Hensel lifting: Newton iteration and relaxed algorithms*

by Romain Lebreton

SCG Lab Meeting

October 11th, 2013

*. This document has been written using the GNU $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ text editor (see www.texmacs.org).

Setting

- We want to solve the polynomial equation over $\mathbb{Q}[[T]]$:

$$P(Y) = Y^2 - Y + T \in (\mathbb{Q}[[T]])[Y]$$

- Consider the *regular* modular root $y_0 = 0$ of $P(Y)$, *i.e.*

$$P(y_0) = 0 \bmod T \text{ and } P'(y_0) \neq 0 \bmod T.$$

- Since $P'(y_0) \neq 0 \bmod T$, *Hensel's Lemma* ensure that $\exists! y \in \mathbb{Q}[[T]]$ such that

$$P(y) = 0, \quad y = y_0 \bmod T.$$

Example.

The lifted root y from y_0 is

$$y = \frac{1 - \sqrt{1 - 4T}}{2} = T + T^2 + 2T^3 + 5T^4 + \mathcal{O}(T^5).$$

Our goal

Definition.

Hensel lifting is the process to compute y from y_0 and P.

This talk.

We will show how to perform Hensel lifting using

1. Newton's iteration
2. Relaxed algorithms

Newton's iteration

Input:

- A polynomial equation $P \in (\mathbb{Q}[[T]])[Y]$
- A regular modular root $y_0 \in \mathbb{Q}$ of P
- A precision $2^N \in \mathbb{N}$

Output:

- The unique lifted root $y \in \mathbb{Q}[[T]]$ at precision 2^N, *i.e.* modulo T^{2^N}

Algorithm

$r = y_0$
for i **from** 1 **to** N
$\quad r = \left(r - \frac{P(r)}{P'(r)} \right) \bmod T^{2^i}$
return r

Newton's iteration

Input:

- A polynomial equation $P \in (\mathbb{Q}[[T]])[Y]$
- A regular modular root $y_0 \in \mathbb{Q}$ of P
- A precision $2^N \in \mathbb{N}$

Output:

- The unique lifted root $y \in \mathbb{Q}[[T]]$ at precision 2^N, *i.e.* modulo T^{2^N}

Algorithm
$r = y_0$ **for** i **from** 1 **to** N $\quad r = \left(r - \frac{P(r)}{P'(r)}\right) \bmod T^{2^i}$ **return** r

Step 0: $\quad r = 0 + \mathcal{O}(T)$

Step 1: $\quad r = 0 + T + \mathcal{O}(T^2)$

Step 2: $\quad r = 0 + T + T^2 + 2\,T^3 + \mathcal{O}(T^4)$

Newton's iteration

Input:

- A polynomial equation $P \in (\mathbb{Q}[[T]])[Y]$
- A regular modular root $y_0 \in \mathbb{Q}$ of P
- A precision $2^N \in \mathbb{N}$

Output:

- The unique lifted root $y \in \mathbb{Q}[[T]]$ at precision 2^N, *i.e.* modulo T^{2^N}

Algorithm

$r = y_0$
for i **from** 1 **to** N
$\quad r = \left(r - \frac{P(r)}{P'(r)}\right) \bmod T^{2^i}$
return r

Remark.

$P'(r)$ is invertible since $P'(r) = P'(y_0) \bmod T$, which is non-zero.

Relaxed algorithms

Definition. (*on-line* or *relaxed* algorithm) [Hennie '66]

$a = \sum_{i \geqslant 0} a_i x^i$	a_0	a_1	a_2	$\cdots$
$b = \sum_{i \geqslant 0} b_i x^i$	b_0	b_1	b_2	$\cdots$
$\downarrow f$				
$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$	c_0			$\cdots$

: reading allowed

Relaxed algorithms

Definition. (*on-line* or *relaxed* algorithm) [Hennie '66]

$a = \sum_{i \geqslant 0} a_i x^i$	a_0	a_1	a_2	$\cdots$
$b = \sum_{i \geqslant 0} b_i x^i$	b_0	b_1	b_2	$\cdots$
$\downarrow f$				
$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$	c_0	c_1		$\cdots$

: reading allowed

Relaxed algorithms

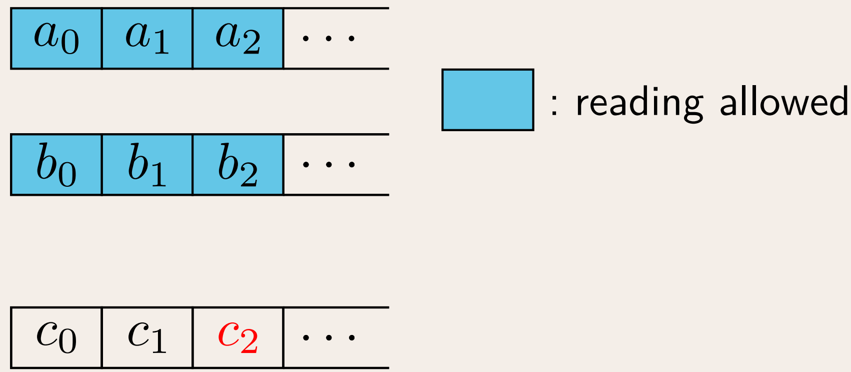
Definition. (*on-line* or *relaxed* algorithm) [Hennie '66]

$$a = \sum_{i \geqslant 0} a_i x^i$$

$$b = \sum_{i \geqslant 0} b_i x^i$$

$$\Big\downarrow f$$

$$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$$



Relaxed algorithms

Definition. (*on-line* or *relaxed* algorithm) [Hennie '66]

$$a = \sum_{i\geqslant 0} a_i x^i \qquad \begin{array}{|c|c|c|c}\hline a_0 & a_1 & a_2 & \cdots \\ \hline\end{array}$$

$$b = \sum_{i\geqslant 0} b_i x^i \qquad \begin{array}{|c|c|c|c}\hline b_0 & b_1 & b_2 & \cdots \\ \hline\end{array}$$

$$\downarrow f$$

$$c = f(a,b) = \sum_{i\geqslant 0} c_i x^i \qquad \begin{array}{|c|c|c|c}\hline c_0 & c_1 & c_2 & \cdots \\ \hline\end{array}$$

: reading allowed

Off-line *or* zealous *algorithm :* *condition not met.*

Example of relaxed algorithms

Relaxed Addition:

Algorithm `Add`

Input: $a, b \in \mathbb{Q}[[T]]$ and $n \in \mathbb{N}$
Output: $c \in \mathbb{Q}[[T]]$ such that $c = (a + b) \bmod T^n$

1. $c = 0$
2. **for** i from 0 to $n - 1$
   a. $c = \texttt{AddStep}(a, b, c, i)$
3. **return** c

Algorithm `AddStep`

Input: $a, b, c \in \mathbb{Q}[[T]]$ and $i \in \mathbb{N}$
Output: $c \in \mathbb{Q}[[T]]$

1. $c = c + (a_i + b_i)\, T^i$
2. **return** c

Remarks.

1. Computations at step i complete those of previous steps to get a result modulo T^i.

2. This addition algorithm is *online*:

$\rightarrow$ it outputs c_i using a_i and b_i.

Example of relaxed algorithms

Naive Relaxed Multiplication:

Algorithm `NaiveMulStep`

Input: $a, b, c \in \mathbb{Q}[[T]]$ and $i \in \mathbb{N}$
Output: $c \in \mathbb{Q}[[T]]$

1. **for** j from 0 to i
   a. $c = c + a_j b_{i-j} T^i$
2. **return** c

Algorithm `NaiveMul`

Input: $a, b \in \mathbb{Q}[[T]]$ and $n \in \mathbb{N}$
Output: $c \in \mathbb{Q}[[T]]$ such that $c = (a + b) \bmod T^n$

1. $c = 0$
2. **for** i from 0 to $n - 1$
   a. $c = \texttt{NaiveMulStep}(a, b, c, i)$
3. **return** c

Remarks.

1. This multiplication algorithm is *online*:

   $\rightarrow$ it outputs c_i using $a_0, \ldots, a_i$ and $b_0, \ldots, b_i$.

2. Its complexity is quadratic !

Fast relaxed multiplications

Problem.

Karatsuba and FFT algorithms are offline.

Challenge.

Find a *quasi-optimal on-line* multiplication algorithm.

Theorem. [Fischer, Stockmeyer '74], [Schröder '97], [van der Hoeven '97]
[Berthomieu, van der Hoeven, Lecerf '11], [L., Schost '13]

Let $\mathsf{M}(N)$ *be the cost of* $a \times b$ *in* $\mathbb{Q}[[T]]$ *at precision* N *by an off-line algorithm.*
Let $\mathsf{R}(N)$ *be the cost of* $a \times b$ *in* $\mathbb{Q}[[T]]$ *at precision* N *by an on-line algorithm.*

Then

$$\mathsf{R}(N) = \mathcal{O}(\mathsf{M}(N) \log N) = \tilde{\mathcal{O}}(N).$$

Fast relaxed multiplications

Problem.

Karatsuba and FFT algorithms are offline.

Challenge.

Find a *quasi-optimal on-line* multiplication algorithm.

Theorem. [Fischer, Stockmeyer '74], [Schröder '97], [van der Hoeven '97] [Berthomieu, van der Hoeven, Lecerf '11], [L., Schost '13]

Let $\mathsf{M}(N)$ *be the cost of* $a \times b$ *in* $\mathbb{Q}[[T]]$ *at precision* N *by an off-line algorithm.*
Let $\mathsf{R}(N)$ *be the cost of* $a \times b$ *in* $\mathbb{Q}[[T]]$ *at precision* N *by an on-line algorithm.*

Then

$$\mathsf{R}(N) = \mathcal{O}(\mathsf{M}(N) \log N) = \tilde{\mathcal{O}}(N).$$

Theorem. [van der Hoeven '07, '12]

$$\mathsf{R}(N) = \mathsf{M}(N) \log (N)^{o(1)}.$$

Seems not to be used yet in practice.

Recursive power series

Definition. *A power series $y \in \mathbb{Q}[[T]]$ is recursive if there exists Φ such that*

- $y = \Phi(y)$
- $\Phi(y)_n$ *only depends on* $y_0, \ldots, y_{n-1}$

$\rightsquigarrow$ It is possible to compute y from Φ and y_0. But how fast?

Example.

The solution y of $P(Y) = Y^2 - Y + T$ s.t. $y_0 = 0$ is recursive (the other is too).

Proof. Let $\Phi(Y) = Y^2 + T$ then $y = \Phi(y)$.

Moreover,

$$\begin{aligned}\Phi(y)_n = (y^2)_n + (T)_n &= y_0\, y_n + y_1\, y_{n-1} + \cdots + y_n\, y_0 + \delta_{1,n} \\ &= y_1\, y_{n-1} + \cdots + y_{n-1}\, y_1 + \delta_{1,n}\end{aligned}$$

since $y_0 = 0$. It does not depend on y_n. □

Shifted Algorithms

Definition of *shifted algorithms*.

$$a = \sum_{i \geqslant 0} a_i x^i$$

a_0	a_1	a_2	$\cdots$

: reading allowed

$$\Big\downarrow f$$

$$c = f(a) = \sum_{i \geqslant 0} c_i x^i$$

c_0			$\cdots$

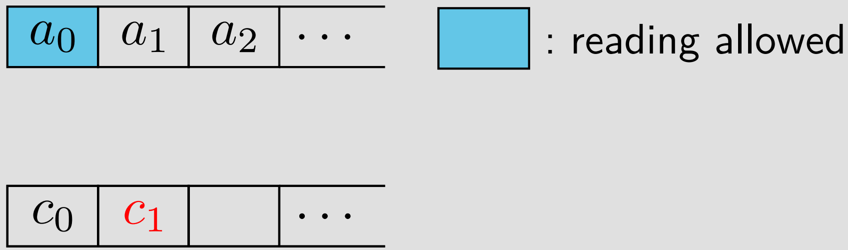
Shifted Algorithms

Definition of *shifted algorithms*.

$$a = \sum_{i \geqslant 0} a_i x^i$$

$$\downarrow f$$

$$c = f(a) = \sum_{i \geqslant 0} c_i x^i$$



Shifted Algorithms

Definition of *shifted algorithms*.

$$a = \sum_{i \geqslant 0} a_i x^i$$

a_0	a_1	a_2	$\cdots$

: reading allowed

$$\Big\downarrow f$$

$$c = f(a) = \sum_{i \geqslant 0} c_i x^i$$

c_0	c_1	c_2	$\cdots$

Shifted Algorithms

Definition of *shifted algorithms*.

$$a = \sum_{i \geqslant 0} a_i x^i \qquad \boxed{a_0}\boxed{a_1}\boxed{a_2}\boxed{\cdots}$$

$$\downarrow f$$

$$c = f(a) = \sum_{i \geqslant 0} c_i x^i \qquad \boxed{c_0}\boxed{c_1}\boxed{c_2}\boxed{\cdots}$$

: reading allowed

Example.

If $s > 0$, then the Algorithm `ShiftStep`: $a \mapsto T^s \times a$ is a shifted algorithm.

Algorithm `ShiftStep`

Input: $a, c \in R_p$, $s \in \mathbb{Z}$ and $i \in \mathbb{N}$

Output: $c \in \mathbb{Q}[[T]]$

1. $c = c + a_{i-s} T^i$
2. **return** c

Example of shifted algorithm

Example.

The online evaluation of $\Psi(Y) = T^2 \times (Y/T)^2 + T$ is shifted w.r.t. Y.

	$(\Psi(Y))_i$	
requires	$(T^2 \times ((Y/T)^2))_i$	
requires	$((Y/T)^2)_{i-2}$	
requires	$(Y/T)_0, \ldots, (Y/T)_{i-2}$	using online multiplication
requires	$Y_0, \ldots, Y_{i-1}$	

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{Q}[[T]]$ such that $y = \Phi(y)$. Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

Proof. $y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$$\begin{array}{c} y = \sum_{i \geqslant 0} y_i x^i \\ \Big\downarrow \Phi \\ \Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \end{array}$$

y_0	?	?	$\cdots$
φ_0	φ_1		$\cdots$

: reading allowed

$\square$

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

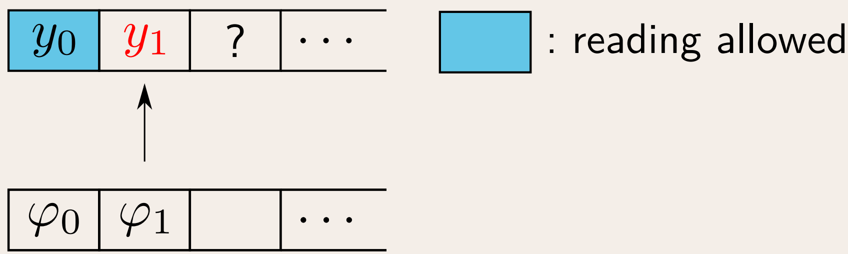
Let $y \in \mathbb{Q}[[T]]$ such that $y = \Phi(y)$. Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

Proof. $y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$$y = \sum_{i \geqslant 0} y_i x^i$$

$$\big\downarrow \Phi$$

$$\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i$$



$\square$

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{Q}[[T]]$ such that $y = \Phi(y)$. Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

Proof. $y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$$\begin{array}{c} y = \sum_{i \geqslant 0} y_i x^i \\ \Big\downarrow \Phi \\ \Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \end{array}$$

y_0	y_1	?	$\cdots$
φ_0	φ_1	φ_2	$\cdots$

: reading allowed

$\square$

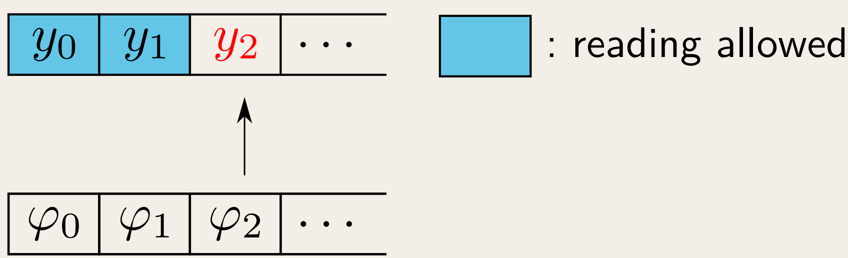
Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{Q}[[T]]$ such that $y = \Phi(y)$. Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

Proof. $y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$$\begin{array}{c} y = \sum_{i \geqslant 0} y_i x^i \\ \Big\downarrow \Phi \\ \Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \end{array}$$



$\square$

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0					1st
					2nd
					3rd
					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
					2nd
					3rd
					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$

Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0					2nd
					3rd
					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0/T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
					3rd
					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`
Input: a shifted algorithm $\mathbf{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n
1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
a. Perform the ith step of the evaluation of $\mathbf{\Psi}$ on input $\boldsymbol{a}$
b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
T					3rd
					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$

Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
    a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
    b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
T	1	1	T^2	$T + T^2$	3rd
					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$

Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
    a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
    b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
T	1	1	T^2	$T + T^2$	3rd
$T + T^2$					4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$

Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0/T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
T	1	1	T^2	$T + T^2$	3rd
$T + T^2$	$1 + T$	$1 + 2T$	$T^2 + 2T^3$	$T + T^2 + 2T^3$	4th
					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
T	1	1	T^2	$T + T^2$	3rd
$T + T^2$	$1 + T$	$1 + 2T$	$T^2 + 2T^3$	$T + T^2 + 2T^3$	4th
$T + T^2 + 2T^3$					5th

Relaxed recursive power series

Algorithm `OnlineRecursivePadic`

Input: a shifted algorithm $\boldsymbol{\Psi}$, a modular root $\boldsymbol{y}_0$ and a precision $n \in \mathbb{N}$
Output: the lifted root $\boldsymbol{y}$ at precision n

1. $\boldsymbol{a} = \boldsymbol{y}_0$, $[c_1, \ldots, c_L] = [0, \ldots, 0]$
2. **for** i from 1 to n
   a. Perform the ith step of the evaluation of $\boldsymbol{\Psi}$ on input $\boldsymbol{a}$
   b. Put the output in $\boldsymbol{a}$
3. **return** $\boldsymbol{a}$

Computations of the lifting of the root y of $P(Y) = Y^2 - Y + T$ from $y_0 = 0$:

c_0	$c_1 := c_0 / T$	$c_2 := c_1^2$	$c_3 := T^2 \times c_2$	$c_4 := c_3 + T$	Step
	$(i-2)$th step	$(i-2)$th step	ith step	ith step	ith
0	0	0	0	0	1st
0	0	0	0	T	2nd
T	1	1	T^2	$T + T^2$	3rd
$T + T^2$	$1 + T$	$1 + 2T$	$T^2 + 2T^3$	$T + T^2 + 2T^3$	4th
$T + T^2 + 2T^3$	$1 + T + 2T^2$	$1 + 2T + 5T^2$	$T^2 + 2T^3 + 5T^4$	$T + T^2 + 2T^3 + 5T^4$	5th

Conclusion

Two general paradigms:

Newton operator	Relaxed algorithms
Solve implicit equations $P(y)=0$	Solve recursive equations $y=\Phi(y)$
Faster for higher precision	
	Less on-line multiplications

Implementations:

Relaxed power series (and T-adics) in Mathemagix

Beginning of a `C++` package based on NTL

Thank you for your attention ;-)

Example of shifted algorithm

Example.

The evaluation Algorithm of $\Psi(Y) = T^2 \times ((Y/T)^2) + T$ is shifted.

S.l.p. $\Gamma_1 = (\_/T; 0), \Gamma_2 = (*; 1, 1), \Gamma_3 = (T^2 \times; 2), \Gamma_4 = (+; 3, T)$

Output sequence $[c_0, \ldots, c_4]$ on input $c_0 = y$ is $[y,\ y/T,\ (y/T)^2,\ T^2 \times ((Y/T)^2),\ T^2 \times ((Y/T)^2) + T]$

Algorithm `EvaluationStep`

Input: $a \in \mathbb{Q}[[T]]$, $\left[c_1^{(0)}, \ldots, c_4^{(0)}\right] \in (\mathbb{Q}[[T]])^4$ and $i \in \mathbb{N}$

Output: $[c_1, \ldots, c_4] \in (\mathbb{Q}[[T]])^4$

1. $c_0 = a$, $[c_1, \ldots, c_4] = \left[c_1^{(0)}, \ldots, c_4^{(0)}\right]$
2. Evaluation of Ψ:
   a. $c_1 = \texttt{ShiftStep}(a, c_1, -1, i-2)$
   b. $c_2 = \texttt{MulStep}(c_1, c_1, c_2, i-2)$
   c. $c_3 = \texttt{ShiftStep}(c_2, c_3, 2, i)$
   d. $c_4 = \texttt{AddStep}(c_3, T, c_4, i)$
3. **return** $[c_1, \ldots, c_4]$

`EvaluationStep` increase the precision by one of the evaluation of Ψ on y.