Contents

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 (mod q)`
- `a1*u1 + a2*u2 == 1 (mod 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 (mod p)`
- `p` is prime

Attributes

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 `Ai`'s:

- `f == A1 * A2 (mod p * q)`
- `Ai == ai (mod q)` $(i = 1, 2)$

Moreover, `q` is updated to `p * q`.
†The preconditions which should be automatically satisfied:

- `f == a1*a2 (mod q)`
- `a1*u1 + a2*u2 == 1 (mod p)`
- `p` divides `q`

1.1.1.3 lift_ladder – lift `u1` and `u2`

lift_ladder(self) →

Update `u1` and `u2` with `U1` and `U2`:

- `a1*U1 + a2*U2 == 1 (mod p**2)`
- `Ui == ui (mod p)` $(i = 1, 2)$

Then, update `p` to `p**2`.
†The preconditions which should be automatically satisfied:

- `a1*u1 + a2*u2 == 1 (mod 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 == a1*...*ar (mod q)`
- `ai*si + bi*ti == 1 (mod p)` $(i = 1, 2, \ldots, r)$
- `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 == a1*...*ar (mod q)`
- `p` is prime

Attributes

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 `Ais`:

- `f == A1*...*Ar (mod p * q)`
- `Ai == ai (mod q)` $(i = 1, \dots, r)$

Moreover, `q` is updated to `p * q`.
†The preconditions which should be automatically satisfied:

- `f == a1*...*ar (mod q)`
- `ai*si + bi*ti == 1 (mod p)` $(i = 1, \dots, r)$
- `p` divides `q`

1.1.2.3 lift_ladder – lift `u1` and `u2`

lift_ladder(self) →

Update `sis` and `tis` with `Sis` and `Tis`:

- `a1*Si + bi*Ti == 1 (mod p**2)`
- `Si == si (mod p)` $(i = 1, \dots, r)$
- `Ti == ti (mod p)` $(i = 1, \dots, r)$

Then, update `p` to `p**2`.
†The preconditions which should be automatically satisfied:

- `ai*si + bi*ti == 1 (mod p)` $(i = 1, \dots, r)$

1.1.3 HenselLiftSimultaneously

The method explained in [1].

†Keep these invariants:

- `ais`, `pi` and `gis` are monic
- `f == g1*...*gr (mod p)`
- `f == d0 + d1*p + d2*p**2 +...+ dk*p**k`
- `hi == g(i+1)*...*gr`
- `1 == gi*si + hi*ti (mod p)` $(i = 1, \ldots, r)$
- deg(`si`) $<$ deg(`hi`), deg(`ti`) $<$ deg(`gi`) $(i = 1, \ldots, r)$
- `p` divides `q`
- `f == l1*...*lr (mod q/p)`
- `f == a1*...*ar (mod q)`
- `ui == ai*yi + bi*zi (mod p)` $(i = 1, \ldots, r)$

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`: ***integer***=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`, `yis` 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`, `yis` 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`

Bibliography

[1] G. E. Collins and M. J. Encarnación. Improved techniques for factoring univariate polynomials. *Journal of Symbolic Computation*, Vol. 21, pp. 313–327, 1996.