Game arguments in computability theory

Alexander Shen
LIRMM, CNRS & UM2, Montpellier; on leave from
ИППИ РАН, Москва

Outline

"A large part of computability theory is essentially about games"

Outline

"A large part of computability theory is essentially about games"

Three illustrations:

Outline

"A large part of computability theory is essentially about games"

Three illustrations:

- Unique programs

Outline

"A large part of computability theory is essentially about games"

Three illustrations:

- Unique programs
- Gap between conditional complexity and total conditional complexity

Outline

“A large part of computability theory is essentially about games”

Three illustrations:

- Unique programs
- Gap between conditional complexity and total conditional complexity
- Finding strings with equal distances

Universal functions

Partial function $U : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$

Universal functions

Partial function $U : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$

Represented by a table:

		x	
n		$U(n,x)$	

Universal functions

Partial function $U : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$

Represented by a table:

		x	
n		$U(n, x)$	

Empty cells mean undefined values

Universal functions

Partial function $U : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$

Represented by a table:

		x	
n		$U(n,x)$	

Empty cells mean undefined values

Function is computable iff the table can be filled by a (non-terminating) algorithmic process

Universal functions

Partial function $U : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$

Represented by a table:

		x	
n		$U(n,x)$	

Empty cells mean undefined values

Function is computable iff the table can be filled by a (non-terminating) algorithmic process

n-th row corresponds to unary function $U_n : x \mapsto U(n,x)$

Universal functions

Partial function $U : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$

Represented by a table:

		x	
n		$U(n,x)$	

Empty cells mean undefined values

Function is computable iff the table can be filled by a (non-terminating) algorithmic process

n-th row corresponds to unary function $U_n : x \mapsto U(n,x)$

Computable U is universal if every unary (partial) computable function appears among U_n

Programmer's view

Binary function is an interpreter: $U(n, x)$ is the output of n-th program on input x

Programmer's view

Binary function is an interpreter: $U(n, x)$ is the output of n-th program on input x

Technical: usually the program n, the input x, and the output $U(n, x)$ are binary strings; this does not matter

Programmer's view

Binary function is an interpreter: $U(n, x)$ is the output of n-th program on input x

Technical: usually the program n, the input x, and the output $U(n, x)$ are binary strings; this does not matter

U_n is the function computed by n-th program

Programmer's view

Binary function is an interpreter: $U(n, x)$ is the output of n-th program on input x

Technical: usually the program n, the input x, and the output $U(n, x)$ are binary strings; this does not matter

U_n is the function computed by n-th program

Each programming language has its interpreter

Programmer's view

Binary function is an interpreter: $U(n, x)$ is the output of n-th program on input x

Technical: usually the program n, the input x, and the output $U(n, x)$ are binary strings; this does not matter

U_n is the function computed by n-th program

Each programming language has its interpreter

Programming language is universal if every (computable) behavior can be implemented by some program

Programmer's view

Binary function is an interpreter: $U(n, x)$ is the output of n-th program on input x

Technical: usually the program n, the input x, and the output $U(n, x)$ are binary strings; this does not matter

U_n is the function computed by n-th program

Each programming language has its interpreter

Programming language is universal if every (computable) behavior can be implemented by some program

Probably the most practical discovery of XXth century

Unique numberings

A universal function provides a unique numbering if all U_n are different

Unique numberings

A universal function provides a unique numbering if all U_n are different

Each computable (partial) function appears exactly once among U_n

Unique numberings

A universal function provides a unique numbering if all U_n are different

Each computable (partial) function appears exactly once among U_n

A bizarre programming language that allows only one program for any programming task

Unique numberings

A universal function provides a unique numbering if all U_n are different

Each computable (partial) function appears exactly once among U_n

A bizarre programming language that allows only one program for any programming task

Does such a thing exist?

Unique numberings

A universal function provides a unique numbering if all U_n are different

Each computable (partial) function appears exactly once among U_n

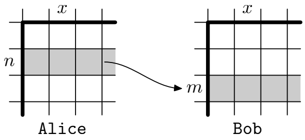
A bizarre programming language that allows only one program for any programming task

Does such a thing exist?

Imagine that we do not know the answer. Even then, we can easily reduce the question to the other one: who wins in some game

The game

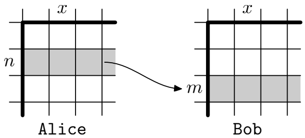
Two players, Alice and Bob, fill two tables (each player has its own one)



The game

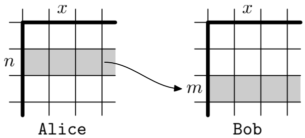
Two players, Alice and Bob, fill two tables (each player has its own one)

Each player sees both but can change only her/his own one



The game

Two players, Alice and Bob, fill two tables (each player has its own one)

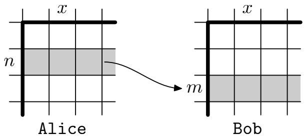


Each player sees both but can change only her/his own one

Alice and Bob alternate; at each step finite number of cells can be filled with natural numbers

The game

Two players, Alice and Bob, fill two tables (each player has its own one)

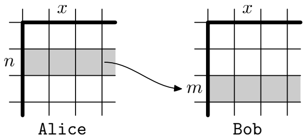


Each player sees both but can change only her/his own one

Alice and Bob alternate; at each step finite number of cells can be filled with natural numbers

Numbers already in the tables stay there forever

The game



Two players, Alice and Bob, fill two tables (each player has its own one)

Each player sees both but can change only her/his own one

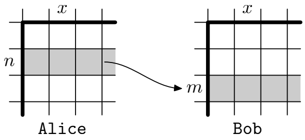
Alice and Bob alternate; at each step finite number of cells can be filled with natural numbers

Numbers already in the tables stay there forever

Game is infinite; the winner is determined by the limit position

The game

Two players, Alice and Bob, fill two tables (each player has its own one)



Each player sees both but can change only her/his own one

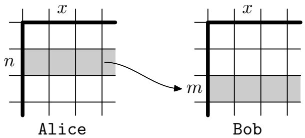
Alice and Bob alternate; at each step finite number of cells can be filled with natural numbers

Numbers already in the tables stay there forever

Game is infinite; the winner is determined by the limit position

Bob wins if: (1) each A-row equals some B-row;

The game



Two players, Alice and Bob, fill two tables (each player has its own one)

Each player sees both but can change only her/his own one

Alice and Bob alternate; at each step finite number of cells can be filled with natural numbers

Numbers already in the tables stay there forever

Game is infinite; the winner is determined by the limit position

Bob wins if: (1) each A-row equals some B-row;
(2) all B-rows are different

Why the game is relevant?

If Alice can win by some computable strategy, unique numberings do not exist

Why the game is relevant?

If Alice can win by some computable strategy, unique numberings do not exist

Let Alice play against Bob that ignores her and (step by step) fills B-table with a universal function that provides a unique numbering.

Why the game is relevant?

If Alice can win by some computable strategy, unique numberings do not exist

Let Alice play against Bob that ignores her and (step by step) fills B-table with a universal function that provides a unique numbering.

Alice actions are computable, so her function is computable, and she loses.

Why the game is relevant?

If Alice can win by some computable strategy, unique numberings do not exist

Let Alice play against Bob that ignores her and (step by step) fills B-table with a universal function that provides a unique numbering.

Alice actions are computable, so her function is computable, and she loses.

If Bob can win by some computable strategy, unique numberings do exist

Why the game is relevant?

If Alice can win by some computable strategy, unique numberings do not exist

Let Alice play against Bob that ignores her and (step by step) fills B-table with a universal function that provides a unique numbering.

Alice actions are computable, so her function is computable, and she loses.

If Bob can win by some computable strategy, unique numberings do exist

Let Bob play against Alice that ignores him and (step by step) fills A-table with some universal function

Why the game is relevant?

If Alice can win by some computable strategy, unique numberings do not exist

Let Alice play against Bob that ignores her and (step by step) fills B-table with a universal function that provides a unique numbering.

Alice actions are computable, so her function is computable, and she loses.

If Bob can win by some computable strategy, unique numberings do exist

Let Bob play against Alice that ignores him and (step by step) fills A-table with some universal function

Bob actions are computable; every computable function is a row in A-table (may be, several rows) and therefore is a row in B-table; all B-rows are different

Who has a winning strategy?

Digression: Borel determinacy theorem (Martin) guarantees the existence of a winning strategy for one of the players

Who has a winning strategy?

Digression: Borel determinacy theorem (Martin) guarantees the existence of a winning strategy for one of the players

But it may be non-computable. Then we get the answer only for universal computable functions with oracle A that is strong enough to compute the strategy.

Who has a winning strategy?

Digression: Borel determinacy theorem (Martin) guarantees the existence of a winning strategy for one of the players

But it may be non-computable. Then we get the answer only for universal computable functions with oracle A that is strong enough to compute the strategy.

Such a strange situation does not occur here

Who has a winning strategy?

Digression: Borel determinacy theorem (Martin) guarantees the existence of a winning strategy for one of the players

But it may be non-computable. Then we get the answer only for universal computable functions with oracle A that is strong enough to compute the strategy.

Such a strange situation does not occur here

R.M.Friedberg: Bob has a winning strategy (1958)

Who has a winning strategy?

Digression: Borel determinacy theorem (Martin) guarantees the existence of a winning strategy for one of the players

But it may be non-computable. Then we get the answer only for universal computable functions with oracle A that is strong enough to compute the strategy.

Such a strange situation does not occur here

R.M.Friedberg: Bob has a winning strategy (1958)

In fact, Friedberg just proved the existence of unique numberings

Bob’s strategy

Bob hires countably many “assistants” $C_1, C_2, \ldots$

Bob's strategy

Bob hires countably many "assistants" $C_1, C_2, \ldots$

Each C_i is "responsible" for i-th row in A-table

Bob's strategy

Bob hires countably many "assistants" $C_1, C_2, \ldots$

Each C_i is "responsible" for i-th row in A-table

The mission of C_i: if i-th row in A is (in the limit) different from all preceding rows, ensure that it also appears somewhere in B-table

Bob's strategy

Bob hires countably many "assistants" $C_1, C_2, \ldots$

Each C_i is "responsible" for i-th row in A-table

The mission of C_i: if i-th row in A is (in the limit) different from all preceding rows, ensure that it also appears somewhere in B-table

This guarantees that every A-row is present in B-table since it appears somewhere in A-table for the first time

Bob's strategy

Bob hires countably many "assistants" $C_1, C_2, \ldots$

Each C_i is "responsible" for i-th row in A-table

The mission of C_i: if i-th row in A is (in the limit) different from all preceding rows, ensure that it also appears somewhere in B-table

This guarantees that every A-row is present in B-table since it appears somewhere in A-table for the first time

All C_i work with the same B-table. They are hired sequentially, so at each step only finitely many C_i are active, and each fills finitely many cells

Bob's strategy

Bob hires countably many "assistants" $C_1, C_2, \ldots$

Each C_i is "responsible" for i-th row in A-table

The mission of C_i: if i-th row in A is (in the limit) different from all preceding rows, ensure that it also appears somewhere in B-table

This guarantees that every A-row is present in B-table since it appears somewhere in A-table for the first time

All C_i work with the same B-table. They are hired sequentially, so at each step only finitely many C_i are active, and each fills finitely many cells

To avoid interference, each C_i first should "reserve" a row in B-table (empty at that moment, and not reserved by anybody else), and work with it; others do not touch this row

So what?

Naïve picture: each assistant C_i, when hired, selects and reserves a row in B-table that is not reserved yet (by others), and then faithfully copies i-th row in A-table into this reserved row.

So what?

Naïve picture: each assistant C_i, when hired, selects and reserves a row in B-table that is not reserved yet (by others), and then faithfully copies i-th row in A-table into this reserved row.

Good news: together they will guarantee that all A-rows appear in B-table

So what?

Naïve picture: each assistant C_i, when hired, selects and reserves a row in B-table that is not reserved yet (by others), and then faithfully copies i-th row in A-table into this reserved row.

Good news: together they will guarantee that all A-rows appear in B-table

Bad news: the process is trivial and useless: C_i just reserves row i and copies i-th row of A-table there, so B-table equals A-table

So what?

Naïve picture: each assistant C_i, when hired, selects and reserves a row in B-table that is not reserved yet (by others), and then faithfully copies i-th row in A-table into this reserved row.

Good news: together they will guarantee that all A-rows appear in B-table

Bad news: the process is trivial and useless: C_i just reserves row i and copies i-th row of A-table there, so B-table equals A-table

We need to ensure that all rows in B-table are different

So what?

Naïve picture: each assistant C_i, when hired, selects and reserves a row in B-table that is not reserved yet (by others), and then faithfully copies i-th row in A-table into this reserved row.

Good news: together they will guarantee that all A-rows appear in B-table

Bad news: the process is trivial and useless: C_i just reserves row i and copies i-th row of A-table there, so B-table equals A-table

We need to ensure that all rows in B-table are different

Difficulty: the condition "i-th A-row differs from the previous ones" cannot be checked at any finite step

Simplified game

More power for Bob: he can “kill” rows in B-table

Simplified game

More power for Bob: he can “kill” rows in B-table

Killed row is never touched again and ignored when the winner is determined

Simplified game

More power for Bob: he can "kill" rows in B-table

Killed row is never touched again and ignored when the winner is determined

Winning condition for the limit state: Bob wins if (1) every A-row appears as a live B-row, and (2) all live B-rows are different

Simplified game

More power for Bob: he can "kill" rows in B-table

Killed row is never touched again and ignored when the winner is determined

Winning condition for the limit state: Bob wins if (1) every A-row appears as a live B-row, and (2) all live B-rows are different

Strategy: each assistant C_i

Simplified game

More power for Bob: he can "kill" rows in B-table

Killed row is never touched again and ignored when the winner is determined

Winning condition for the limit state: Bob wins if (1) every A-row appears as a live B-row, and (2) all live B-rows are different

Strategy: each assistant C_i

- keeps a counter N_i: how many B-rows he has killed

Simplified game

More power for Bob: he can "kill" rows in B-table

Killed row is never touched again and ignored when the winner is determined

Winning condition for the limit state: Bob wins if (1) every A-row appears as a live B-row, and (2) all live B-rows are different

Strategy: each assistant C_i

- keeps a counter N_i: how many B-rows he has killed
- has some B-row reserved by him (the first free one at the beginning or when the previous one is killed)

Simplified game

More power for Bob: he can "kill" rows in B-table

Killed row is never touched again and ignored when the winner is determined

Winning condition for the limit state: Bob wins if (1) every A-row appears as a live B-row, and (2) all live B-rows are different

Strategy: each assistant C_i

- keeps a counter N_i: how many B-rows he has killed
- has some B-row reserved by him (the first free one at the beginning or when the previous one is killed)
- if i-th A-row differs from all previous A-rows if we look at places $1 \ldots N_i$ only, copies its content into the reserved row, otherwise kills it (increasing N_i), chooses a new reserved row and copies i-th A-row there

Why the strategy works

There are two possibilities for C_i

Why the strategy works

There are two possibilities for C_i

- either he changes the reserved row infinitely many times, killing the previous one again and again ($N_i \to \infty$); in this case no trace among the live rows of limit B-table is left;

Why the strategy works

There are two possibilities for C_i

- either he changes the reserved row infinitely many times, killing the previous one again and again ($N_i \to \infty$); in this case no trace among the live rows of limit B-table is left;
- or starting from some moment, C_i never kills his reserved row and just copies the contents of i-th row of A into it; then i-th row of A is among the live rows in B-table.

Why the strategy works

There are two possibilities for C_i

- either he changes the reserved row infinitely many times, killing the previous one again and again ($N_i \to \infty$); in this case no trace among the live rows of limit B-table is left;
- or starting from some moment, C_i never kills his reserved row and just copies the contents of i-th row of A into it; then i-th row of A is among the live rows in B-table.

We need to prove: the second case happens iff i-th row in the limit A-table differs from all the previous rows in it. (Then only the first occurence of each A-row will be copied into B-table, and all live B-rows are different.)

Why the strategy works (continued)

We should prove that two things are not possible:

Why the strategy works (continued)

We should prove that two things are not possible:

- *i-th row is different from all previous rows in the limit A-table, but $N_i \to \infty$.*

Why the strategy works (continued)

We should prove that two things are not possible:

- i-th row is different from all previous rows in the limit A-table, but $N_i \to \infty$. Indeed, look at all places where i-th row differs from the preceding ones. Let N be an upper bound for them. Wait until first i rows stabilize at first N places and until N_i exceeds N. Then S_i sees that i-th row is different from preceding ones in one of the first N_i places. Why would he kill the reserved row and increase N_i?

Why the strategy works (continued)

We should prove that two things are not possible:

- i-th row is different from all previous rows in the limit A-table, but $N_i \to \infty$. Indeed, look at all places where i-th row differs from the preceding ones. Let N be an upper bound for them. Wait until first i rows stabilize at first N places and until N_i exceeds N. Then S_i sees that i-th row is different from preceding ones in one of the first N_i places. Why would he kill the reserved row and increase N_i?
- N_i has finite limit value, but i-th row appears earlier in the limit A-table.

Why the strategy works (continued)

We should prove that two things are not possible:

- i-th row is different from all previous rows in the limit A-table, but $N_i \to \infty$. Indeed, look at all places where i-th row differs from the preceding ones. Let N be an upper bound for them. Wait until first i rows stabilize at first N places and until N_i exceeds N. Then S_i sees that i-th row is different from preceding ones in one of the first N_i places. Why would he kill the reserved row and increase N_i?
- N_i has finite limit value, but i-th row appears earlier in the limit A-table. Let $N = \lim N_i$; wait until rows $1 \ldots i$ stabilize up to position N, and N_i reaches N. Then S_i sees the coincidence, why doesn't he increase N_i?

How to win without killing rows

Some additional tricks needed...

How to win without killing rows

Some additional tricks needed...

First, we agree that each A-row can contain only even number of non-empty cells, so cells are filled in pairs (we postpone adding a number until another one arrives).

How to win without killing rows

Some additional tricks needed...

First, we agree that each A-row can contain only even number of non-empty cells, so cells are filled in pairs (we postpone adding a number until another one arrives).

This may destroy the winning condition for “odd” A-rows (with finite odd number of non-empty cells = functions with finite domain of odd cardinality). We add all such functions (they can be easily enumerated) to B-table one by one: a special assistant looks for the first missing one and adds it in a fresh row, then again, etc. No clashes with copied rows.

How to win without killing rows

Some additional tricks needed...

First, we agree that each A-row can contain only even number of non-empty cells, so cells are filled in pairs (we postpone adding a number until another one arrives).

This may destroy the winning condition for "odd" A-rows (with finite odd number of non-empty cells = functions with finite domain of odd cardinality). We add all such functions (they can be easily enumerated) to B-table one by one: a special assistant looks for the first missing one and adds it in a fresh row, then again, etc. No clashes with copied rows.

Finally, instead of killing a row, we can fill some other cells in it to get an odd row that has not appeared yet, with the same result. (This should be coordinated with the previous step, so no row is added twice.)

What is wrong with unique numberings?

What is wrong with unique numberings?

- Are the universal functions (programming languages) with uniqueness property useful?

What is wrong with unique numberings?

- Are the universal functions (programming languages) with uniqueness property useful?
- No. The problem is that "s-m-n theorem" does not work for them

What is wrong with unique numberings?

- Are the universal functions (programming languages) with uniqueness property useful?
- No. The problem is that "*s*-*m*-*n* theorem" does not work for them
- In programming terms: there is no compiler from a normal language to such a strange one

What is wrong with unique numberings?

- Are the universal functions (programming languages) with uniqueness property useful?
- No. The problem is that "*s*-*m*-*n* theorem" does not work for them
- In programming terms: there is no compiler from a normal language to such a strange one
- (Gödel property for a universal function) for every function $V(m,x)$ there exists a total computable c (compiler) such that $V(m,x) = U(c(m),x)$ for all m and x (both sides are defined or undefined at the same time, and equal if defined)

What is wrong with unique numberings?

- Are the universal functions (programming languages) with uniqueness property useful?
- No. The problem is that "*s*-*m*-*n* theorem" does not work for them
- In programming terms: there is no compiler from a normal language to such a strange one
- (Gödel property for a universal function) for every function $V(m,x)$ there exists a total computable c (compiler) such that $V(m,x) = U(c(m),x)$ for all m and x (both sides are defined or undefined at the same time, and equal if defined)
- Another classical result: all universal functions with Gödel property are isomorphic (and do not have the uniqueness property).

2: conditional and total conditional complexity

2: conditional and total conditional complexity

- Kolmogorov complexity of x is a length of a minimal program (without input) that produces x
- Depends on the programming language
- There is an optimal one that makes complexity minimal (up to $O(1)$ additive term)
- Fix an optimal one and denote complexity by $C(x)$
- Defined up to $O(1)$ additive term

2: conditional and total conditional complexity

- Kolmogorov complexity of x is a length of a minimal program (without input) that produces x
- Depends on the programming language
- There is an optimal one that makes complexity minimal (up to $O(1)$ additive term)
- Fix an optimal one and denote complexity by $C(x)$
- Defined up to $O(1)$ additive term
- Conditional complexity of x given y: the minimal length of a program that transforms input y to output x (for an optimal language). Notation: $C(x|y)$

2: conditional and total conditional complexity

- Kolmogorov complexity of x is a length of a minimal program (without input) that produces x
- Depends on the programming language
- There is an optimal one that makes complexity minimal (up to $O(1)$ additive term)
- Fix an optimal one and denote complexity by $C(x)$
- Defined up to $O(1)$ additive term
- Conditional complexity of x given y: the minimal length of a program that transforms input y to output x (for an optimal language). Notation: $C(x|y)$
- Let us replace minimal length by minimal complexity; then any language is OK (optimality not needed).

2: conditional and total conditional complexity

- Kolmogorov complexity of x is a length of a minimal program (without input) that produces x
- Depends on the programming language
- There is an optimal one that makes complexity minimal (up to $O(1)$ additive term)
- Fix an optimal one and denote complexity by $C(x)$
- Defined up to $O(1)$ additive term
- Conditional complexity of x given y: the minimal length of a program that transforms input y to output x (for an optimal language). Notation: $C(x|y)$
- Let us replace minimal length by minimal complexity; then any language is OK (optimality not needed).
- Total conditional complexity: minimal complexity of a total program that transforms y to x; notation $CT(x|y)$

Do they differ?

Do they differ?

- Evidently, conditional complexity does not exceed total conditional complexity

Do they differ?

- Evidently, conditional complexity does not exceed total conditional complexity
- How big the difference could be?

Do they differ?

- Evidently, conditional complexity does not exceed total conditional complexity
- How big the difference could be?
- The difference could be maximal: for a given n there exist strings x and y of length n such that $C(x|y) = O(1)$ but $CT(x|y) \geq n$. (Two extreme cases.)

Do they differ?

- Evidently, conditional complexity does not exceed total conditional complexity
- How big the difference could be?
- The difference could be maximal: for a given n there exist strings x and y of length n such that $C(x|y) = O(1)$ but $CT(x|y) \geq n$. (Two extreme cases.)

(Digression) CT attracted attention recently (Bauwens, Vereshchagin). If information distance between x and y is small ($C(x|y) \approx 0$ and $C(y|x) \approx 0$), still x and y can have different properties. But if $CT(x|y) \approx 0$ and $CT(y|x) \approx 0$, then x is mapped to y by a simple computable permutation, so x and y are very similar.

The game

The game

- Two sets X and Y of 2^n elements, e.g., n-bit strings

The game

- Two sets X and Y of 2^n elements, e.g., n-bit strings
- Alice and Bob alternate and see each other's moves

The game

- Two sets X and Y of 2^n elements, e.g., n-bit strings
- Alice and Bob alternate and see each other's moves
- Alice constructs a partial function $a\colon Y \to X$; at each move she may define several (may be, zero) values of a (but cannot change already defined values)

The game

- Two sets X and Y of 2^n elements, e.g., n-bit strings
- Alice and Bob alternate and see each other's moves
- Alice constructs a partial function $a\colon Y \to X$; at each move she may define several (may be, zero) values of a (but cannot change already defined values)
- Bob constructs total functions $b_1, b_2, \ldots : Y \to X$; at each step several functions may be added, but the length of the list should be less than 2^n.

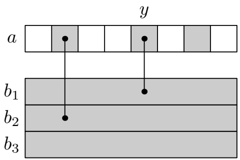
The game

- Two sets X and Y of 2^n elements, e.g., n-bit strings
- Alice and Bob alternate and see each other's moves
- Alice constructs a partial function $a\colon Y \to X$; at each move she may define several (may be, zero) values of a (but cannot change already defined values)
- Bob constructs total functions $b_1, b_2, \ldots : Y \to X$; at each step several functions may be added, but the length of the list should be less than 2^n.
- Alice wins if in the limit there exists some $y \in Y$ such that $a(y)$ is defined and different from all $b_i(y)$

Game field

Game field

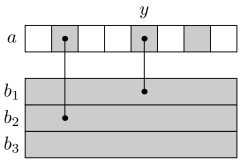
Alice fills the upper row
(with elements of X, step by step)



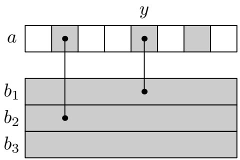
Game field

Alice fills the upper row
(with elements of X, step by step)

Bob adds new (completely filled)
rows to the table, at most $2^n - 1$ rows



Game field

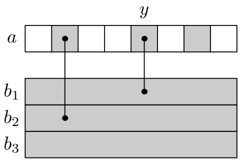


Alice fills the upper row
(with elements of X, step by step)

Bob adds new (completely filled)
rows to the table, at most $2^n - 1$ rows

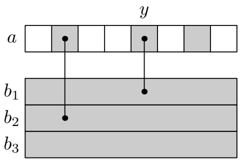
Alice's goal: some element of her row
does not appear again in the B-column under it

Game field



- Alice fills the upper row (with elements of X, step by step)
- Bob adds new (completely filled) rows to the table, at most $2^n - 1$ rows
- Alice's goal: some element of her row does not appear again in the B-column under it
- Bob's goal: every element of A-table appears in one of B-rows in the same column

Game field



Alice fills the upper row
(with elements of X, step by step)

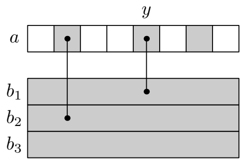
Bob adds new (completely filled)
rows to the table, at most $2^n - 1$ rows

Alice's goal: some element of her row
does not appear again in the B-column under it

Bob's goal: every element of A-table appears in one of
B-rows in the same column

Alice's strategy: choose a free cell, put an element that does not exist in its column yet, and wait until Bob answers with a new row

Game field



Alice fills the upper row
(with elements of *X*, step by step)

Bob adds new (completely filled)
rows to the table, at most $2^n - 1$ rows

Alice's goal: some element of her row
does not appear again in the *B*-column under it

Bob's goal: every element of *A*-table appears in one of
B-rows in the same column

Alice's strategy: choose a free cell, put an element that does not exist in its column yet, and wait until Bob answers with a new row

Alice wins: there are enough elements of *X*, and she has more moves than Bob

Why this is enough

Why this is enough

- X and Y are sets of n-bit strings

Why this is enough

- X and Y are sets of n-bit strings
- Bob ignores Alice and add rows that correspond to total algoritms of complexity less than n (restricted to Y, assuming that all the values are in X); there are less than 2^n of them

Why this is enough

- X and Y are sets of n-bit strings
- Bob ignores Alice and add rows that correspond to total algoritms of complexity less than n (restricted to Y, assuming that all the values are in X); there are less than 2^n of them
- Alice uses her strategy

Why this is enough

- X and Y are sets of n-bit strings
- Bob ignores Alice and add rows that correspond to total algoritms of complexity less than n (restricted to Y, assuming that all the values are in X); there are less than 2^n of them
- Alice uses her strategy
- y is an element where Alice wins, and x is $a(y)$

Why this is enough

- X and Y are sets of n-bit strings
- Bob ignores Alice and add rows that correspond to total algoritms of complexity less than n (restricted to Y, assuming that all the values are in X); there are less than 2^n of them
- Alice uses her strategy
- y is an element where Alice wins, and x is $a(y)$
- since Alice process is computable given n, and n is determined by y, we guarantee that $C(x|y) = O(1)$

Why this is enough

- X and Y are sets of n-bit strings
- Bob ignores Alice and add rows that correspond to total algoritms of complexity less than n (restricted to Y, assuming that all the values are in X); there are less than 2^n of them
- Alice uses her strategy
- y is an element where Alice wins, and x is $a(y)$
- since Alice process is computable given n, and n is determined by y, we guarantee that $C(x|y) = O(1)$
- by construction $CT(x|y) \geq n$

3: Muchnik–Vyugin's result

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how “far” is x from information that exists in y

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different
- if we need one number as a distance, we can consider $C(x|y) + C(y|x)$, but pair $C(x|y), C(y|x)$ is more informative

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different
- if we need one number as a distance, we can consider $C(x|y) + C(y|x)$, but pair $C(x|y), C(y|x)$ is more informative
- Question: for a given x and n, can we find y such that both $C(x|y)$ and $C(y|x)$ are $n + O(1)$?

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different
- if we need one number as a distance, we can consider $C(x|y) + C(y|x)$, but pair $C(x|y), C(y|x)$ is more informative
- Question: for a given x and n, can we find y such that both $C(x|y)$ and $C(y|x)$ are $n + O(1)$?
- Some conditions needed: if $C(x) \ll n$, then $C(x|y) \ll n$.

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different
- if we need one number as a distance, we can consider $C(x|y) + C(y|x)$, but pair $C(x|y), C(y|x)$ is more informative
- Question: for a given x and n, can we find y such that both $C(x|y)$ and $C(y|x)$ are $n + O(1)$?
- Some conditions needed: if $C(x) \ll n$, then $C(x|y) \ll n$.
- (M. Vyugin) if $C(x) > 2n$, then such a y exists

3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different
- if we need one number as a distance, we can consider $C(x|y) + C(y|x)$, but pair $C(x|y), C(y|x)$ is more informative
- Question: for a given x and n, can we find y such that both $C(x|y)$ and $C(y|x)$ are $n + O(1)$?
- Some conditions needed: if $C(x) \ll n$, then $C(x|y) \ll n$.
- (M. Vyugin) if $C(x) > 2n$, then such a y exists
- with $O(\log n)$ instead of $O(1)$ this is easy: take the shortest program for x, and replace n first bits in it by random ones

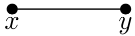
3: Muchnik–Vyugin's result

- $C(x|y)$ measures how "far" is x from information that exists in y
- not symmetric: $C(x|y)$ and $C(y|x)$ can be very different
- if we need one number as a distance, we can consider $C(x|y) + C(y|x)$, but pair $C(x|y), C(y|x)$ is more informative
- Question: for a given x and n, can we find y such that both $C(x|y)$ and $C(y|x)$ are $n + O(1)$?
- Some conditions needed: if $C(x) \ll n$, then $C(x|y) \ll n$.
- (M. Vyugin) if $C(x) > 2n$, then such a y exists
- with $O(\log n)$ instead of $O(1)$ this is easy: take the shortest program for x, and replace n first bits in it by random ones
- topological argument replaces $2n$ by $n + O(\log n)$.

More strings

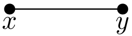
More strings

We constructed for a given x some y that has “distance” n in both directions

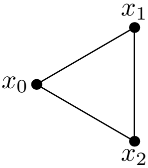


More strings

We constructed for a given x some y that has "distance" n in both directions

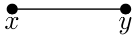


Now for a given x_0 we want to find a "right triangle" $x_0x_1x_2$ where all six "distances" are $n + O(1)$

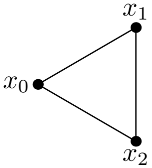


More strings

We constructed for a given x some y that has "distance" n in both directions



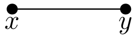
Now for a given x_0 we want to find a "right triangle" $x_0x_1x_2$ where all six "distances" are $n + O(1)$



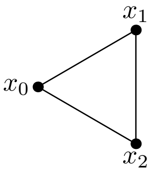
Muchnik and Vyugin has shown that this is possible if $C(x) > cn$ for large enough c (that does not depend on x and n)

More strings

We constructed for a given x some y that has "distance" n in both directions



Now for a given x_0 we want to find a "right triangle" $x_0x_1x_2$ where all six "distances" are $n + O(1)$

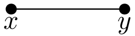


Muchnik and Vyugin has shown that this is possible if $C(x) > cn$ for large enough c (that does not depend on x and n)

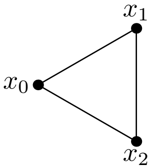
Similar statement is true for $4, 5, \ldots$ strings.

More strings

We constructed for a given x some y that has "distance" n in both directions



Now for a given x_0 we want to find a "right triangle" $x_0x_1x_2$ where all six "distances" are $n + O(1)$



Muchnik and Vyugin has shown that this is possible if $C(x) > cn$ for large enough c (that does not depend on x and n)

Similar statement is true for $4, 5, \ldots$ strings.

Game proof: the game is straightforward, but the strategy is quite complicated

Andrej Muchnik (1958-2007)