Computational Complexity of real functions

Amaury Pouly

April 28, 2015

Outline

1. Complexity of real functions
   - Introduction
   - Computable Analysis
   - GPAC
   - Analog Church Thesis
2. Toward a Complexity Theory for the GPAC
   - What is the problem ?
   - A complexity class
3. Conclusion

Motivating example

Motivating example

Example (Sine function)

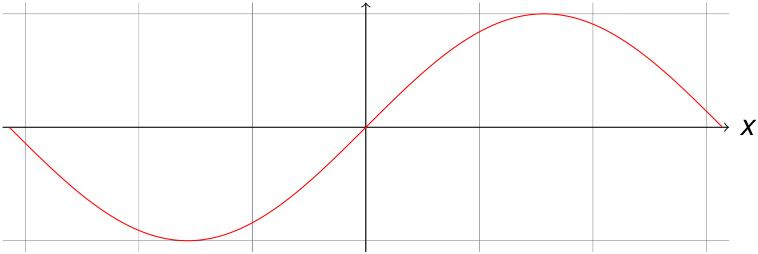
Given $x \in \mathbb{R}$, compute $\sin(x)$.

Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$.

$\Rightarrow$ "clearly sin is computable:"

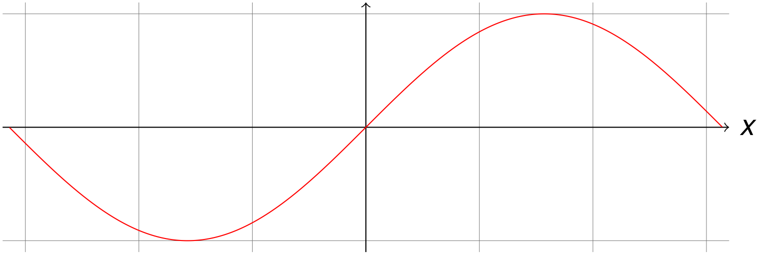


Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$.

$\Rightarrow$ "clearly sin is computable:"



But...

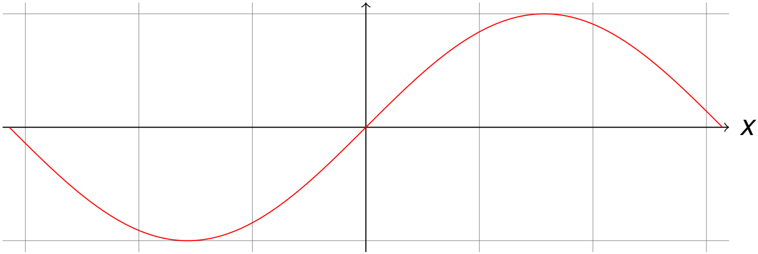
- how do you represent a real number ? (infinite object)

Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$.

$\Rightarrow$ "clearly sin is computable:"



But...

- how do you represent a real number ? (infinite object)
- what is a program working on them ?

A classical approach

Computable analysis

A classical approach

Computable analysis

- a real number is a program:

A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations

A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
- a function is a program transformation:

A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
- a function is a program transformation: it transformes one approximation into another

A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
- a function is a program transformation: it transformes one approximation into another

- Intuition: can draw the graph of a function with arbitrary zoom
- Very analytic, approximation theory
- Can lift Turing complexity to real functions
- Has a nice theory of open sets

Computable real

Computable real

Definition (Computable Real)

A real $r \in \mathbb{R}$ is computable is one can compute an arbitrary close approximation for a given precision:

Computable real

Definition (Computable Real)

A real $r \in \mathbb{R}$ is computable is one can compute an arbitrary close approximation for a given precision:

$$\text{Given } p \in \mathbb{N}, \text{ compute } r_p \text{ s.t. } |r - r_p| \leqslant 2^{-p}$$

Computable real

Definition (Computable Real)

A real $r \in \mathbb{R}$ is computable is one can compute an arbitrary close approximation for a given precision:

$$\text{Given } p \in \mathbb{N}, \text{ compute } r_p \text{ s.t. } |r - r_p| \leqslant 2^{-p}$$

Example

Rational numbers, π, e, ...

Computable real

Definition (Computable Real)

A real $r \in \mathbb{R}$ is computable is one can compute an arbitrary close approximation for a given precision:

$$\text{Given } p \in \mathbb{N}, \text{ compute } r_p \text{ s.t. } |r - r_p| \leqslant 2^{-p}$$

Example

Rational numbers, π, e, ...

Example (Non-computable real)

$$r = \sum_{n=0}^{\infty} d_n 2^{-n}$$

where

$$d_n = 1 \Leftrightarrow \text{the } n^{th} \text{ Turing Machine halts on input } n$$

Computable function

Definition (Computable function)

$f : [a, b] \to \mathbb{R}$ is computable iff $\exists m, \psi$ computable functions s.t $\forall n \in \mathbb{N}$:

- $\forall x, y, |x - y| \leqslant 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leqslant 2^{-n}$ ► effective continuity
- $\forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leqslant 2^{-n}$ ► approximability

Computable function

Definition (Computable function)

$f : [a, b] \to \mathbb{R}$ is computable iff $\exists m, \psi$ computable functions s.t $\forall n \in \mathbb{N}$:

- $\forall x, y, |x - y| \leqslant 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leqslant 2^{-n}$ ▸ effective continuity
- $\forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leqslant 2^{-n}$ ▸ approximability

Definition (Equivalent)

$f : [a, b] \to \mathbb{R}$ is computable iff $\exists M$ a Turing Machine s.t. $\forall x \in [a, b]$ and oracle $\mathcal{O}$ computing x, $M^{\mathcal{O}}$ computes $f(x)$.

Computable function

Definition (Computable function)

$f : [a,b] \to \mathbb{R}$ is computable iff $\exists m, \psi$ computable functions s.t $\forall n \in \mathbb{N}$:

- $\forall x, y, |x - y| \leqslant 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leqslant 2^{-n}$ ► effective continuity
- $\forall r \in \mathbb{Q}, |\psi(r,n) - f(r)| \leqslant 2^{-n}$ ► approximability

Definition (Equivalent)

$f : [a,b] \to \mathbb{R}$ is computable iff $\exists M$ a Turing Machine s.t. $\forall x \in [a,b]$ and oracle $\mathcal{O}$ computing x, $M^{\mathcal{O}}$ computes $f(x)$.

Example

Polynomials, trigonometric functions, $e^{\cdot}$, $\sqrt{\cdot}$, ...

Computable function

Definition (Computable function)

$f : [a, b] \to \mathbb{R}$ is computable iff $\exists m, \psi$ computable functions s.t $\forall n \in \mathbb{N}$:

- $\forall x, y, |x - y| \leqslant 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leqslant 2^{-n}$ ► effective continuity
- $\forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leqslant 2^{-n}$ ► approximability

Definition (Equivalent)

$f : [a, b] \to \mathbb{R}$ is computable iff $\exists M$ a Turing Machine s.t. $\forall x \in [a, b]$ and oracle $\mathcal{O}$ computing x, $M^{\mathcal{O}}$ computes $f(x)$.

Example

Polynomials, trigonometric functions, $e^{\cdot}$, $\sqrt{\cdot}$, ...

Example (Counter-Example)

$$f(x) = \lceil x \rceil \qquad \blacktriangleright \text{ not continuous}$$

Thoughts on the model

- reuses existing theory on Turing machines

Thoughts on the model

- reuses existing theory on Turing machines
- gives “natural” complexity classes related to the classical ones

Thoughts on the model

- reuses existing theory on Turing machines
- gives "natural" complexity classes related to the classical ones
- but feels very discrete machine oriented

Thoughts on the model

- reuses existing theory on Turing machines
- gives “natural” complexity classes related to the classical ones
- but feels very discrete machine oriented

Question

Can we give a purely analog model of computation ?

GPAC

General Purpose Analog Computer

- by Claude Shanon (1941)

GPAC

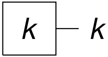
General Purpose Analog Computer

- by Claude Shanon (1941)
- idealization of an analog computer: Differential Analyzer

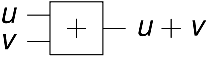
GPAC

General Purpose Analog Computer

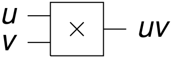
- by Claude Shanon (1941)
- idealization of an analog computer: Differential Analyzer
- circuit built from:



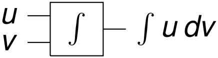
A constant unit



An adder unit



An multiplier unit



An integrator unit

GPAC: beyond the circuit approach

Theorem

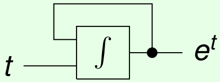
y is generated by a GPAC iff it is a component of the solution $y = (y_1, \ldots, y_d)$ of the ordinary differential equation (ODE):

$$\begin{cases} y'(t) = p(y(t)) \\ y(t_0) = y_0 \end{cases}$$

where p is a vector of polynomials.

GPAC: examples

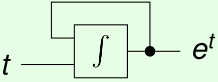
Example (One variable, linear system)



$$\begin{cases} y' = y \\ y(0) = 1 \end{cases}$$

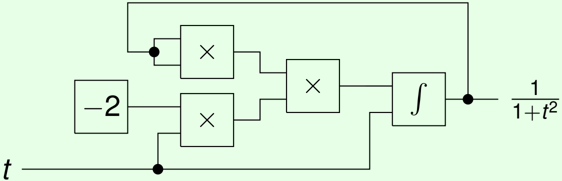
GPAC: examples

Example (One variable, linear system)



$$\begin{cases} y' &= y \\ y(0) &= 1 \end{cases}$$

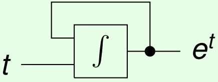
Example (One variable, nonlinear system)



$$\begin{cases} y' &= -2ty^2 \\ y(0) &= 1 \end{cases}$$

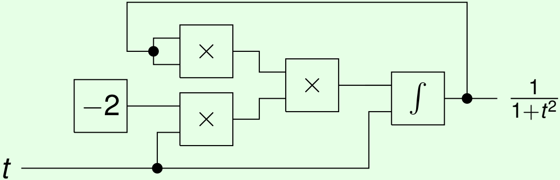
GPAC: examples

Example (One variable, linear system)



$$\begin{cases} y' = y \\ y(0) = 1 \end{cases}$$

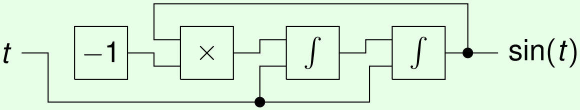
Example (Two variable, nonlinear system)



$$\begin{cases} y' = -2ty^2 \\ y(0) = 1 \\ t' = 1 \\ t(0) = 0 \end{cases}$$

GPAC: examples

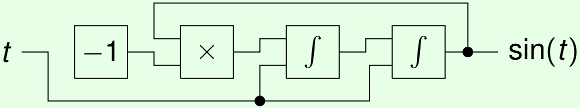
Example (Two variables, linear system)



$$\begin{cases} y' = z \\ z' = -y \\ y(0) = 0 \\ z(0) = 1 \end{cases}$$

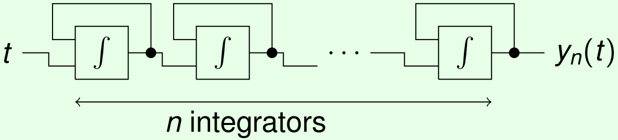
GPAC: examples

Example (Two variables, linear system)



$$\begin{cases} y' &= z \\ z' &= -y \\ y(0) &= 0 \\ z(0) &= 1 \end{cases}$$

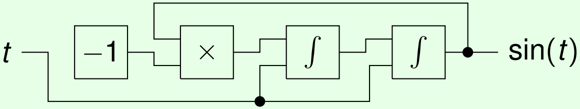
Exercice (Tear your mind apart)



$$\begin{cases} y_1' = y_1 \\ y_2' = y_2 y_1 \\ \vdots \\ y_n' = y_n y_{n-1} \cdots y_2 y_1 \end{cases}$$

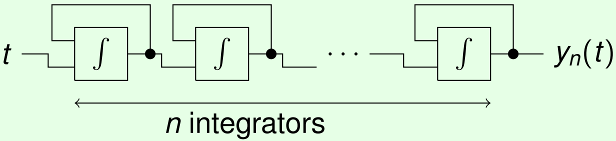
GPAC: examples

Example (Two variables, linear system)



$$\begin{cases} y' &= z \\ z' &= -y \\ y(0) &= 0 \\ z(0) &= 1 \end{cases}$$

Exercice (Tear your mind apart)



$$\begin{cases} y_1(t) = e^t \\ y_2(t) = e^{e^t} \\ \quad\ldots \\ y_n(t) = e^{e^{\cdot^{\cdot^{\cdot^{t}}}}} \end{cases}$$

Slight issue is...

- the GPAC generated functions are analytical

Slight issue is...

- the GPAC generated functions are analytical
- the computable functions from Computable Analysis are “only” continuous

Question

Can we bridge the gap ? Why should we ?

The case of discrete computations

Many models:

- Recursive functions
- Turing machines
- λ-calculus
- circuits
- ...

The case of discrete computations

Many models:

- Recursive functions
- Turing machines
- λ-calculus
- circuits
- ...

Church Thesis

All reasonable discrete models of computation are equivalent.

The case of discrete computations

Many models:

- Recursive functions
- Turing machines
- λ-calculus
- circuits
- ...

Church Thesis

All reasonable discrete models of computation are equivalent.

Can be extended to complexity when relevant.

GPAC: back to the basics

Definition

f is **generated** by a GPAC iff it is a component of the solution y of:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases}$$

GPAC: back to the basics

Definition

f is **generated** by a GPAC iff it is a component of the solution y of:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= y_0 \end{cases}$$

Definition

f is **computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= q(x) \end{cases}$$

satisfies $f(x) = \lim_{t\to\infty} y_1(t)$.

GPAC: back to the basics

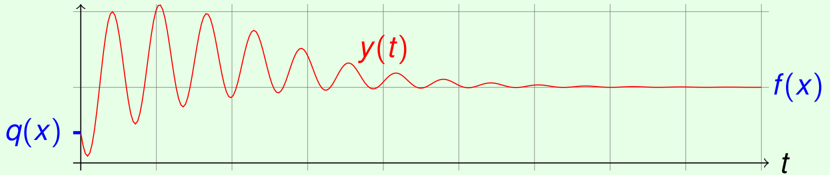
Definition

f is **computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

$$\begin{cases} y' &= p(y) \\ y(t_0) &= q(x) \end{cases}$$

satisfies $f(x) = \lim_{t\to\infty} y_1(t)$.

Example



Computable Analysis = GPAC ? (again)

Theorem (Bournez, Campagnolo, Graça, Hainry)

f is GPAC-computable functions iff it is computable (in the sense of Computable Analysis).

Time Scaling

System	#1	#2
PIVP	$\begin{cases} y'(t) = p(y(t)) \\ y(1) = q(x) \end{cases}$	$\begin{cases} z'(t) = u(t)p(z(t)) \\ u'(t) = u(t) \\ z(t_0) = q(x) \\ u(1) = 1 \end{cases}$

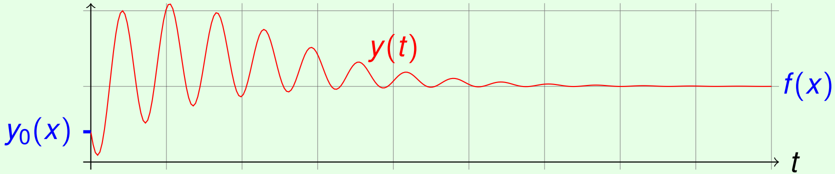
Time Scaling

System	#1	#2
PIVP	$\begin{cases} y'(t) = p(y(t)) \\ y(1) = q(x) \end{cases}$	$\begin{cases} z'(t) = u(t)p(z(t)) \\ u'(t) = u(t) \\ z(t_0) = q(x) \\ u(1) = 1 \end{cases}$

Remark

Same curve, different speed: $u(t) = e^t$ and $z(t) = y(e^t)$

Example



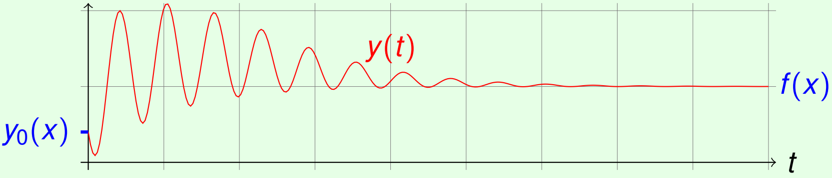
Time Scaling

System	#1	#2
PIVP	$\begin{cases} y'(t) = p(y(t)) \\ y(1) = q(x) \end{cases}$	$\begin{cases} z'(t) = u(t)p(z(t)) \\ u'(t) = u(t) \\ z(t_0) = q(x) \\ u(1) = 1 \end{cases}$
Computed Function	Same	

Remark

Same curve, different speed: $u(t) = e^t$ and $z(t) = y(e^t)$

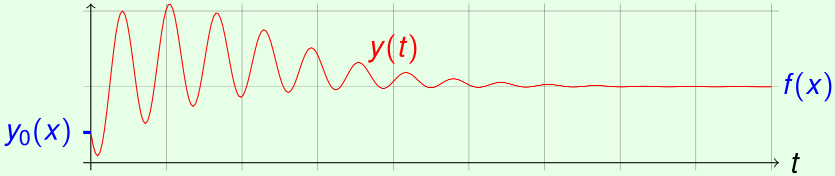
Example



Time Scaling

PIVP	$y' = p(y)$	$z(t) = y(e^t) \rightarrow \begin{cases} z' = up(z) \\ u' = u \end{cases}$
Computed Function	Same	
Convergence		Exponentially faster

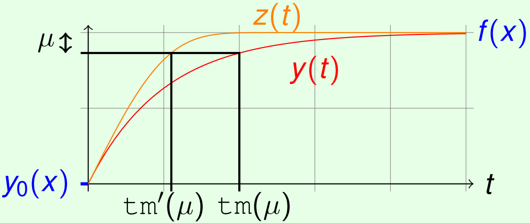
Example



Time Scaling

PIVP	$y' = p(y)$	$z(t) = y(e^t) \rightarrow \begin{cases} z' = up(z) \\ u' = u \end{cases}$
Computed Function	Same	
Time for precision μ	$\mathtt{tm}(\mu)$	$\mathtt{tm}'(\mu) = \log(\mathtt{tm}(\mu))$

Example



$$\|y_1(\mathtt{tm}(\mu)) - f(x)\| \leqslant \mu$$

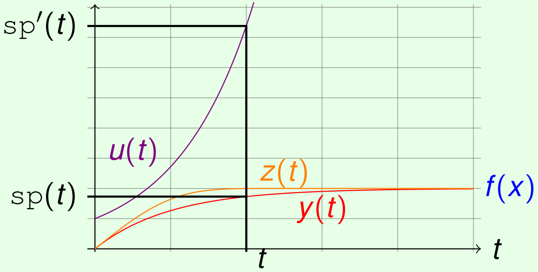
Remark

$\mathtt{tm}$ is not a good measure of complexity.

Time Scaling

PIVP	$y' = p(y)$	$z(t) = y(e^t) \to \begin{cases} z' = up(z) \\ u' = u \end{cases}$
Computed Function	Same	
Time for precision μ	$\mathtt{tm}(\mu)$	$\mathtt{tm}'(\mu) = \log(\mathtt{tm}(\mu))$
Bounding box for PIVP at time t	$\mathtt{sp}(t)$	$\mathtt{sp}'(t) = \max(\mathtt{sp}(e^t), e^t)$

Example



$$\mathtt{sp}(t) = \sup_{\xi \in [1,t]} \|y(\xi)\|$$

$$\mathtt{sp}'(t) = \sup_{\xi \in [1,t]} \|z(\xi), u(\xi)\|$$

Time Scaling

PIVP	$y' = p(y)$	$z(t) = y(e^t) \rightarrow \begin{cases} z' = up(z) \\ u' = u \end{cases}$
Computed Function	Same	
Time for precision μ	$\mathtt{tm}(\mu)$	$\mathtt{tm}'(\mu) = \log(\mathtt{tm}(\mu))$
Bounding box for PIVP at time t	$\mathtt{sp}(t)$	$\mathtt{sp}'(t) = \max(\mathtt{sp}(e^t), e^t)$

Remark

- $\mathtt{tm}(\mu)$ and $\mathtt{sp}(t)$ depend on the convergence rate

Time Scaling

PIVP	$y' = p(y)$	$z(t) = y(e^t) \rightarrow \begin{cases} z' = up(z) \\ u' = u \end{cases}$
Computed Function	Same	
Time for precision μ	$\mathtt{tm}(\mu)$	$\mathtt{tm}'(\mu) = \log(\mathtt{tm}(\mu))$
Bounding box for PIVP at time t	$\mathtt{sp}(t)$	$\mathtt{sp}'(t) = \max(\mathtt{sp}(e^t), e^t)$
Bounding box for PIVP at precision μ	$\mathtt{sp}(\mathtt{tm}(\mu))$	$\max(\mathtt{sp}(\mathtt{tm}(\mu)), \mathtt{tm}(\mu))$

Remark

- $\mathtt{tm}(\mu)$ and $\mathtt{sp}(t)$ depend on the convergence rate
- $\mathtt{sp}(\mathtt{tm}(\mu))$ seems not

Proper Measures

Proper measures of “complexity”:

- time scaling invariant
- property of the curve

Proper Measures

Proper measures of “complexity”:

- time scaling invariant
- property of the curve

Possible choices:

- Bounding Box at precision $\mu \Rightarrow$ Ok but geometric interpretation ?

Proper Measures

Proper measures of “complexity”:

- time scaling invariant
- property of the curve

Possible choices:

- Bounding Box at precision μ $\Rightarrow$ Ok but geometric interpretation ?
- Length of the curve until precision μ $\Rightarrow$ Much more intuitive

Complexity based on the length of the curve

Definition

f is **poly-computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

$$\begin{cases} y'(t) = p(y(t)) \\ y(t_0) = q(x) \end{cases}$$

satisfies that $\|f(x) - y_1(t)\| \leqslant e^{-\mu}$ when $\ell(t) \geqslant \mathtt{len}(\|x\|, \mu)$ where:

- $\mathtt{len}$ is a polynomial
- $\ell(t)$ is the length of the curve y from 0 to t

An equivalent classes

Definition

f is **poly-computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

$$\begin{cases} y'(t) = p(y(t)) \\ y(t_0) = q(x) \end{cases}$$

satisfies that:

- $\|f(x) - y_1(t)\| \leqslant e^{-\mu}$ when $t \geqslant \text{poly}(\|x\|, \mu)$
- $\|y(t)\| \leqslant \text{poly}(\|x\|, t)$

Equivalence theorem

Theorem

f is poly-computable if and only if it is computable in polytime in the sense of Computable Analysis.

Conclusion

- Complexity theory for a continuous model of computation
- Natural, machine-independent definition of real computable functions

Conclusion

- Complexity theory for a continuous model of computation
- Natural, machine-independent definition of real computable functions

Future work:

- Other complexity classes (time or space)
- Better understand how the restrictions constraint the complexity

Questions ?

- Do you have any questions ?