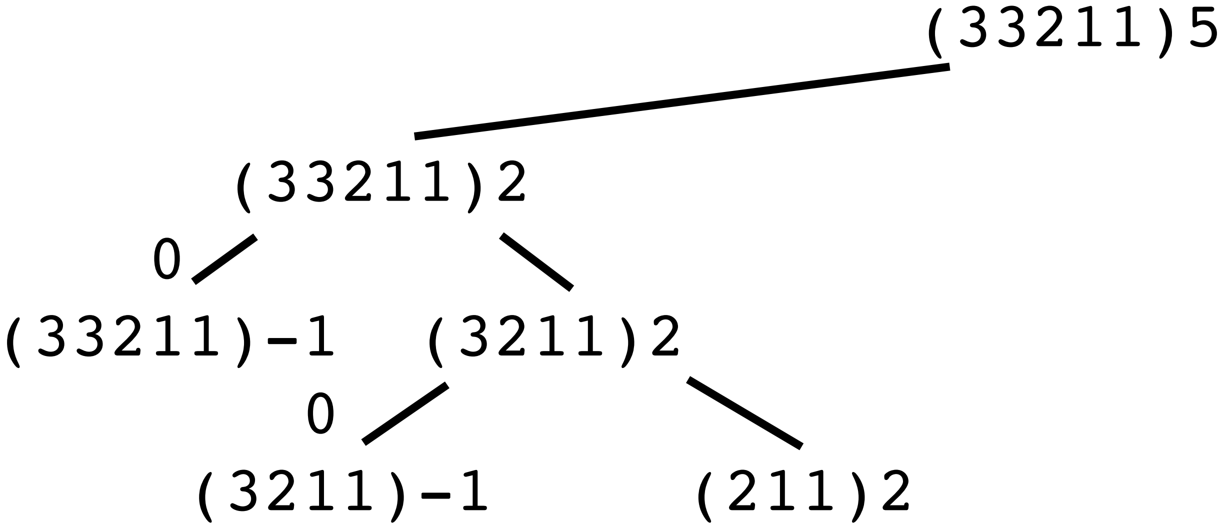
0

(33211)-1

(3211)2

0

(3211)-1



(33211)5

(33211)2

0

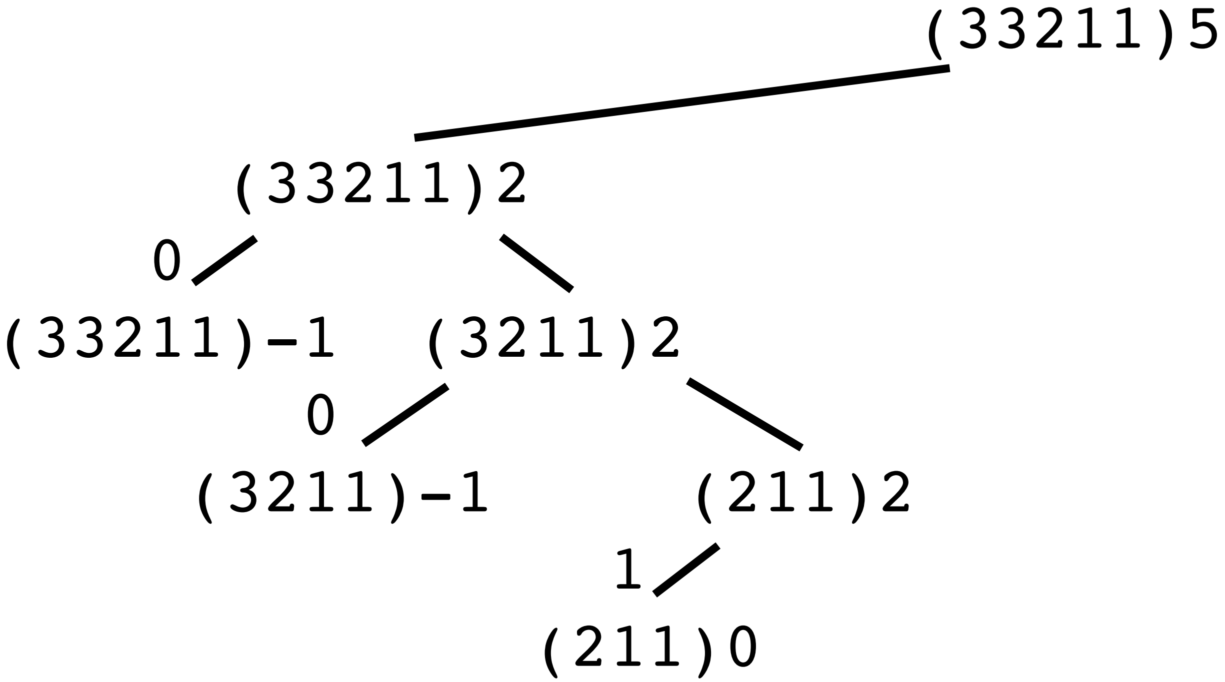
(33211)-1

(3211)2

0

(3211)-1

(211)2



(33211)5

(33211)2

0

(33211)-1

(3211)2

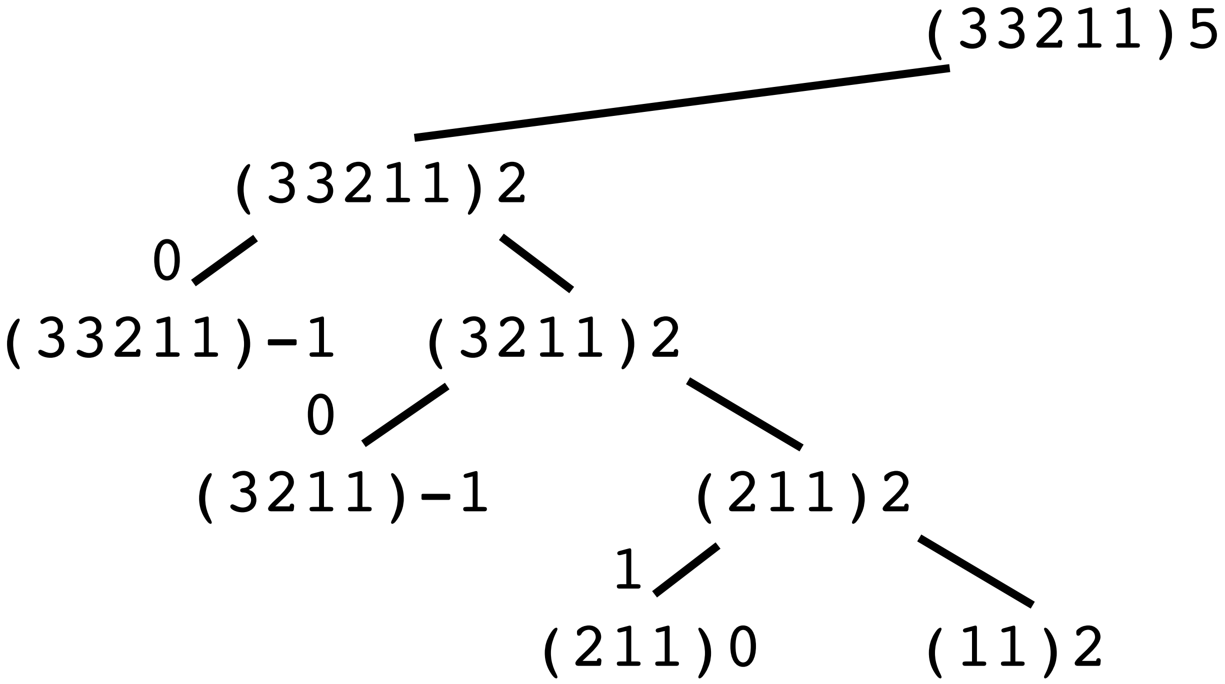
0

(3211)-1

(211)2

1

(211)0



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

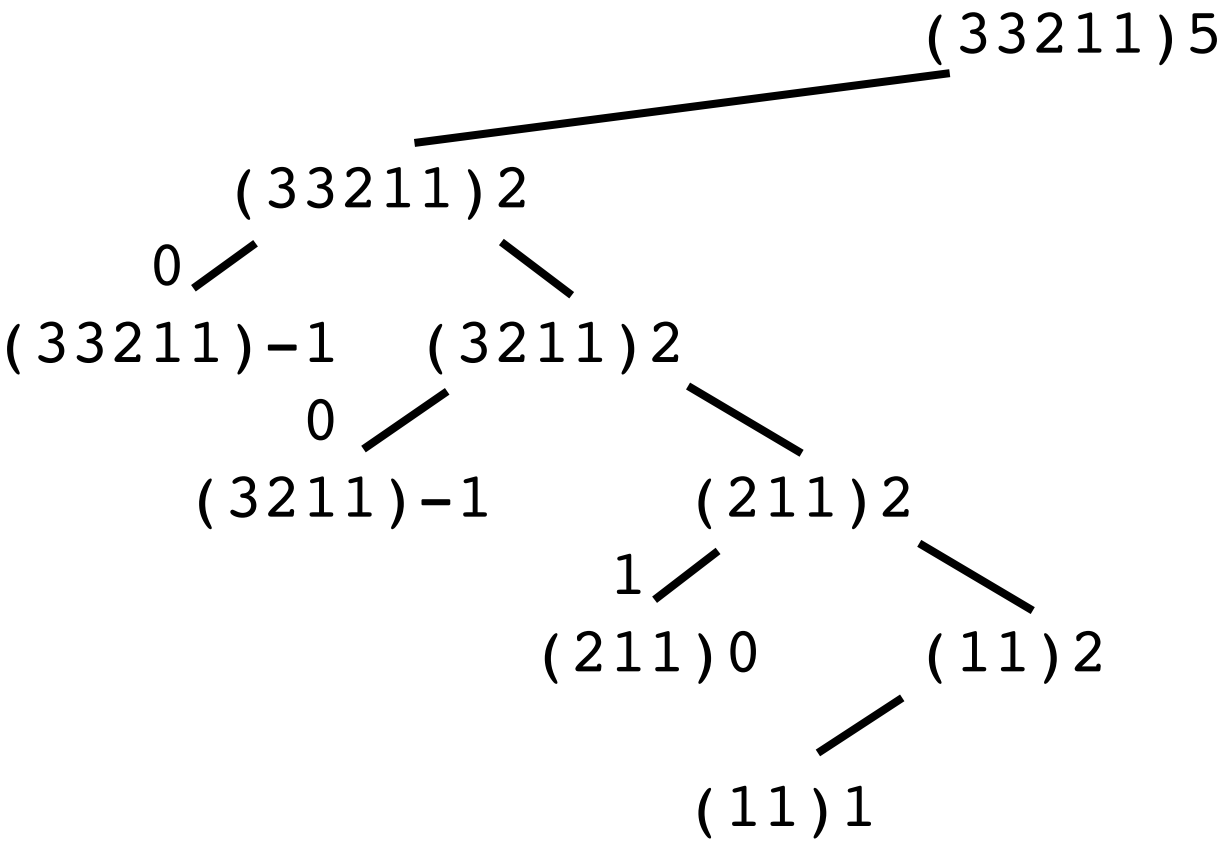
(3211)-1

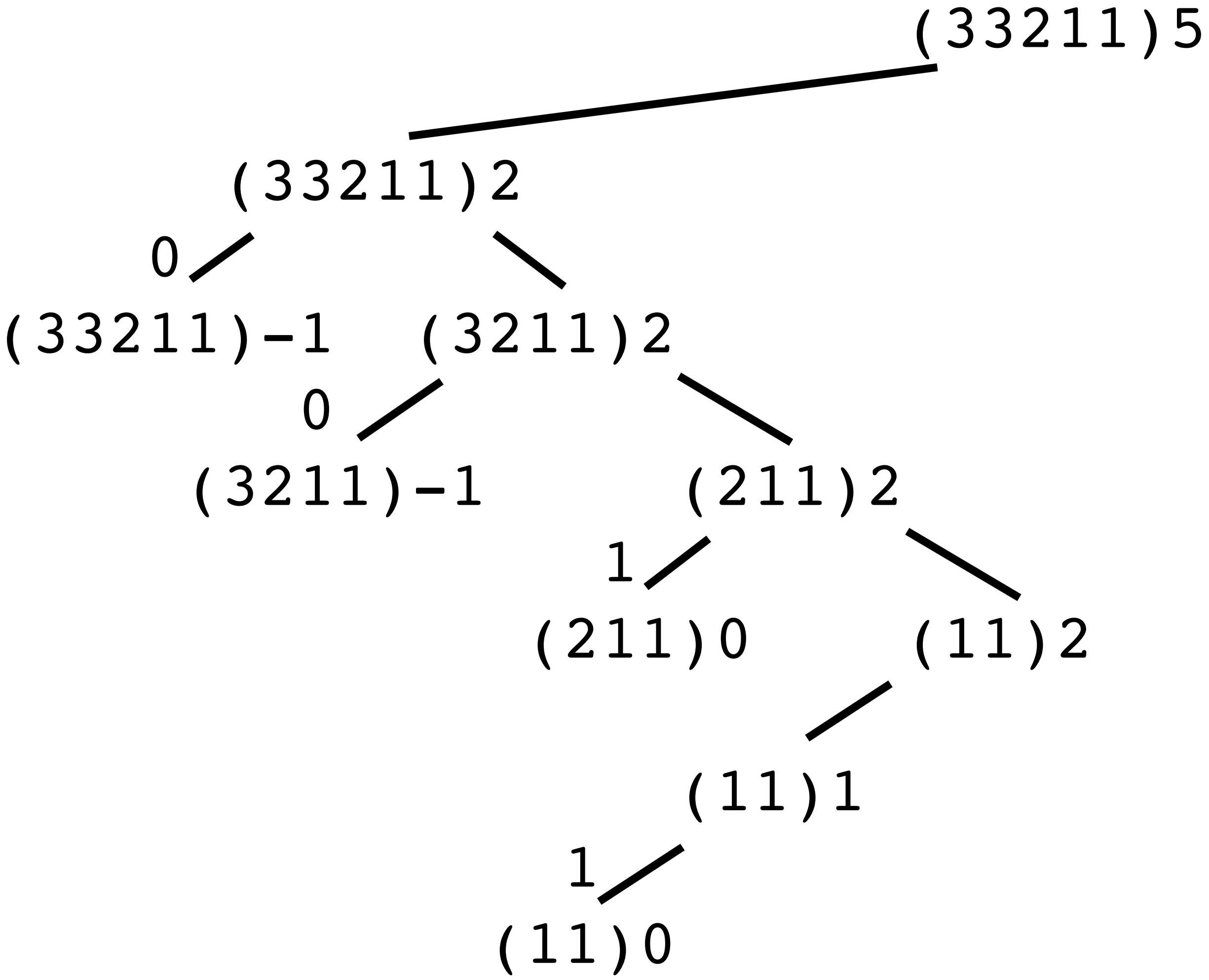
(211)2

1

(211)0

(11)2


(33211)5
(33211)2
0
(33211)-1
(3211)2
0
(3211)-1
(211)2
1
(211)0
(11)2
(11)1



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

(3211)-1

(211)2

1

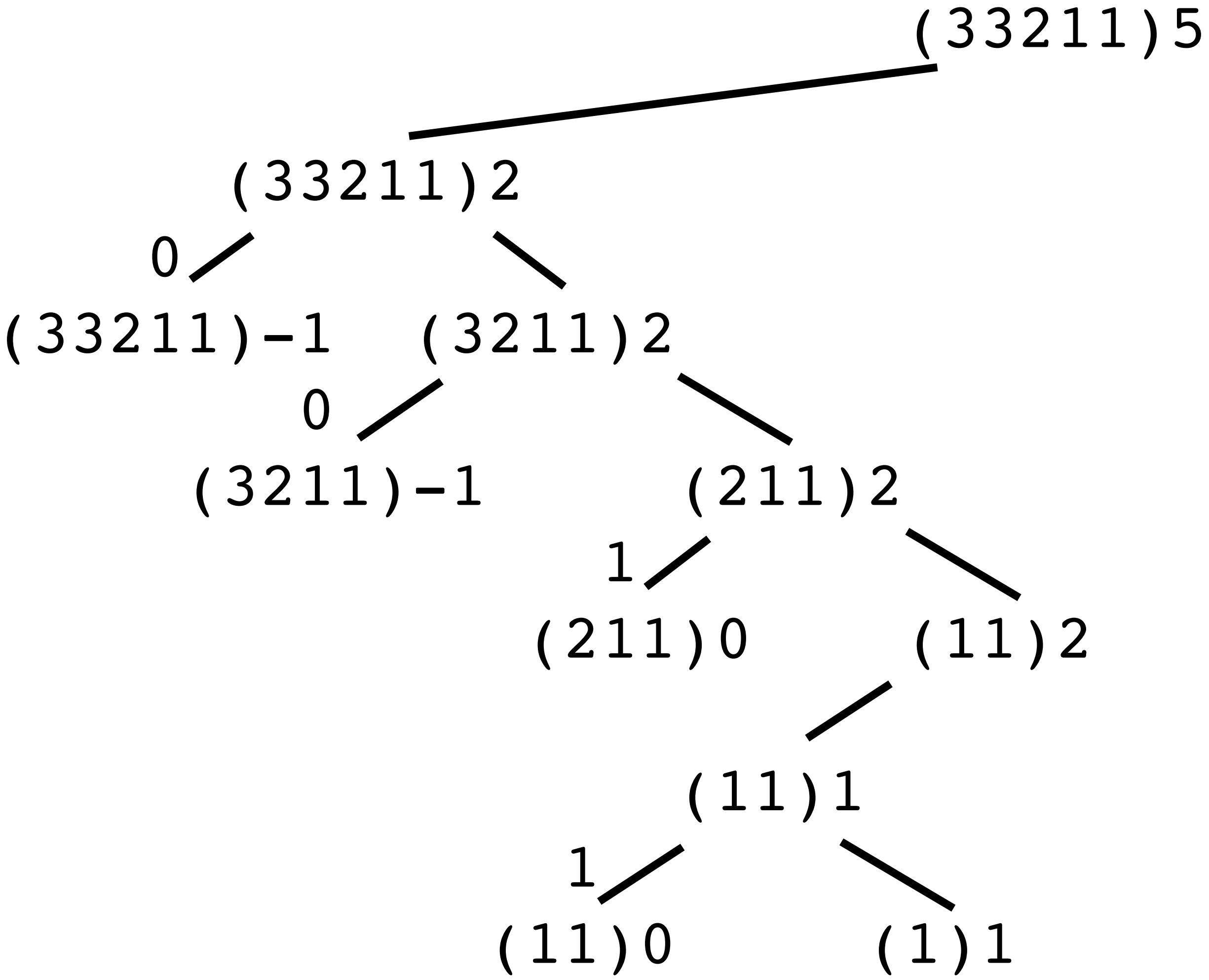
(211)0

(11)2

(11)1

1

(11)0



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

(3211)-1

(211)2

1

(211)0

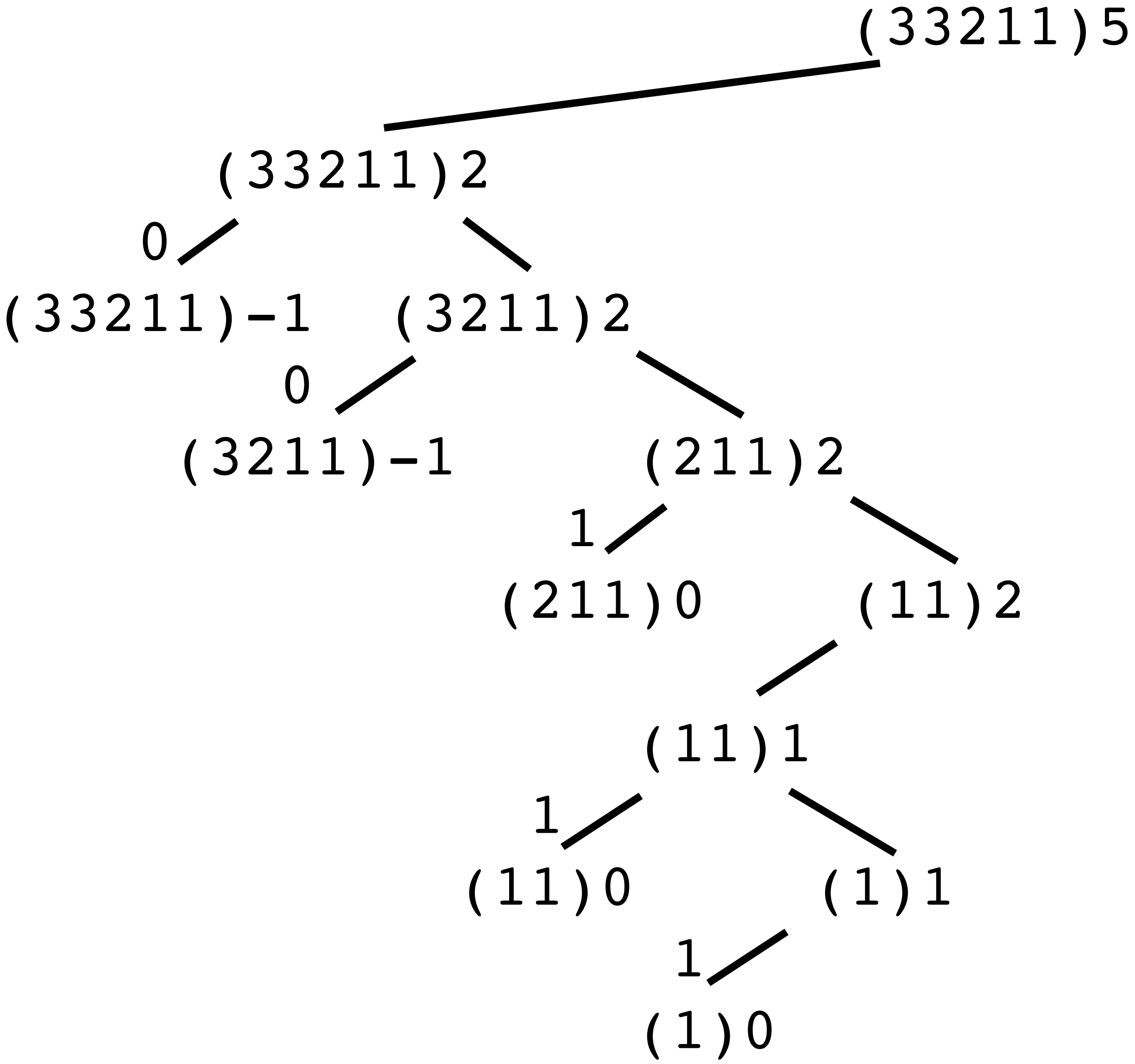
(11)2

(11)1

1

(11)0

(1)1



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

(3211)-1

(211)2

1

(211)0

(11)2

(11)1

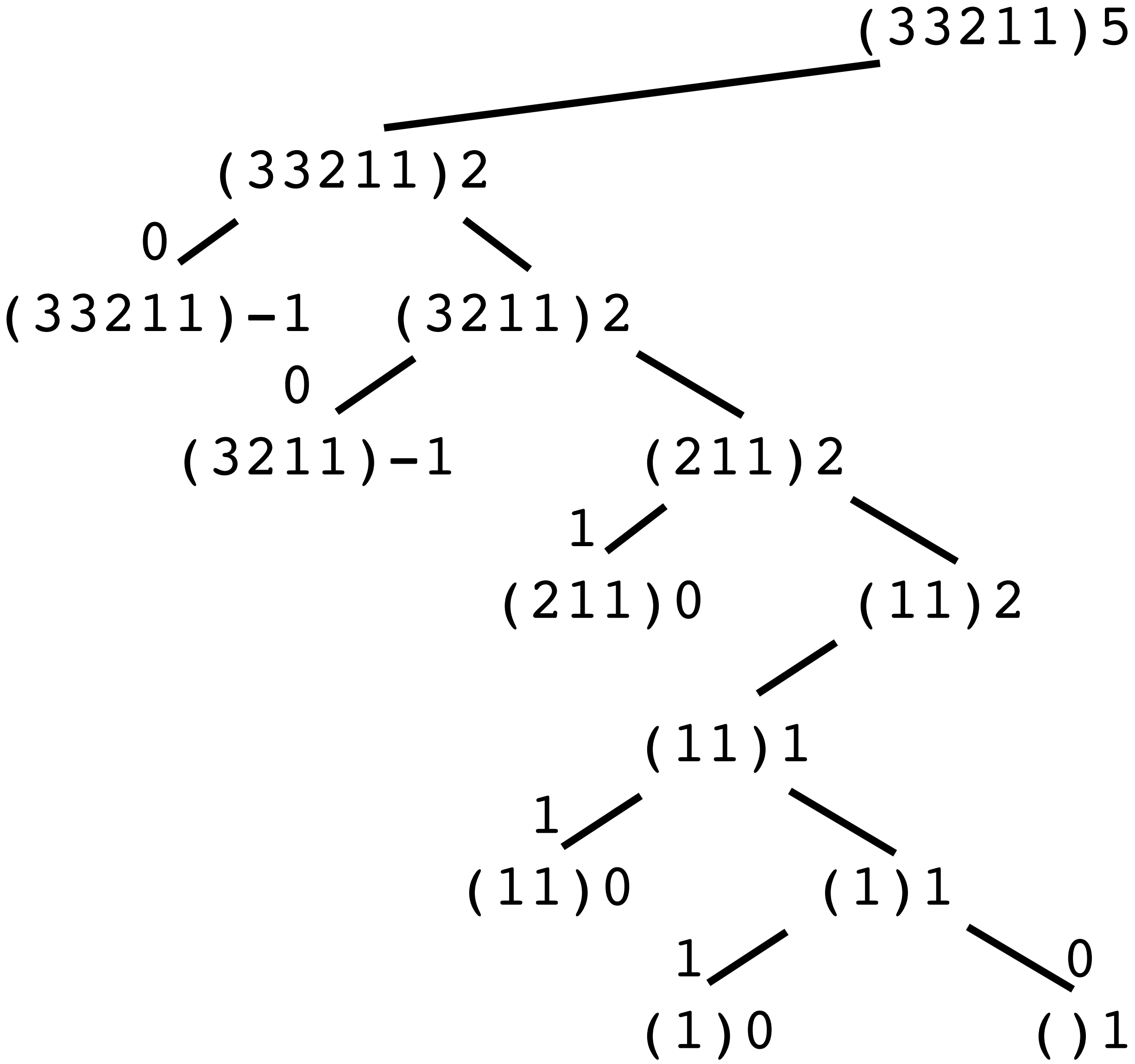
1

(11)0

(1)1

1

(1)0



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

(3211)-1

(211)2

1

(211)0

(11)2

(11)1

1

(11)0

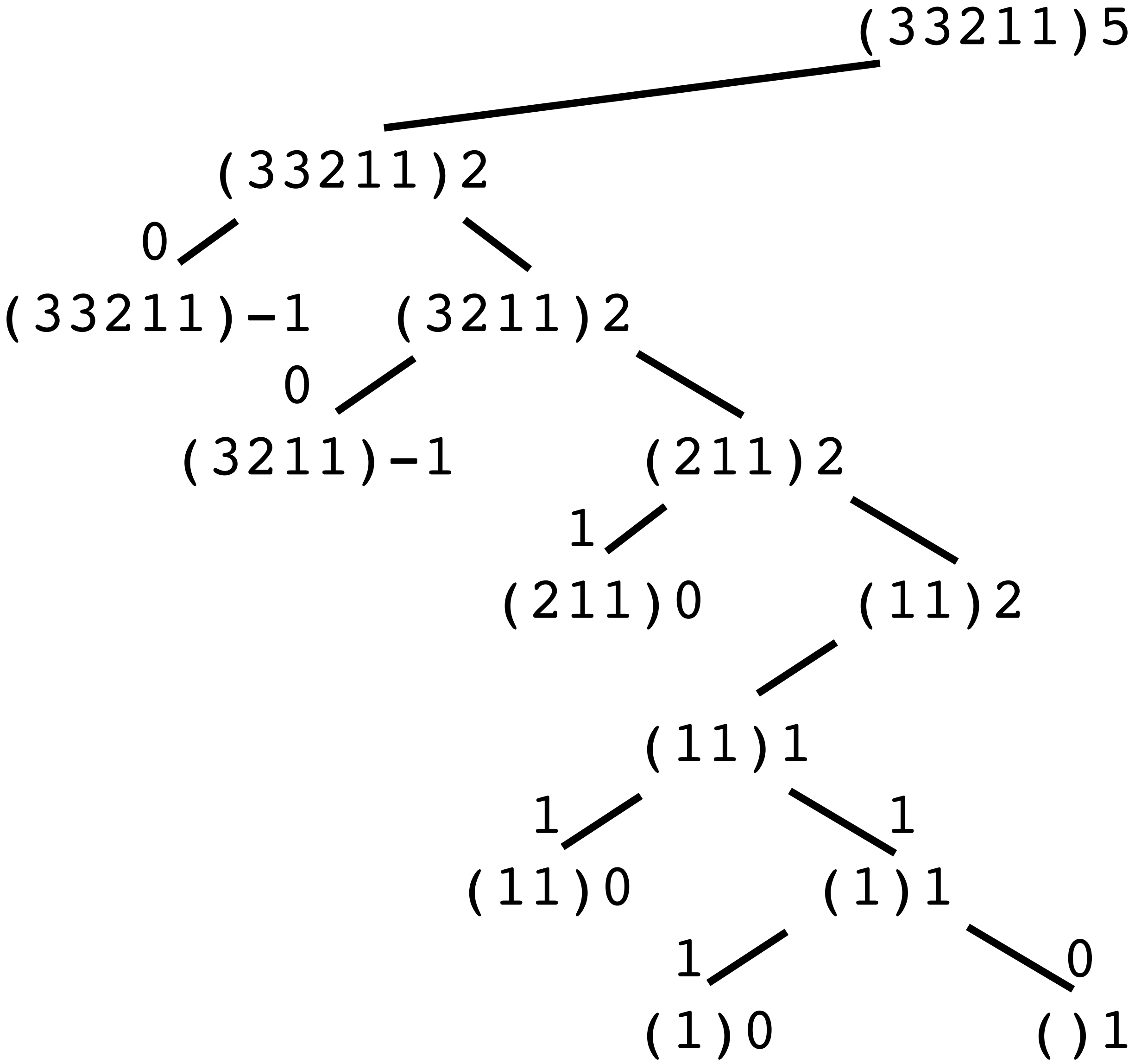
(1)1

1

(1)0

0

()1



(33211)5

(33211)2

0

(33211)-1 (3211)2

0

(3211)-1 (211)2

1

(211)0 (11)2

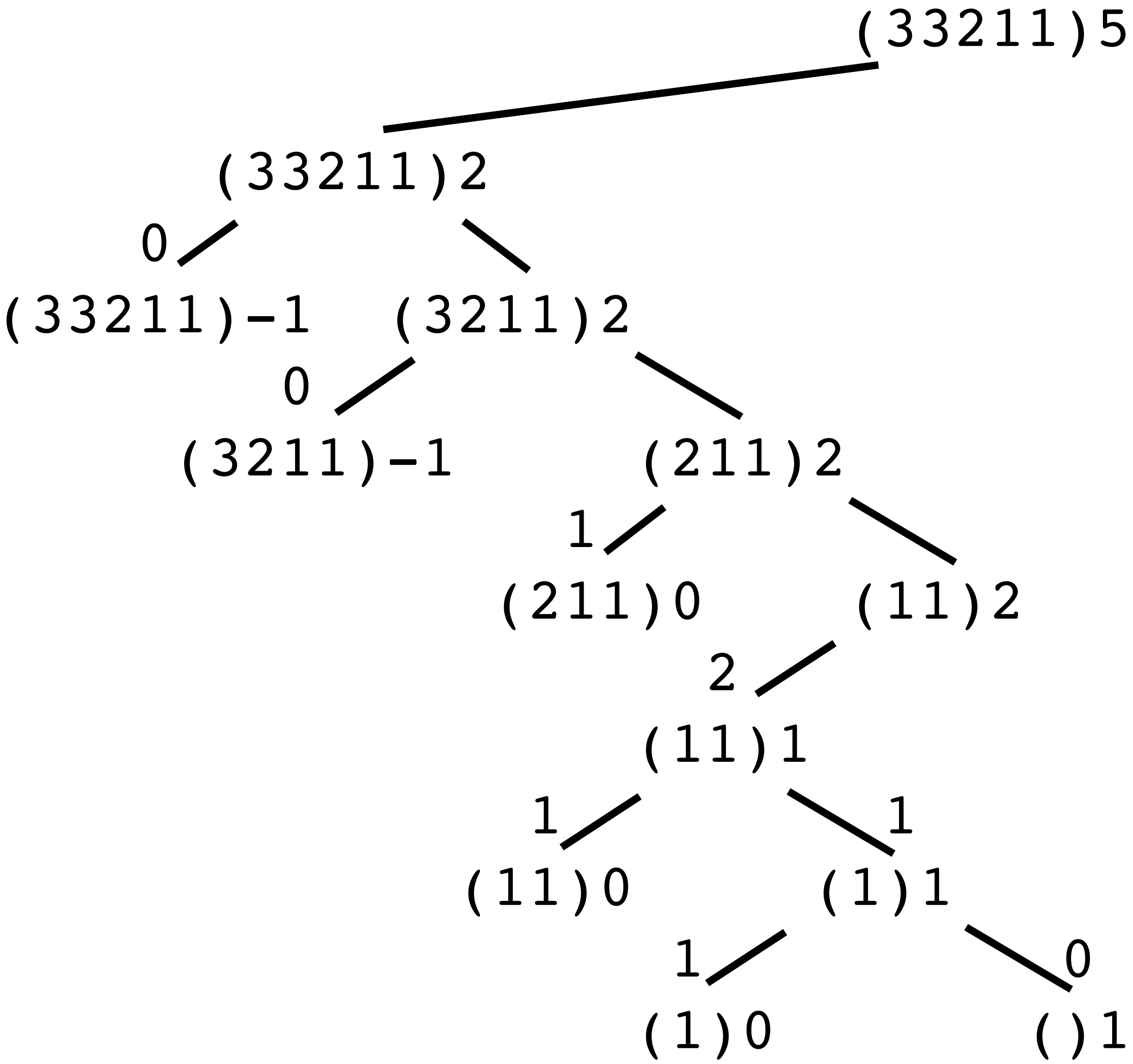
(11)1

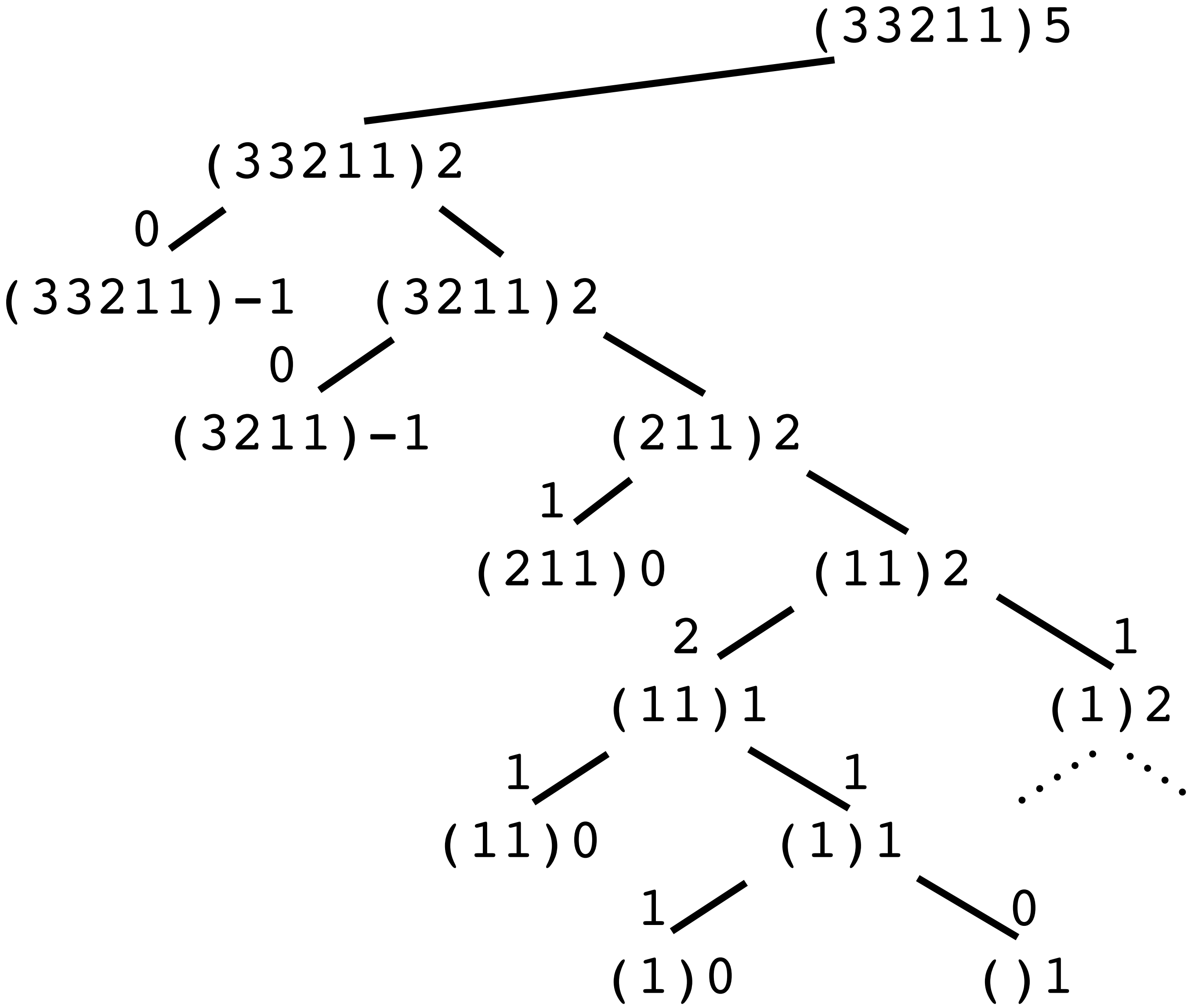
1 1

(11)0 (1)1

1 0

(1)0 ()1


(33211)5
(33211)2
0
(33211)-1
(3211)2
0
(3211)-1
(211)2
1
(211)0
(11)2
2
(11)1
1
(11)0
1
(1)1
1
(1)0
0
()1



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

(3211)-1

(211)2

1

(211)0

(11)2

2

(11)1

1

(1)2

1

(11)0

1

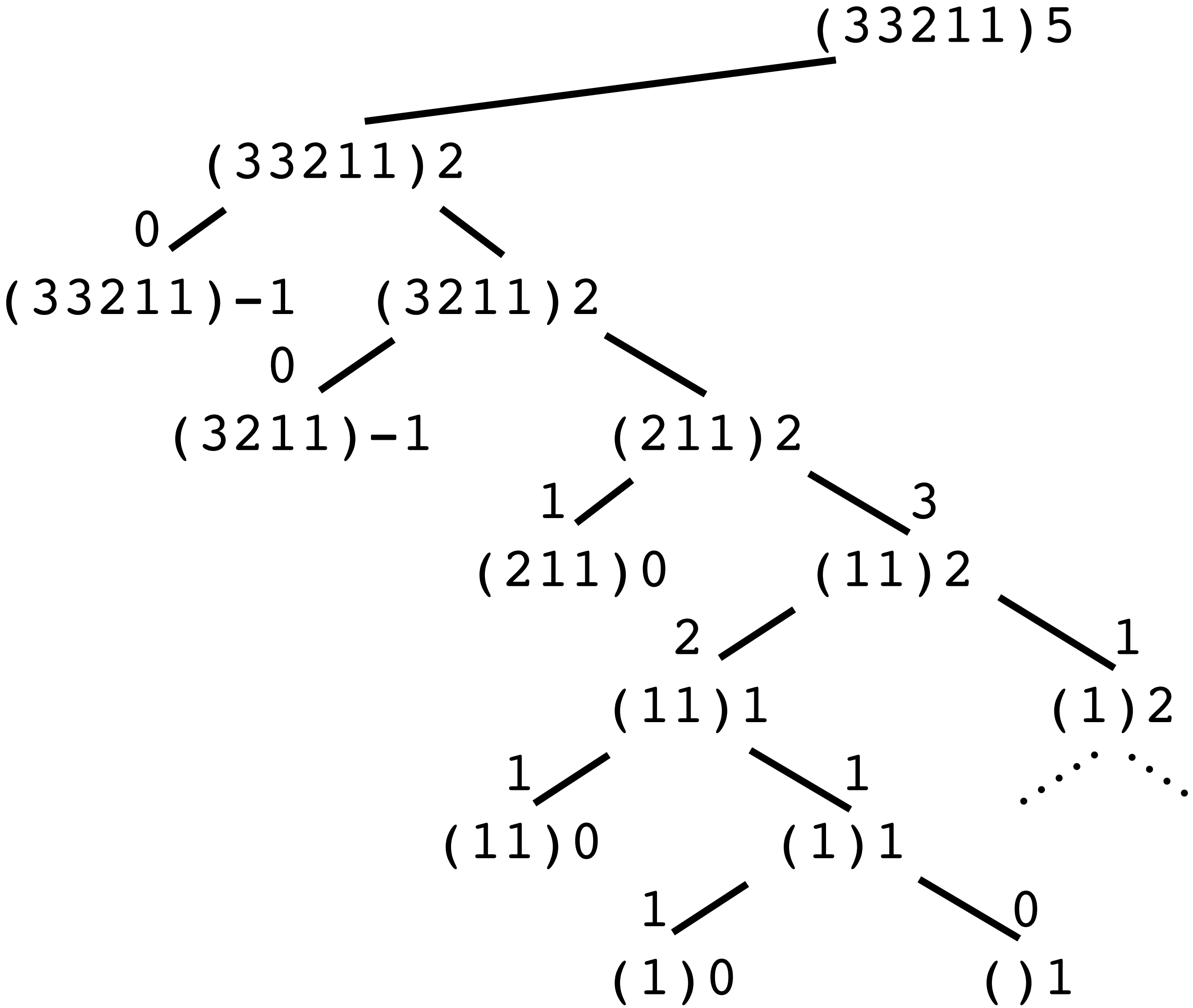
(1)1

1

(1)0

0

()1



(33211)5

(33211)2

0

(33211)-1 (3211)2

0

(3211)-1 (211)2

1 3

(211)0 (11)2

2 1

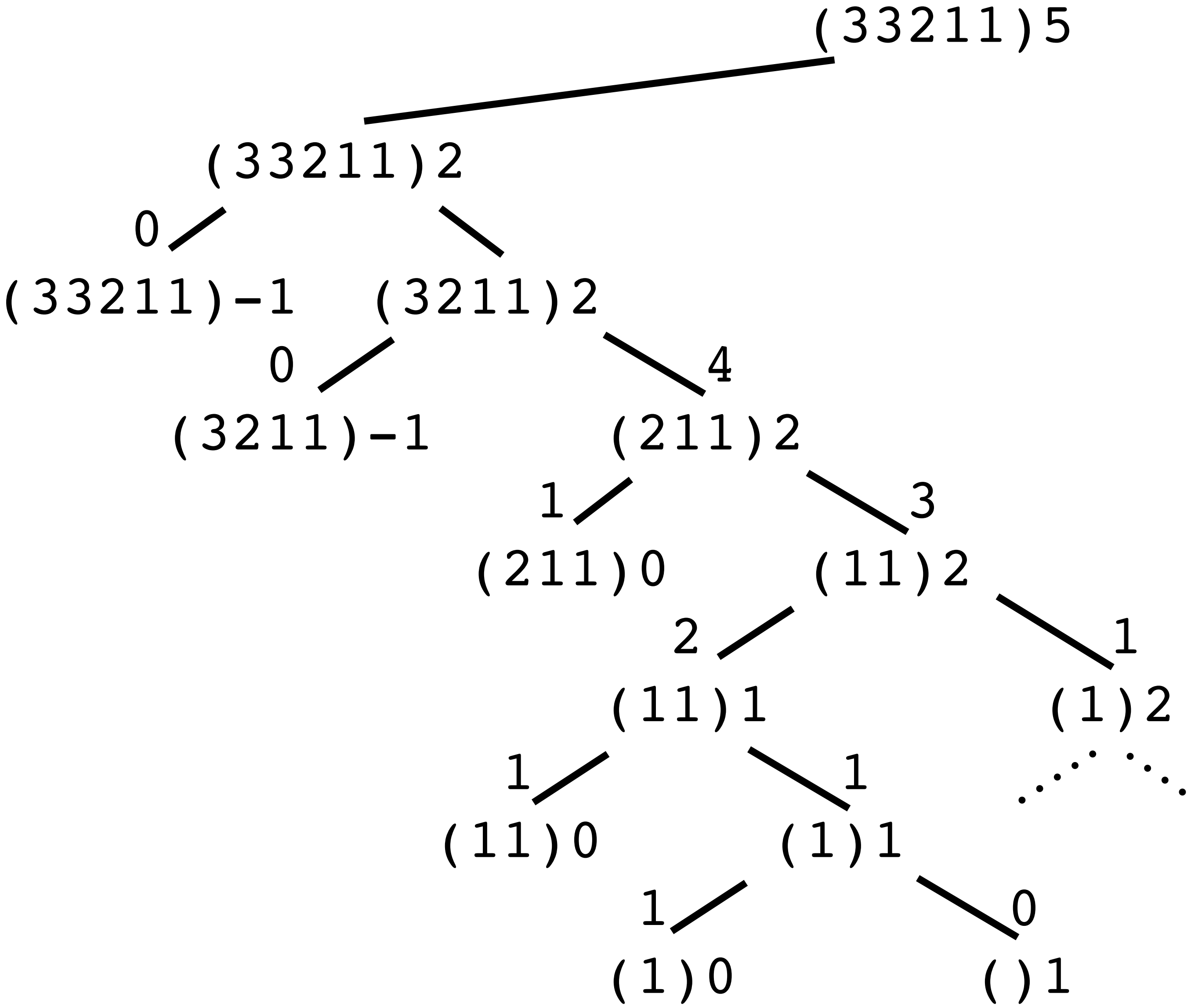
(11)1 (1)2

1 1

(11)0 (1)1

1 0

(1)0 ()1



(33211)5

(33211)2

0

(33211)-1

(3211)2

0

(3211)-1

4

(211)2

1

(211)0

3

(11)2

2

(11)1

1

(1)2

1

(11)0

1

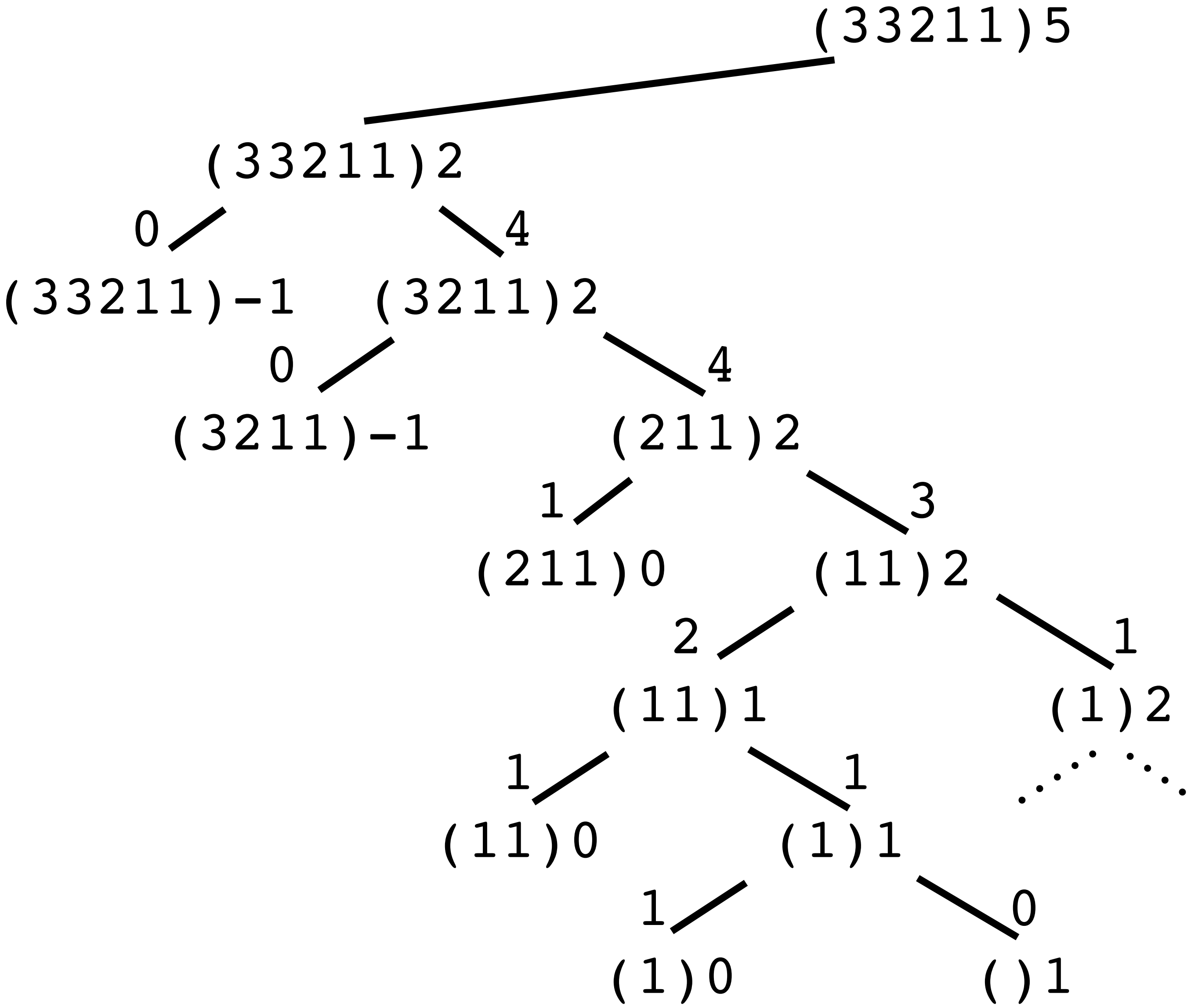
(1)1

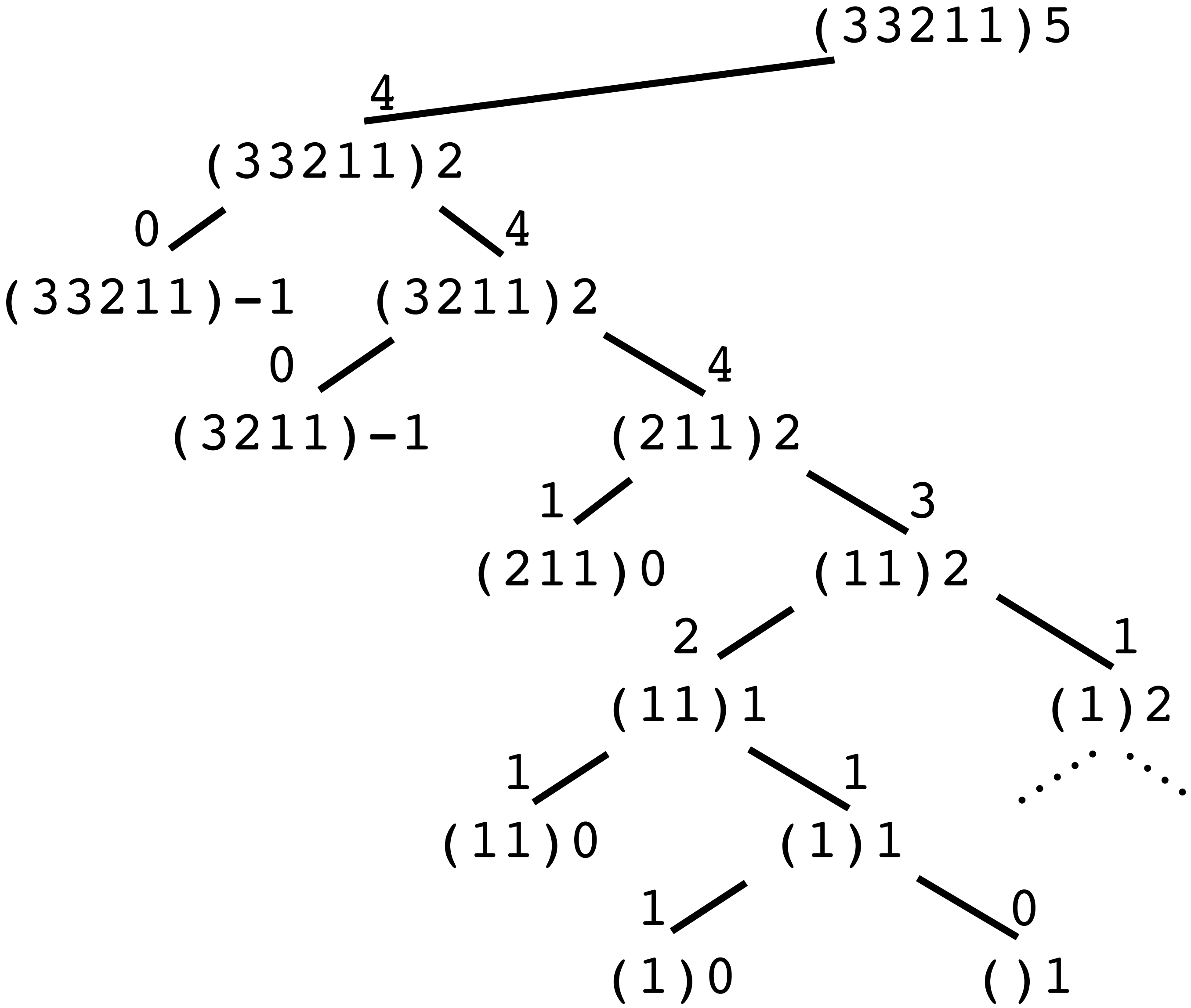
1

(1)0

0

()1


(33211)5
(33211)2
0
4
(33211)-1
(3211)2
0
4
(3211)-1
(211)2
1
3
(211)0
(11)2
2
1
(11)1
(1)2
1
1
(11)0
(1)1
1
0
(1)0
()1


(33211)5

4

(33211)2

0 4

(33211)-1 (3211)2

0 4

(3211)-1 (211)2

1 3

(211)0 (11)2

2 1

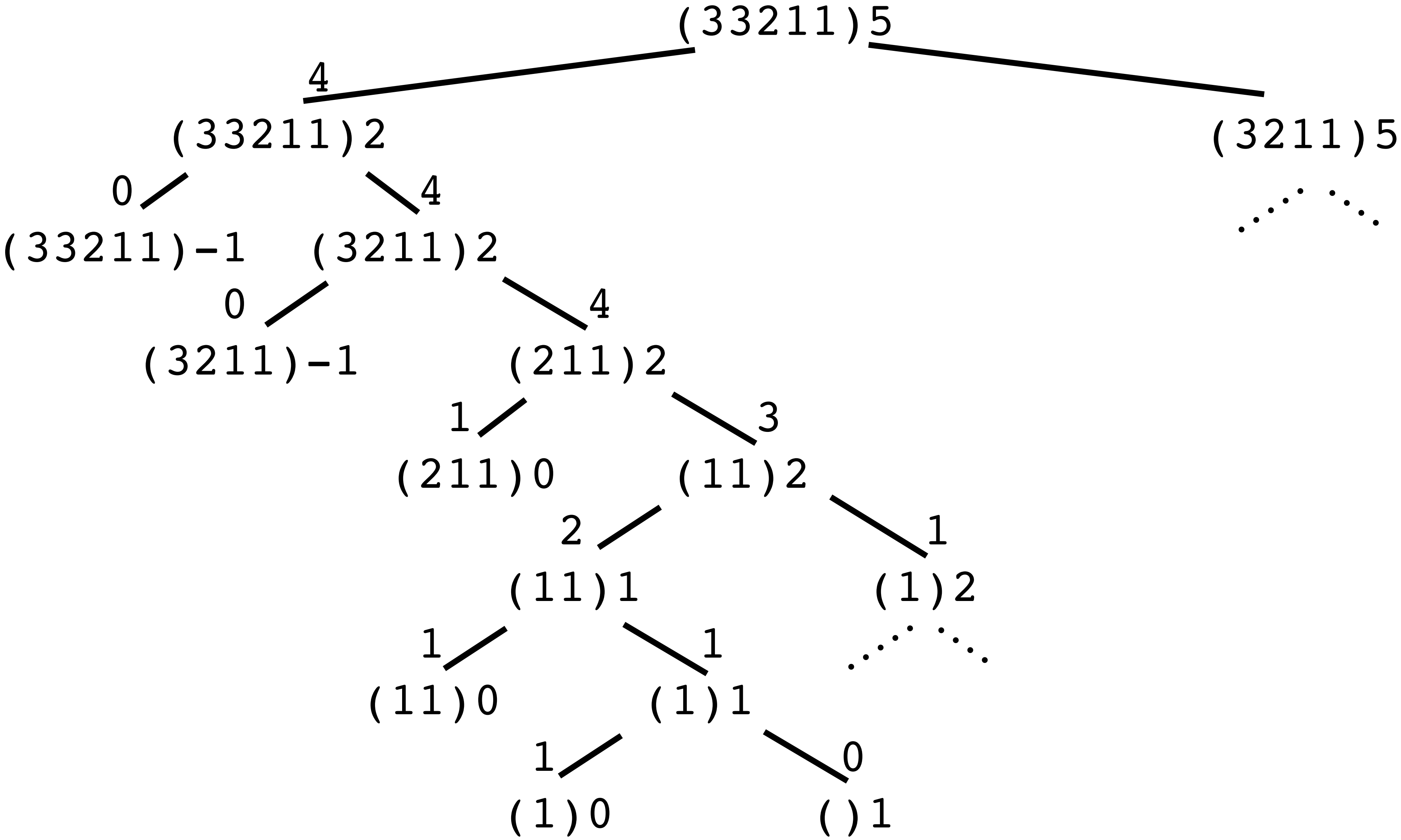
(11)1 (1)2

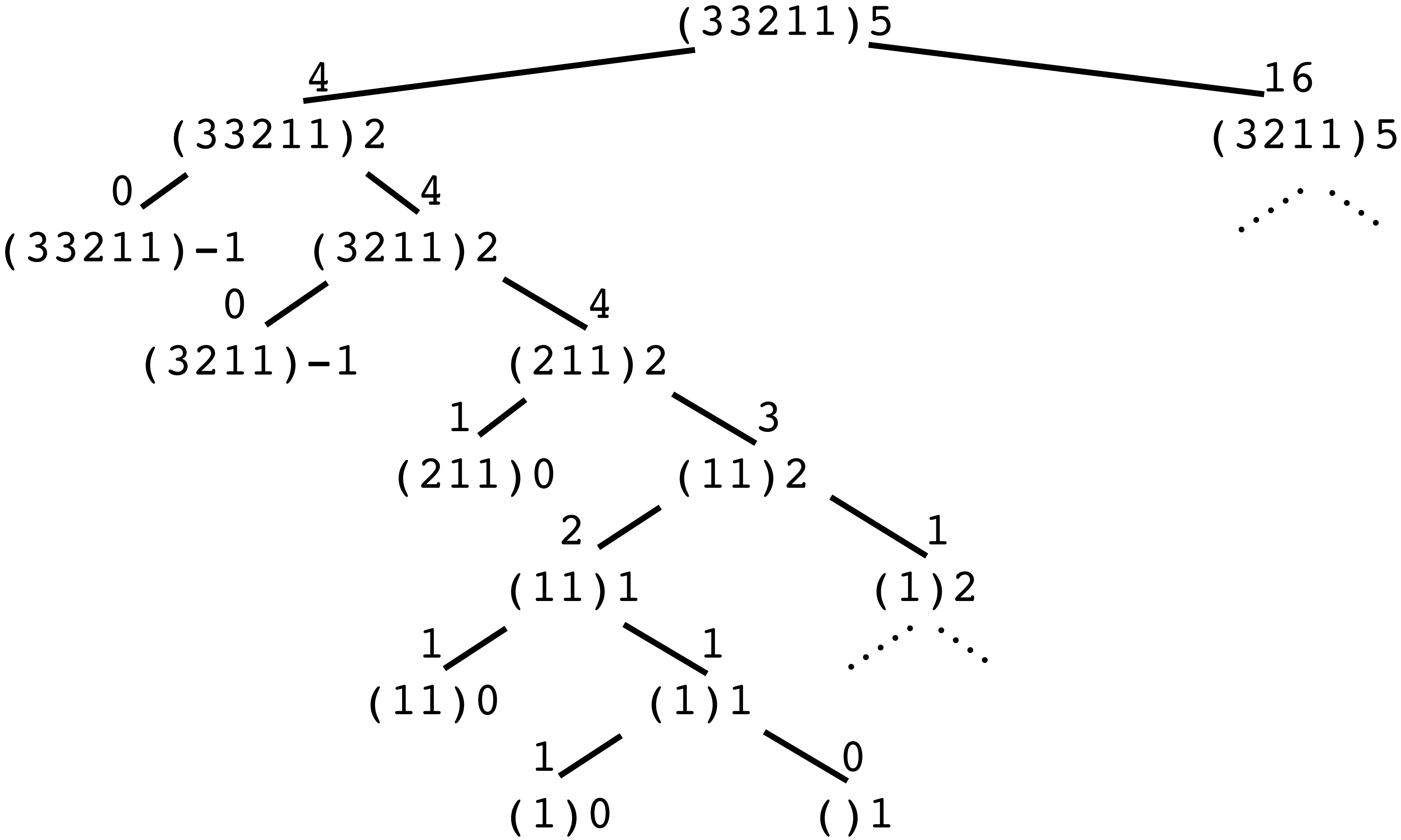
1 1

(11)0 (1)1

1 0

(1)0 ()1


(33211)5
4
(33211)2
(3211)5
0
4
(33211)-1
(3211)2
0
4
(3211)-1
(211)2
1
3
(211)0
(11)2
2
1
(11)1
(1)2
1
1
(11)0
(1)1
1
0
(1)0
()1


(33211)5
4
16
(33211)2
(3211)5
0
4
(33211)-1
(3211)2
0
4
(3211)-1
(211)2
1
3
(211)0
(11)2
2
1
(11)1
(1)2
1
1
(11)0
(1)1
1
0
(1)0
()1

CoSc 450: Programming Paradigms

```
(count-combos ‘(3 3 2 1 1) 5)
```

```
(define count-combos
  (lambda (prize-list amount)
```

```
(count-combos ‘(3 3 2 1 1) 5)
```

```
(define count-combos
  (lambda (prize-list amount)
    .
    .
    .
    (+
```

```
(count-combos ‘(3 3 2 1 1) 5)
```

```
(define count-combos
  (lambda (prize-list amount)
    .
    .
    .
    (+ (count-combos prize-list (- amount (car prize-list)))
```

```
(count-combos ‘(3 3 2 1 1) 5)
```

```
(define count-combos
  (lambda (prize-list amount)
    .
    .
    .
    (+ (count-combos prize-list (- amount (car prize-list)))
       (count-combos (cdr prize-list) amount))))
```