Maik Thanh Nguyen

IP = PSPACE

Dresden, 17.07.2025

Content

- What is IP?
- Arithmetization
- Introducing the protocol
- Problems within the protocol and solutions
- Some protocol examples
- Correctness

What is IP?

- A prover tries to convince the Verifier of membership
- Verifier sceptically checks the Prover's arguments before making a decision
- The interaction might involve several rounds of communication
- The prover might have unlimited power but the verifier operate in P
- The message length and number of rounds should be polynomial

What is IP?

- A prover tries to convince the Verifier of membership
- Verifier sceptically checks the Prover's arguments before making a decision
- The interaction might involve several rounds of communication
- The prover might have unlimited power but the verifier operate in P
- The message length and number of rounds should be polynomial
- A language L is in IP if there is a polynomial verifier V such that, for every word w :

$$\text{if } w \in L \text{ then there is a Prover } P \text{ with } \Pr[V \leftrightarrow P \text{ accepts}] \geq \frac{2}{3}$$

$$\text{if } w \notin \text{L then for all Prover } P \text{ with } \Pr\,[V \leftrightarrow P \text{ accepts}] \leq \frac{1}{3}$$

PSPACE $\subseteq$ IP

For this inclusion, we use a well known PSPACE-complete problem, namely True-QBF

PSPACE $\subseteq$ IP

For this inclusion, we use a well known PSPACE-complete problem, namely True-QBF

- QBF-Truth (abbrev. with QBF) is the set of all valid quantified boolean formulas without free variables and for any variable p we have $p \in \{0, 1\}$

PSPACE $\subseteq$ IP

For this inclusion, we use a well known PSPACE-complete problem, namely True-QBF

- QBF-Truth (abbrev. with QBF) is the set of all valid quantified boolean formulas without free variables and for any variable p we have $p \in \{0, 1\}$
- $\forall x \exists y\, (x \vee y)$, $\exists x \exists y \neg(x \wedge y)$

PSPACE $\subseteq$ IP

For this inclusion, we use a well known PSPACE-complete problem, namely True-QBF

- QBF-Truth (abbrev. with QBF) is the set of all valid quantified boolean formulas without free variables and for any variable p we have $p \in \{0, 1\}$
- $\forall x \exists y \, (x \vee y)$, $\exists x \exists y \neg(x \wedge y)$
- QBF-Truth$_{NNF}$ (abbrev. with QBF') is QBF-Truth but negations are only applied on variables
- $\exists x \exists y \neg(x \wedge y)$ is not in NNF but $\exists x \exists y (\neg x \vee \neg y)$

Arithmetization

The prover has to convince the verifier that the formula is valid but in case of an invalid formula it should reject with high probability (for all prover)

Arithmetization

The prover has to convince the verifier that the formula is valid but in case of an invalid formula it should reject with high probability (for all prover)

- The idea is to arithmetize the formula

Arithmetization

The prover has to convince the verifier that the formula is valid but in case of an invalid formula it should reject with high probability (for all prover)

- The idea is to arithmetize the formula
- $x \wedge y$ becomes $x * y$
- $x \vee y$ becomes $x + y$
- $\neg x$ becomes $1 - x$

Arithmetization

The prover has to convince the verifier that the formula is valid but in case of an invalid formula it should reject with high probability (for all prover)

- The idea is to arithmetize the formula
- $x \land y$ becomes $x * y$
- $x \lor y$ becomes $x + y$
- $\neg x$ becomes $1 - x$
- $\forall x\, \phi$ becomes $a_0 * a_1$ where $a_0 = \phi[x := 0]$ and $a_1 = \phi[x := 1]$
- $\exists x\, \phi$ becomes $a_0 + a_1$ where $a_0 = \phi[x := 0]$ and $a_1 = \phi[x := 1]$

Arithmetization

Example : $\phi = \forall x \exists y \, (\neg x \vee \neg y)$

Arithmetization

Example : $\phi = \forall x \exists y \, (\neg x \vee \neg y)$
Arithmetize :
$(\neg x \vee \neg y) \xrightarrow{arith.} ((1-x)+(1-y)) \xrightarrow{\exists arith.} \sum_{y\in\{0,1\}}((1-x)+(1-y)) \xrightarrow{\forall arith.}$

Arithmetization

Example : $\phi = \forall x \exists y \, (\neg x \vee \neg y)$
Arithmetize :
$(\neg x \vee \neg y) \xrightarrow{arith.} ((1-x)+(1-y)) \xrightarrow{\exists arith.} \sum_{y\in\{0,1\}}((1-x)+(1-y)) \xrightarrow{\forall arith.}$
$\prod_{x\in\{0,1\}} \sum_{y\in\{0,1\}}((1-x)+(1-y)) = \phi_{arith}$

- $\phi_{arith} = 3$
- If ϕ is true then $\phi_{arith} > 0$
- If ϕ is false then $\phi_{arith} = 0$
- This can be shown by structural induction

Protocol-Problem

Before we start the communication, we will encounter a problem.

Protocol-Problem

Before we start the communication, we will encounter a problem.

- On input <ϕ>, the prover sends a value $c > 0$ to the verifier and tries to convince that c is the arithmetic value of ϕ

Protocol-Problem

Before we start the communication, we will encounter a problem.

- On input <ϕ>, the prover sends a value $c > 0$ to the verifier and tries to convince that c is the arithmetic value of ϕ
- ϕ_{arith} could be double exponential

$\phi = \forall x_1 ... \forall x_m \exists y \exists z.(y \vee z)$. What is ϕ_{arith}? We calculate it step by step.

Protocol-Problem

Before we start the communication, we will encounter a problem.

- On input <ϕ>, the prover sends a value $c > 0$ to the verifier and tries to convince that c is the arithmetic value of ϕ
- ϕ_{arith} could be double exponential

$\phi = \forall x_1 ... \forall x_m \exists y \exists z.(y \vee z)$. What is ϕ_{arith}? We calculate it step by step.
$\phi' = \exists y \exists z(y \vee z)$. Then $\phi'_{arith} = \sum_{y \in \{0,1\}} \sum_{z \in \{0,1\}} (y + z) = 4$

Protocol-Problem

Before we start the communication, we will encounter a problem.

- On input <ϕ>, the prover sends a value $c > 0$ to the verifier and tries to convince that c is the arithmetic value of ϕ
- ϕ_{arith} could be double exponential

$\phi = \forall x_1 ... \forall x_m \exists y \exists z.(y \vee z)$. What is ϕ_{arith}? We calculate it step by step.
$\phi' = \exists y \exists z(y \vee z)$. Then $\phi'_{arith} = \sum_{y \in \{0,1\}} \sum_{z \in \{0,1\}} (y + z) = 4$
$\phi_{arith} = \prod_{x_1 \in \{0,1\}} \cdots \prod_{x_m \in \{0,1\}} \phi'_{arith} = 4^{2^m}$

- It holds for formula ϕ with string length $n : \phi_{arith} \leq 2^{2^n}$.
  This can be shown by structural induction.
- We solve this problem by using modulo with a suitable value

Protocol-Problem

- Pick a value $k \geq 2^n$ with two conditions :
- k must be presentable in linear many bits

Protocol-Problem

- Pick a value $k \geq 2^n$ with two conditions :
- k must be presentable in linear many bits
- the calculation $mod\ k$ must preserve "> 0" for valid and "$= 0$" for invalid formulas
- It holds that: for any $a \leq 2^{2^n}, a > 0$, there exists a prime number $k \in [2^n, 2^{3n}]$ s.t. $a \not\equiv 0\ (mod\ k)$

Protocol continued

- Prover sends value c, prime number k and a proof b for the prime number property (it is possible to give a polynomial proof)

Protocol continued

- Prover sends value c, prime number k and a proof b for the prime number property (it is possible to give a polynomial proof)
- Verifier check $c > 0$, $k \in [2^n, 2^{3n}]$ and b is a correct proof for prime property

Even if k and b are correct, the verifier stays sceptical about c.

Protocol continued

- Prover sends value c, prime number k and a proof b for the prime number property (it is possible to give a polynomial proof)
- Verifier check $c > 0$, $k \in [2^n, 2^{3n}]$ and b is a correct proof for prime property

Even if k and b are correct, the verifier stays sceptical about c.

- If $\phi = \phi_1 \wedge \phi_2$, then ask prover to send a_1 and a_2 and check $c = a_1 * a_2$. If it's true then ask the prover to prove that the value of ϕ_{1arith} is a_1 and ϕ_{2arith} is a_2
- For $\phi = \phi_1 \vee \phi_2$, we ask for a_1 and a_2 s.t. c = $a_1 + a_2$

Protocol continued

- Prover sends value c, prime number k and a proof b for the prime number property (it is possible to give a polynomial proof)
- Verifier check $c > 0$, $k \in [2^n, 2^{3n}]$ and b is a correct proof for prime property

Even if k and b are correct, the verifier stays sceptical about c.

- If $\phi = \phi_1 \wedge \phi_2$, then ask prover to send a_1 and a_2 and check $c = a_1 * a_2$. If it's true then ask the prover to prove that the value of ϕ_{1arith} is a_1 and ϕ_{2arith} is a_2
- For $\phi = \phi_1 \vee \phi_2$, we ask for a_1 and a_2 s.t. c = $a_1 + a_2$
- In case $\phi = \forall x \phi_1$ we asked for a polynomial $p(x)$ that represents the arithmetic presentation of ϕ_1 where x is free and we check $c = p(0) * p(1)$
- If it is true, the verifier sends randomly a number d between $\{0, ..., k-1\} = GF(K)$ and caluclate $p(d)$. Now the verifier expects the prover to prove the value of $\phi_1[x := d]$ is $p(d)$
- The same process happens when we have $\phi = \exists x \phi_1$, but we check $c = p(0) + p(1)$

Protocol continue

- When every variable got a number in $GF(K)$, say $y_1, ..., y_n$ the verifier calculates $\phi_{arith}(y_1, ..., y_n)$ and accepts if its equal to $q(y_1, ..., y_n)$ (last polynomial sent by prover), else reject

Example

$\phi = \forall x \exists y(\neg x \vee y) \wedge \exists z \exists w\,(z \vee w)$

$$\phi_{arith} = \underbrace{(\prod_x \sum_y ((1-x)+y))}_{\phi_{\mathbf{1}arith}} * \underbrace{(\sum_z \sum_w (z+w))}_{\phi_{\mathbf{2}arith}} \quad x, y, z, w \in \{0,1\}$$

Example

$\phi = \forall x \exists y(\neg x \vee y) \wedge \exists z \exists w\,(z \vee w)$

$$\phi_{arith} = \underbrace{(\prod_x \sum_y ((1-x)+y))}_{\phi_{\mathbf{1}arith}} * \underbrace{(\sum_z \sum_w (z+w))}_{\phi_{\mathbf{2}arith}} \quad x, y, z, w \in \{0,1\}$$

Prover $\xrightarrow{c = 12,\, k = 11,\, b}$ Verifier

check $c > 0, k, b$

Example

$$\phi = \forall x \exists y (\neg x \vee y) \wedge \exists z \exists w \, (z \vee w)$$

$$\phi_{arith} = \underbrace{(\prod_x \sum_y ((1-x)+y))}_{\phi_{\mathbf{1}arith}} * \underbrace{(\sum_z \sum_w (z+w))}_{\phi_{\mathbf{2}arith}} \quad x, y, z, w \in \{0,1\}$$

$$\begin{array}{lcl}
\text{Prover} & & \text{Verifier} \\
 & \xrightarrow{\quad c = 12,\, k = 11,\, b \quad} & \text{check } c > 0, k, b \\
 & \xleftarrow{\quad \text{asks for } a_1 \text{ and } a_2 \quad} & \\
 & \xrightarrow{\quad \text{sends } a_1 = 3,\, a_2 = 4 \quad} & \text{check } c = a_1 * a_2 \\
 & \xleftarrow{\quad \text{prove } a_1 = \phi_{1arith},\, a_2 = \phi_{2arith} \quad} &
\end{array}$$

Example

$\phi = \forall x \exists y (\neg x \vee y) \wedge \exists z \exists w \, (z \vee w)$

$$\phi_{arith} = \underbrace{(\prod_x \sum_y ((1-x)+y))}_{\phi_{\mathbf{1}arith}} * \underbrace{(\sum_z \sum_w (z+w))}_{\phi_{\mathbf{2}arith}} \quad x, y, z, w \in \{0,1\}$$

Prover		Verifier
	$\xrightarrow{c = 12,\, k = 11,\, b}$	check $c > 0, k, b$
	$\xleftarrow{\text{asks for } a_1 \text{ and } a_2}$	
	$\xrightarrow{\text{sends } a_1 = 3,\, a_2 = 4}$	check $c = a_1 * a_2$
	$\xleftarrow[\text{send me } p_1(x) \text{ and } p_2(z)]{\text{prove } a_1 = \phi_{1arith},\, a_2 = \phi_{2arith}}$	

Example continue

Prover

calculate $\phi_{1arith}(x)$, $\phi_{2arith}(z)$

$$\xrightarrow[p_1(x)=\sum_y (1-x)+y=-2x+3]{\text{sends } p_1(x)=\phi_{1arith}(x),\ \phi_{2arith}(z)}$$

Verifier

check $a_\mathbf{1} = p_\mathbf{1}(0) * p_\mathbf{1}(1)$

check $a_\mathbf{2} = p_\mathbf{2}(0) + p_\mathbf{2}(1)$

Example continue

Prover		Verifier
calculate $\phi_{1arith}(x)$, $\phi_{2arith}(z)$	sends $p_1(x) = \phi_{1arith}(x)$, $\phi_{2arith}(z)$ $\longrightarrow$ $p_1(x) = \sum_y (1-x) + y = -2x + 3$	check $a_{\mathbf{1}} = p_{\mathbf{1}}(0) * p_{\mathbf{1}}(1)$ check $a_{\mathbf{2}} = p_{\mathbf{2}}(0) + p_{\mathbf{2}}(1)$
	sends $p_1(d) = 10, p_2(d) = 5$ $\longleftarrow$ ask for $p_1'(d, y), p_2'(d, w)$	Choose randomly $d \in GF(k)$,say $d = 2$

Example continue

Prover		Verifier
calculate $\phi_{1arith}(x)$, $\phi_{2arith}(z)$	sends $p_1(x) = \phi_{1arith}(x), \phi_{2arith}(z)$ $\longrightarrow$ $p_1(x) = \sum_y (1-x) + y = -2x + 3$	check $a_1 = p_1(0) * p_1(1)$ check $a_2 = p_2(0) + p_2(1)$
	sends $p_1(d) = 10, p_2(d) = 5$ $\longleftarrow$ ask for $p_1'(d,y), p_2'(d,w)$	Choose randomly $d \in GF(k)$,say $d = 2$
	sends $p_1'(d,y) = \phi_{1arith}(d,y)$ $\longrightarrow$ sends $p_2'(d,w) = \phi_{2arith}(d,w)$	$p_1'(d,0) * p_1'(d,1) = p_1(d)$ $p_2'(d,0) * p_2'(d,1) = p_2(d)$ Choose $c \in GF(k)$

Example continue

Prover | **Verifier**

Prover: calculate $\phi_{1arith}(x)$, $\phi_{2arith}(z)$

$$\xrightarrow[p_1(x) = \sum_y (1-x) + y = -2x + 3]{\text{sends } p_1(x) = \phi_{1arith}(x),\ \phi_{2arith}(z)}$$

Verifier: check $a_1 = p_1(0) * p_1(1)$

check $a_2 = p_2(0) + p_2(1)$

$$\xleftarrow[\text{ask for } p_1'(d,y),\ p_2'(d,w)]{\text{sends } p_1(d) = 10,\ p_2(d) = 5}$$

Verifier: Choose randomly $d \in GF(k)$,say $d = 2$

$$\xrightarrow[\text{sends } p_2'(d,w) = \phi_{2arith}(d,w)]{\text{sends } p_1'(d,y) = \phi_{1arith}(d,y)}$$

Verifier: $p_1'(d,0) * p_1'(d,1) = p_1(d)$

$p_2'(d,0) * p_2'(d,1) = p_2(d)$

Choose $c \in GF(k)$

The verifier check $\phi_{arith}(d,c,d,c) = p_1'(d,c) * p_2'(d,c)$. It accepts.

Next Example

$$\phi = \forall x \exists y\, (x \land y) \xrightarrow{arith.} \phi_{arith} = \prod_x \sum_y x * y$$

Next Example

$$\phi = \forall x \exists y \, (x \wedge y) \xrightarrow{arith.} \phi_{arith} = \prod_x \sum_y x * y$$

Prover cannot tell the truth because the verifier would reject instantly.

Prover Verifier

Next Example

$$\phi = \forall x \exists y\,(x \wedge y) \xrightarrow{arith.} \phi_{arith} = \prod_x \sum_y x * y$$

Prover cannot tell the truth because the verifier would reject instantly.

$$\text{Prover} \xrightarrow{c = 3,\, p = 131,\, b} \text{Verifier}$$

check $c > 0, 131, b$

$$\xleftarrow{\text{asks for } p_1(x)}$$

Next Example

$$\phi = \forall x \exists y \, (x \wedge y) \xrightarrow{arith.} \phi_{arith} = \prod_x \sum_y x * y$$

Prover cannot tell the truth because the verifier would reject instantly.

$$\text{Prover} \xrightarrow{c = 3,\, p = 131,\, b} \text{Verifier}$$

check $c > 0, 131, b$

$$\xleftarrow{\text{asks for } p_1(x)}$$

$$\xrightarrow[\text{prover has to lie}]{\text{sends } p_1(x) = 2x + x}$$

check $p_1(0) * p_1(1) = c$

Next Example

$$\phi = \forall x \exists y\, (x \wedge y) \xrightarrow{arith.} \phi_{arith} = \prod_x \sum_y x * y$$

Prover cannot tell the truth because the verifier would reject instantly.

Prover		Verifier
	$c = 3, p = 131, b$ $\longrightarrow$	check $c > 0, 131, b$
	asks for $p_1(x)$ $\longleftarrow$	
	sends $p_1(x) = 2x + x$ $\longrightarrow$ prover has to lie	check $p_1(0) * p_1(1) = c$
	sends $p_1(d)$ $\longleftarrow$	choose $d \in GF(p)$

If $\phi_{arith}(d) = p_1(d = 3) = 7$ then we lost. Chances should be low

Next Example

$$\phi = \forall x \exists y\, (x \wedge y) \xrightarrow{arith.} \phi_{arith} = \prod_x \sum_y x * y$$

Prover cannot tell the truth because the verifier would reject instantly.

Prover $\xrightarrow{c = 3,\, p = 131,\, b}$ Verifier

Verifier: check $c > 0, 131, b$

$\xleftarrow{\text{asks for } p_1(x)}$

$\xrightarrow{\text{sends } p_1(x) = 2x + x}$

prover has to lie

check $p_1(0) * p_1(1) = c$

choose $d \in GF(p)$

$\xleftarrow{\text{sends } p_1(d)}$

If $\phi_{arith}(d) = p_1(d = 3) = 7$ then we lost. Chances should be low

$\xrightarrow{\text{sends } p_1'(d, y) = 7 * y}$

check $p_1'(d, 0) + p_1'(d, 1) = p_1(d)$. Chooses $g = 4$

The verifier check $\phi_{arith}(d, g) = p_1'(d, g)$. $12 \neq 28$. Verifier rejects.

Simple QBF

- Problem : Prover could send polynomial with exponential degree and the verifier cannot check it

Simple QBF

- Problem : Prover could send polynomial with exponential degree and the verifier cannot check it
- Example : $\phi = \forall x_1, ..., \forall x_m (x_1 \vee ... \vee x_m) \xrightarrow{arith\phi(x_1)} \phi_{arith}(x_1) = \prod_{x_2} ... \prod_{x_m} (x_1 + x_2 + ... + x_m) \rightarrow deg(\phi_{arith}(x)) \leq 2^{m-1}$

Simple QBF

- Problem : Prover could send polynomial with exponential degree and the verifier cannot check it
- Example : $\phi = \forall x_1, ..., \forall x_m(x_1 \vee ... \vee x_m) \xrightarrow{arith\phi(x_1)} \phi_{arith}(x_1) = \prod_{x_2} ... \prod_{x_m}(x_1 + x_2 + ... + x_m) \to deg(\phi_{arith}(x)) \leq 2^{m-1}$
- A QBF ϕ is called simple, if any occurence of a variable is separated by at most one universal quantifier from its point of quantification.
- Example : $\forall x_1 \forall x_2 \exists x_3 \; [(x_1 \vee x_2) \wedge \forall x_4(x_2 \vee x_3 \vee x_4)]$
- Counterexample : $\forall x_1 \forall x_2[(x_1 \vee x_2) \wedge \forall x_3(\neg x_1 \vee x_3)]$

Simple QBF

- Problem : Prover could send polynomial with exponential degree and the verifier cannot check it
- Example : $\phi = \forall x_1, ..., \forall x_m(x_1 \vee ... \vee x_m) \xrightarrow{arith\phi(x_1)} \phi_{arith}(x_1) = \prod_{x_2} ... \prod_{x_m}(x_1 + x_2 + ... + x_m) \rightarrow deg(\phi_{arith}(x)) \leq 2^{m-1}$
- A QBF ϕ is called simple, if any occurence of a variable is separated by at most one universal quantifier from its point of quantification.
- Example : $\forall x_1 \forall x_2 \exists x_3 \, [(x_1 \vee x_2) \wedge \forall x_4(x_2 \vee x_3 \vee x_4)]$
- Counterexample : $\forall x_1 \forall x_2[(x_1 \vee x_2) \wedge \forall x_3(\neg x_1 \vee x_3)]$
- We can reduce any QBF formula into a Simple QBF in polynomial time
- If ϕ is a simple QBF formula of length n,and $p(x)$ be a polynomial of ϕ_{arith}. Then $deg(p(x) \leq 2n)$. This can be shown by induction.

Correctness

Now we check for the correctness of the protocol

- If ϕ is true, a truthful Prover can ensure that V accepts

Correctness

Now we check for the correctness of the protocol

- If ϕ is true, a truthful Prover can ensure that V accepts
- If ϕ is false, the chance that V accepts is very small

Correctness

Now we check for the correctness of the protocol

- If ϕ is true, a truthful Prover can ensure that V accepts
- If ϕ is false, the chance that V accepts is very small
- We use "Schwartz-Zippel" lemma. Let p be a non-zero multivariate polynomial $p(x_1, ..., x_m)$ with degree $\leq d$ and S a finite set of integers. If $a_1, ..., a_m$ are chosen randomly independently and uniformly from S, then

$$Pr[p(a_1, ..., a_m) = 0] \leq \frac{d}{|S|}$$

Correctness

Now we check for the correctness of the protocol

- If ϕ is true, a truthful Prover can ensure that V accepts
- If ϕ is false, the chance that V accepts is very small
- We use "Schwartz-Zippel" lemma. Let p be a non-zero multivariate polynomial $p(x_1, ..., x_m)$ with degree $\leq d$ and S a finite set of integers. If $a_1, ..., a_m$ are chosen randomly independently and uniformly from S, then

$$Pr[p(a_1, ..., a_m) = 0] \leq \frac{d}{|S|}$$

- Prover sends wrong polynomial $p \neq h = \phi_{arith}$ in the i-th round. Verifier chooses randomly $c \in GF(p)$ where $p \geq 2^n$. Furthermore we have $deg(g - h) \leq 2n$. Then $Pr[p(c) = h(c)] = Pr[\text{Error i-th round}] \leq \frac{2n}{2^n}$.

Correctness continue

- That means Pr[No Error in i-th round]$\geq 1 - \frac{2n}{2^n}$
- Because random number are chosen independently, and after $m \leq n$ rounds, we have :

$$Pr[Error] = 1 - Pr[No\ Error] = 1 - \prod_{i=1}^{m} Pr[No\ Error\ in\ i\text{-}th\ round]$$

$$\leq (1 - (1 - \frac{2n}{2^n}))^n$$

- The last approximation is true because : $\prod_{i=1}^{m} Pr[no\ error\ in\ i\text{-}th\ round] \geq (1 - \frac{2n}{2^n})^m \geq (1 - \frac{2n}{2^n})^n$

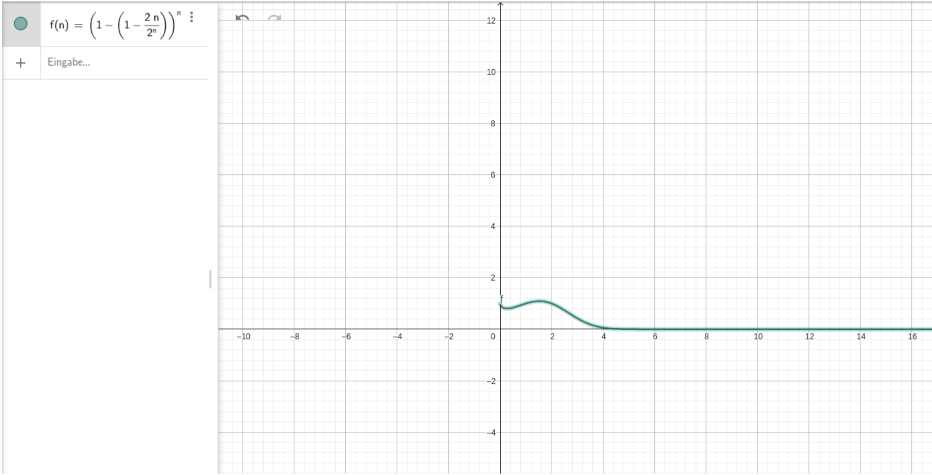

Figure: $n \to \infty$, $Range = (-2, 0)$