TECHNISCHE UNIVERSITÄT DRESDEN

hello

IP = PSPACE

CONFERENCE, DATE

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 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 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 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 $\exists x \exists y(\neg x \vee \neg y)$

PSPACE $\subseteq$ IP

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

- QBF-Truth 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 $\exists x \exists y(\neg x \vee \neg y)$
- QBF $\leq_m^P$ QBF-Truth$_{NNF}$ can be easily done by the verifier
- it suffices to show QBF' $\in$ IP because we can reduce any problem in PSPACE to QBF in polynomial time

How do we find an algorithm for QBF' s.t it satisfies the IP conditions?

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
- $x \wedge y$ becomes x*y
- $x \vee y$ becomes x+y
- $\neg x$ becomes 1-x

x	y	$x \wedge y = x * y$	$x \wedge y = x + y$	$\neg(x \wedge y) = 1 - (x * y)$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	2	0

$$\phi(x_1, x_2, ..., x_n) = 1 \Leftrightarrow \phi_{arith}(x_1, x_2, ..., x_n) > 0$$

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

- $\forall x \, \phi$ becomes $a_0 * a_1$ where $a_0 = \phi[x := 0]$ and $a_1 = \phi[x := 1]$

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

- $\forall x\, \phi$ 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

How do we arithmetize $\forall$ and $\exists$?

- $\forall x\, \phi$ 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 $a_1 = \phi[x := 1]$

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

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

- $\forall x\, \phi$ 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 $a_1 = \phi[x := 1]$

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

Arithmetize : $\neg(x \wedge y) \xrightarrow{arith.} 1 - (x * y) \xrightarrow{\exists arith.} \sum_{y \in \{0,1\}} 1 - (x * y) \xrightarrow{\forall arith.}$

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

- $\forall x\, \phi$ 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 $a_1 = \phi[x := 1]$

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

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

- $\phi_{arith} = 2$

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

- $\forall x\, \phi$ 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 $a_1 = \phi[x := 1]$

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

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

- $\phi_{arith} = 2$
- If ϕ is true then $\phi_{arith} > 0$
- If ϕ is false then $\phi_{arith} = 0$

Arithmetization

How do we arithmetize $\forall$ and $\exists$?

- $\forall x\, \phi$ 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 $a_1 = \phi[x := 1]$

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

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

- $\phi_{arith} = 2$
- If ϕ is true then $\phi_{arith} > 0$
- If ϕ is false then $\phi_{arith} = 0$
- This can be shown by structural induction

Arithmetization

- $\phi = x$. If ϕ is true then $\phi_{arith} = 1$ and if ϕ is true then $\phi_{arith} = 0$
- $\phi = \neg x$ is the same but turned around

Arithmetization

- $\phi = x$. If ϕ is true then $\phi_{arith} = 1$ and if ϕ is true then $\phi_{arith} = 0$
- $\phi = \neg x$ is the same but turned around
- Suppose ϕ_1 and ϕ_2 are true, that means $\phi_{1arith} > 0$ and $\phi_{2arith} > 0$ (it's similar for the other case)

Arithmetization

- $\phi = x$. If ϕ is true then $\phi_{arith} = 1$ and if ϕ is true then $\phi_{arith} = 0$
- $\phi = \neg x$ is the same but turned around
- Suppose ϕ_1 and ϕ_2 are true, that means $\phi_{1arith} > 0$ and $\phi_{2arith} > 0$ (it's similar for the other case)
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ also holds
- For $\forall \phi_1$, we have by induction that $\phi_1[x := 0], \phi_1[x := 1]$ is true, so the multiplication of two positive value is positive (same for $\exists \phi_1$)

Protocol-Problem

How do we start the communication between prover and verifier?

Protocol-Problem

How do we start the communication between prover and verifier?

- On input <ϕ>, the prove sends a value c>0 to the verifier and tries to convince that c is the arithmetic value of ϕ
  (Remember : the verifier can not calculate its value by itself because it could be double exponential)

Protocol-Problem

How do we start the communication between prover and verifier?

- On input <ϕ>, the prove sends a value c>0 to the verifier and tries to convince that c is the arithmetic value of ϕ
  (Remember : the verifier can not calculate its value by itself because it could be double exponential)
- Problem : the value could be exponential but the verifier has only polynomial time

Protocol-Problem

How do we start the communication between prover and verifier?

- On input <ϕ>, the prove sends a value c>0 to the verifier and tries to convince that c is the arithmetic value of ϕ
  (Remember : the verifier can not calculate its value by itself because it could be double exponential)
- Problem : the value could be exponential but the verifier has only polynomial time

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

Protocol-Problem

How do we start the communication between prover and verifier?

- On input <ϕ>, the prove sends a value c>0 to the verifier and tries to convince that c is the arithmetic value of ϕ
  (Remember : the verifier can not calculate its value by itself because it could be double exponential)
- Problem : the value could be exponential but the verifier has only polynomial time

$\phi = \forall x_1 ... \forall x_m \exists y \exists z.(y \vee z)$. What is ϕ_{arith}? We caluclate it step by step. Let $\phi' = \exists y \exists z (y \vee z)$. Then $\phi'_{arith} = \sum_{y\{0,1\}} \sum_{z\{0,1\}} y + z - y * z = 3$

Protocol-Problem

How do we start the communication between prover and verifier?

- On input <ϕ>, the prove sends a value c>0 to the verifier and tries to convince that c is the arithmetic value of ϕ
  (Remember : the verifier can not calculate its value by itself because it could be double exponential)
- Problem : the value could be exponential but the verifier has only polynomial time

$\phi = \forall x_1 ... \forall x_m \exists y \exists z.(y \vee z)$. What is ϕ_{arith}? We caluclate it step by step. Let $\phi' = \exists y \exists z(y \vee z)$. Then $\phi'_{arith} = \sum_{y\{0,1\}} \sum_{z\{0,1\}} y + z - y * z = 3$
$\phi_{arith} = \prod_{x_1 \in \{0,1\}} \cdots \prod_{x_m \in \{0,1\}} \phi'_{arith} = 3^{2^m}$

- It holds that for formula ϕ with string length n : $\phi_{arith} \leq 2^{2^n}$
- 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
- Number Theory Theorem : for any $a \leq 2^{2^n}$, $a > 0$, there exist 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 of ϕ_1 is a_1 and ϕ_2 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 of ϕ_1 is a_1 and ϕ_2 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 $\phi = \exists \phi_1$, but we check $c = p(0) + p(1)$