

# Contents

<b>1</b>	<b>Classes</b>	<b>2</b>
1.1	poly.hensel – Hensel lift . . . . .	2
1.1.1	HenselLiftPair – Hensel lift for a pair . . . . .	3
1.1.1.1	lift – lift one step . . . . .	4
1.1.1.2	lift_factors – lift <b>a1</b> and <b>a2</b> . . . . .	4
1.1.1.3	lift_ladder – lift <b>u1</b> and <b>u2</b> . . . . .	4
1.1.2	HenselLiftMulti – Hensel lift for multiple polynomials . . . . .	4
1.1.2.1	lift – lift one step . . . . .	6
1.1.2.2	lift_factors – lift factors . . . . .	6
1.1.2.3	lift_ladder – lift <b>u1</b> and <b>u2</b> . . . . .	6
1.1.3	HenselLiftSimultaneously . . . . .	7
1.1.3.1	lift – lift one step . . . . .	8
1.1.3.2	first_lift – the first step . . . . .	8
1.1.3.3	general_lift – next step . . . . .	8
1.1.4	lift_upto – main function . . . . .	8

# Chapter 1

## Classes

### 1.1 poly.hensel – Hensel lift

- **Classes**
  - †**HenselLiftPair**
  - †**HenselLiftMulti**
  - †**HenselLiftSimultaneously**
- **Functions**
  - **lift\_upto**

In this module document, *polynomial* means integer polynomial.

### 1.1.1 HenselLiftPair – Hensel lift for a pair

#### Initialize (Constructor)

```
HenselLiftPair(f: polynomial, a1: polynomial, a2: polynomial, u1: polynomial, u2: polynomial, p: integer, q: integer=p)  
→ HenselLiftPair
```

This object keeps integer polynomial pair which will be lifted by Hensel's lemma.

The argument should satisfy the following preconditions:

- f, a1 and a2 are monic
- $f == a1 * a2 \pmod{q}$
- $a1 * u1 + a2 * u2 == 1 \pmod{p}$
- p divides q and both are positive

```
from_factors(f: polynomial, a1: polynomial, a2: polynomial, p: integer)  
→ HenselLiftPair
```

This is a class method to create and return an instance of `HenselLiftPair`. You do not have to precompute u1 and u2 for the default constructor; they will be prepared for you from other arguments.

The argument should satisfy the following preconditions:

- f, a1 and a2 are monic
- $f == a1 * a2 \pmod{p}$
- p is prime

#### Attribute

**point** :  
factors a1 and a2 as a list.

## Methods

### 1.1.1.1 lift – lift one step

**lift(self) →**

Lift polynomials by so-called the quadratic method.

### 1.1.1.2 lift\_factors – lift a1 and a2

**lift\_factors(self) →**

Update factors by lifted integer coefficient polynomials  $A_i$ 's:

- $f == A_1 * A_2 \pmod{p * q}$
- $A_i == a_i \pmod{q} \ (i = 1, 2)$

Moreover,  $q$  is updated to  $p * q$ .

†The preconditions which should be automatically satisfied:

- $f == a_1 * a_2 \pmod{q}$
- $a_1 * u_1 + a_2 * u_2 == 1 \pmod{p}$
- $p$  divides  $q$

### 1.1.1.3 lift\_ladder – lift u1 and u2

**lift\_ladder(self) →**

Update  $u_1$  and  $u_2$  with  $U_1$  and  $U_2$ :

- $a_1 * U_1 + a_2 * U_2 == 1 \pmod{p**2}$
- $U_i == u_i \pmod{p} \ (i = 1, 2)$

Then, update  $p$  to  $p**2$ .

†The preconditions which should be automatically satisfied:

- $a_1 * u_1 + a_2 * u_2 == 1 \pmod{p}$

## 1.1.2 HenselLiftMulti – Hensel lift for multiple polynomials

### Initialize (Constructor)

**HenselLiftMulti(f: *polynomial*, factors: *list*, ladder: *tuple*, p: *integer*, q: *integer*=p) → *HenselLiftMulti***

This object keeps integer polynomial factors which will be lifted by Hensel's lemma. If the number of factors is just two, then you should use **HenselLiftPair**.

**factors** is a list of polynomials; we refer those polynomials as **a1**, **a2**, ... **ladder** is a tuple of two lists **sis** and **tis**, both lists consist polynomials. We refer polynomials in **sis** as **s1**, **s2**, ..., and those in **tis** as **t1**, **t2**, ... Moreover, we define **bi** as the product of **aj**'s for  $i < j$ . The argument should satisfy the following preconditions:

- **f** and all of **factors** are monic
- $f == a_1 * \dots * a_r \pmod{q}$
- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, 2, \dots, r)$
- **p** divides **q** and both are positive

```
from _factors(f: polynomial, factors: list, p: integer)
    → HenselLiftMulti
```

This is a class method to create and return an instance of **HenselLiftMulti**. You do not have to precompute **ladder** for the default constructor; they will be prepared for you from other arguments.

The argument should satisfy the following preconditions:

- **f** and all of **factors** are monic
- $f == a_1 * \dots * a_r \pmod{q}$
- **p** is prime

## Attribute

**point** :  
factors **ais** as a list.

## Methods

### 1.1.2.1 lift – lift one step

**lift(self) →**

Lift polynomials by so-called the quadratic method.

### 1.1.2.2 lift\_factors – lift factors

**lift\_factors(self) →**

Update factors by lifted integer coefficient polynomials  $A_i$ s:

- $f == A_1 \dots A_r \pmod{p * q}$
- $A_i == a_i \pmod{q} \ (i = 1, \dots, r)$

Moreover,  $q$  is updated to  $p * q$ .

†The preconditions which should be automatically satisfied:

- $f == a_1 \dots a_r \pmod{q}$
- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, \dots, r)$
- $p$  divides  $q$

### 1.1.2.3 lift\_ladder – lift $u_1$ and $u_2$

**lift\_ladder(self) →**

Update  $s_i$ s and  $t_i$ s with  $S_i$ s and  $T_i$ s:

- $a_1 * S_i + b_i * T_i == 1 \pmod{p**2}$
- $S_i == s_i \pmod{p} \ (i = 1, \dots, r)$
- $T_i == t_i \pmod{p} \ (i = 1, \dots, r)$

Then, update  $p$  to  $p**2$ .

†The preconditions which should be automatically satisfied:

- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, \dots, r)$

### 1.1.3 HenselLiftSimultaneously

The method explained in [?].

†Keep these invariants:

- $a_i, p_i$  and  $g_i$  are monic
- $f == g_1 * \dots * g_r \pmod{p}$
- $f == d_0 + d_1 p + d_2 p^2 + \dots + d_k p^k$
- $h_i == g_{(i+1)} * \dots * g_r$
- $1 == g_i s_i + h_i t_i \pmod{p} \ (i = 1, \dots, r)$
- $\deg(s_i) < \deg(h_i), \deg(t_i) < \deg(g_i) \ (i = 1, \dots, r)$
- $p$  divides  $q$
- $f == l_1 * \dots * l_r \pmod{q/p}$
- $f == a_1 * \dots * a_r \pmod{q}$
- $u_i == a_i y_i + b_i z_i \pmod{p} \ (i = 1, \dots, r)$

#### Initialize (Constructor)

```
HenselLiftSimultaneously(target: polynomial, factors: list, cofactors:
list, bases: list, p: integer)
    → HenselLiftSimultaneously
```

This object keeps integer polynomial factors which will be lifted by Hensel's lemma.

```
f = target, gi in factors, his in cofactors and sis and tis are in bases.
from _factors(target: polynomial, factors: list, p: integer, ubound: in-
teger=sys.maxint)
    → HenselLiftSimultaneously
```

This is a class method to create and return an instance of `HenselLiftSimultaneously`, whose factors are lifted by `HenselLiftMulti` upto `ubound` if it is smaller than `sys.maxint`, or upto `sys.maxint` otherwise. You do not have to precompute auxiliary polynomials for the default constructor; they will be prepared for you from other arguments.

```
f = target, gis in factors.
```

## Methods

### 1.1.3.1 lift – lift one step

**lift(self) →**

The lift. You should call this method only.

### 1.1.3.2 first\_lift – the first step

**first\_lift(self) →**

Start lifting.

`f == l1*l2*...*lr (mod p**2)`

Initialize `dis`, `uis`, `vis` and `zis`. Update `ais`, `bis`. Then, update `q` with `p**2`.

### 1.1.3.3 general\_lift – next step

**general\_lift(self) →**

Continue lifting.

`f == a1*a2*...*ar (mod p*q)`

Initialize `ais`, `ubis`, `vis` and `zis`. Then, update `q` with `p*q`.

### 1.1.4 lift\_upto – main function

**lift\_upto(self, target: *polynomial*, factors: *list*, p: *integer*, bound: *integer*)**  
**→ *tuple***

Hensel lift `factors` mod `p` of `target` upto `bound` and return `factors` mod `q` and the `q` itself.

These preconditions should be satisfied:

- `target` is monic.
- `target == product(factors) mod p`

The result (`factors`, `q`) satisfies the following postconditions:

- there exist  $k$  s.t. `q == p**k >= bound` and
- `target == product(factors) mod q`