Bounded Queries in Recursion Theory

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Consider the following problem:

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Consider the following problem:
Input You are given three programs e_1, e_2, e_3.

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Consider the following problem:
Input You are given three programs e_1, e_2, e_3.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Consider the following problem:

Input You are given three programs e_1, e_2, e_3.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

(Output is one of 000, 001, 010, 011, 100, 101, 110, 111.)

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Consider the following problem:
Input You are given three programs e_1, e_2, e_3.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

(Output is one of 000, 001, 010, 011, 100, 101, 110, 111.)
Not computable since HALT is not computable.

Asking about Three Programs

Notation $\mathrm{HALT}(e)$ is 1 if $e \in \mathrm{HALT}$ and 0 otherwise.

Consider the following problem:
Input You are given three programs e_1, e_2, e_3.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

(Output is one of 000, 001, 010, 011, 100, 101, 110, 111.)
Not computable since HALT is not computable.

But What if... See next slide.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if you are only allowed **2** queries to HALT?

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if you are only allowed **2** queries to HALT?
VOTE

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if you are only allowed **2** queries to HALT?
VOTE

- **Known** cannot solve with 2 queries.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if you are only allowed **2** queries to HALT?
VOTE

- **Known** cannot solve with 2 queries.
- **Known** can solve with 2 queries.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if you are only allowed **2** queries to HALT?
VOTE

- **Known** cannot solve with 2 queries.
- **Known** can solve with 2 queries.
- **Unknown to Science**.

What if You Could Make Queries to HALT?

Input You are given three programs e_1, e_2, e_3.
Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)\mathrm{HALT}(e_3)$.

We will allow queries to HALT.

If could make **3** queries to HALT then you could solve.

What if you are only allowed **2** queries to HALT?
VOTE

- **Known** cannot solve with 2 queries.
- **Known** can solve with 2 queries.
- **Unknown to Science**.

Answer on next slide.

Known Can Solve With 2 Queries

We will need the following notation.

Known Can Solve With 2 Queries

We will need the following notation.

Notation Let e_1, e_2, e_3 be programs. $A(i)$ is the program that runs all of them at the same time until i of them halt.

Known Can Solve With 2 Queries

We will need the following notation.

Notation Let e_1, e_2, e_3 be programs. $A(i)$ is the program that runs all of them at the same time until i of them halt.

$A(i) \in \mathrm{HALT}$ iff at least i of the programs are in HALT.

Known Can Solve With 2 Queries

We will need the following notation.

Notation Let e_1, e_2, e_3 be programs. $A(i)$ is the program that runs all of them at the same time until i of them halt.

$A(i) \in \mathrm{HALT}$ iff at least i of the programs are in HALT.

Key **Do $\ge i$ of $e_1, e_2, e_3 \in \mathrm{HALT}$** is a query to HALT.

Known Can Solve With 2 Queries

We will need the following notation.

Notation Let e_1, e_2, e_3 be programs. $A(i)$ is the program that runs all of them at the same time until i of them halt.

$A(i) \in \mathrm{HALT}$ iff at least i of the programs are in HALT.

Key Do $\geq i$ of $e_1, e_2, e_3 \in \mathrm{HALT}$ is a query to HALT.

We will use $A(i)$ in the algorithm on the next slide.

Known Can Solve With 2 Queries

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?
   2.1 If YES then **Ask** Are ≥ 3 of e_1, e_2, e_3 in HALT?

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?
    2.1 If YES then **Ask** Are ≥ 3 of e_1, e_2, e_3 in HALT?
    If YES then output 111.

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?
   2.1 If YES then **Ask** Are ≥ 3 of e_1, e_2, e_3 in HALT?
   If YES then output 111.
   If NO then **exactly** 2 of e_1, e_2, e_3 are in HALT.

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?
   2.1 If YES then **Ask** Are ≥ 3 of e_1, e_2, e_3 in HALT?
   If YES then output 111.
   If NO then **exactly** 2 of e_1, e_2, e_3 are in HALT.
   What to do? Discuss!

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?
   2.1 If YES then **Ask** Are ≥ 3 of e_1, e_2, e_3 in HALT?
   If YES then output 111.
   If NO then **exactly** 2 of e_1, e_2, e_3 are in HALT.
   What to do? Discuss!
   RUN e_1, e_2, e_3 UNTIL 2 of them halt. When they do, you know exactly which ones halt.

Known Can Solve With 2 Queries

1. Input e_1, e_2, e_3.
2. **Ask** Are ≥ 2 of e_1, e_2, e_3 in HALT?
   2.1 If YES then **Ask** Are ≥ 3 of e_1, e_2, e_3 in HALT?
   If YES then output 111.
   If NO then **exactly** 2 of e_1, e_2, e_3 are in HALT.
   What to do? Discuss!
   RUN e_1, e_2, e_3 UNTIL 2 of them halt. When they do, you know exactly which ones halt.
   2.2 If NO then similar. Find out HOW MANY of e_1, e_2, e_3 are in HALT and then RUN them all to see which ones HALT.

Notes On The Result

1. Konstantine voted **Known** **Cannot be done with 2 queries.** He was right but wrong.

Notes On The Result

1. Konstantine voted **Known** **Cannot be done with 2 queries.** He was right but wrong. Actually wrong but has a point.

Notes On The Result

1. Konstantine voted **Known** **Cannot be done with 2 queries.** He was right but wrong. Actually wrong but has a point.

   Note the following:

Notes On The Result

1. Konstantine voted **Known** **Cannot be done with 2 queries.** He was right but wrong. Actually wrong but has a point.

   Note the following:

   If in the algorithm the **wrong** information was supplied to the questions then the algorithm could $\uparrow$.

Notes On The Result

1. Konstantine voted **Known** **Cannot be done with 2 queries.** He was right but wrong. Actually wrong but has a point.

   Note the following:

   If in the algorithm the **wrong** information was supplied to the questions then the algorithm could $\uparrow$.

   Known If you require the algorithm to halt even with wrong answers, then you need 3 queries.

Notes On The Result

1. Konstantine voted **Known** **Cannot be done with 2 queries.** He was right but wrong. Actually wrong but has a point.

   Note the following:

   If in the algorithm the **wrong** information was supplied to the questions then the algorithm could $\uparrow$.

   Known If you require the algorithm to halt even with wrong answers, then you need 3 queries.

2. I did 3-queries-for-2. We will generalize on next slide.

What if Given n Programs?

Given $e_1, \ldots, e_n$ want to know

$$\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_n).$$

What if Given n Programs?

Given $e_1, \ldots, e_n$ want to know

$$\text{HALT}(e_1) \cdots \text{HALT}(e_n).$$

Work with your neighbor on the question:
Let $n \geq 3$. How many queries to HALT do you need to find $\text{HALT}(e_1) \cdots \text{HALT}(e_n)$?

Here is the Answer

n	No. of q's
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	4
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	5

Here is the Answer

n	No. of q's
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	4
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	5

If $2^i \leq n \leq 2^{i+1} - 1$ then takes $i + 1$ queries.

Here is the Answer

n	No. of q's
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	4
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	5

If $2^i \le n \le 2^{i+1} - 1$ then takes $i + 1$ queries.

Is there a better algorithm? Next slide looks at $n = 2$.

Asking about Two Programs

Consider the following problem:

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

(Output is one of 00, 01, 10, 11)

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

(Output is one of 00, 01, 10, 11)

VOTE

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

(Output is one of 00, 01, 10, 11)

VOTE

- **Known** cannot solve with 1 query.

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

(Output is one of 00, 01, 10, 11)

VOTE

- **Known** cannot solve with 1 query.
- **Known** can solve with 1 query.

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

(Output is one of 00, 01, 10, 11)

VOTE

- **Known** cannot solve with 1 query.
- **Known** can solve with 1 query.
- **Unknown to Science**

Asking about Two Programs

Consider the following problem:

Input You are given two programs e_1, e_2.

Output $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$.

(Output is one of 00, 01, 10, 11)

VOTE

- **Known** cannot solve with 1 query.
- **Known** can solve with 1 query.
- **Unknown to Science**

Answer on next slide.

Known Cannot Solve With 1 Query

Known Cannot Solve With 1 Query

Need a new viewpoint.

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Thm If f can be computed with 1 query to X then $f \in \mathrm{EN}(2)$.

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Thm If f can be computed with 1 query to X then $f \in \mathrm{EN}(2)$.
Proof

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Thm If f can be computed with 1 query to X then $f \in \mathrm{EN}(2)$.
Proof

$M_1(x)$ runs the 1-query Alg for f. Answer query YES.

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Thm If f can be computed with 1 query to X then $f \in \mathrm{EN}(2)$.

Proof

$M_1(x)$ runs the 1-query Alg for f. Answer query YES.

$M_2(x)$ runs the 1-query Alg for f. Answer query NO.

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Thm If f can be computed with 1 query to X then $f \in \mathrm{EN}(2)$.

Proof

$M_1(x)$ runs the 1-query Alg for f. Answer query YES.

$M_2(x)$ runs the 1-query Alg for f. Answer query NO.

Since the query asked either has answer Y or answer N, at least one of $M_1(x)$ and $M_2(x)$ will be correct.

Known Cannot Solve With 1 Query

Need a new viewpoint.

Def A function $f \in \mathrm{EN}(2)$ if there exists 2 Turing Machines M_1, M_2 such that

$$(\forall x)[f(x) \in \{M_1(x), M_2(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Thm If f can be computed with 1 query to X then $f \in \mathrm{EN}(2)$.
Proof

$M_1(x)$ runs the 1-query Alg for f. Answer query YES.

$M_2(x)$ runs the 1-query Alg for f. Answer query NO.

Since the query asked either has answer Y or answer N, at least one of $M_1(x)$ and $M_2(x)$ will be correct.

Note that which one is correct may vary. It may be that on $M_1(17) \downarrow = f(17)$ but $M_2(22) \downarrow = f(22)$.

Known Cannot Solve With 1 Query

Notation $(a, b) =_1 (c, d)$ means $a = c$. Sim for $\neq_1$.

Known Cannot Solve With 1 Query

Notation $(a, b) =_1 (c, d)$ means $a = c$. Sim for $\neq_1$.

Assume, BWOC that the function $f(e_1, e_2) = \mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ can be computed with **one** query to HALT.

Known Cannot Solve With 1 Query

Notation $(a, b) =_1 (c, d)$ means $a = c$. Sim for $\neq_1$.

Assume, BWOC that the function $f(e_1, e_2) = \mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ can be computed with **one** query to HALT.

By last slide there are two TM's M_1, M_2 such that $(\forall e_1, e_2)[\mathrm{HALT}(e_1)\mathrm{HALT}(e_2) \in \{M_1(e_1, e_2), M_2(e_1, e_2)\}$.

Known Cannot Solve With 1 Query

Notation $(a, b) =_1 (c, d)$ means $a = c$. Sim for $\neq_1$.

Assume, BWOC that the function $f(e_1, e_2) = \mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ can be computed with **one** query to HALT.

By last slide there are two TM's M_1, M_2 such that $(\forall e_1, e_2)[\mathrm{HALT}(e_1)\mathrm{HALT}(e_2) \in \{M_1(e_1, e_2), M_2(e_1, e_2)\}$.

We will use this to get HALT is decidable.

Known Cannot Solve With 1 Query

Notation $(a, b) =_1 (c, d)$ means $a = c$. Sim for $\neq_1$.

Assume, BWOC that the function $f(e_1, e_2) = \mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ can be computed with **one** query to HALT.

By last slide there are two TM's M_1, M_2 such that $(\forall e_1, e_2)[\mathrm{HALT}(e_1)\mathrm{HALT}(e_2) \in \{M_1(e_1, e_2), M_2(e_1, e_2)\}$.

We will use this to get HALT is decidable.

Two cases. On the next two slides.

Case 1

Motivation We have M_1, M_2 which take **two** inputs

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:
Case 1 $(\forall e_1)(\exists e_2)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge M_1(e_1, e_2) =_1 M_2(e_1, e_2)]$.

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:
Case 1 $(\forall e_1)(\exists e_2)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge M_1(e_1, e_2) =_1 M_2(e_1, e_2)]$.

1. Input e_1

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:
Case 1 $(\forall e_1)(\exists e_2)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge M_1(e_1, e_2) =_1 M_2(e_1, e_2)]$.

1. Input e_1
2. For $(e_2, s) \in \mathbb{N} \times \mathbb{N}$

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:
Case 1 $(\forall e_1)(\exists e_2)$
$[M_1(e_1, e_2) \downarrow \wedge M_2(e_1, e_2) \downarrow \wedge M_1(e_1, e_2) =_1 M_2(e_1, e_2)]$.

1. Input e_1
2. For $(e_2, s) \in \mathbb{N} \times \mathbb{N}$
   2.1 Run $M_1(e_1, e_2)$ and $M_2(e_1, e_2)$ for s steps.

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:
Case 1 $(\forall e_1)(\exists e_2)$
$[M_1(e_1, e_2) \downarrow \wedge M_2(e_1, e_2) \downarrow \wedge M_1(e_1, e_2) =_1 M_2(e_1, e_2)]$.

1. Input e_1
2. For $(e_2, s) \in \mathbb{N} \times \mathbb{N}$
   2.1 Run $M_1(e_1, e_2)$ and $M_2(e_1, e_2)$ for s steps.
   2.2 If they both $\downarrow$ and agree on first spot, output that spot. Else go to next (e_2, s).

Case 1

Motivation We have M_1, M_2 which take **two** inputs
But we want to solve HALT which takes **one** input.
what if we could always find a helpful second input:
Case 1 $(\forall e_1)(\exists e_2)$
$[M_1(e_1, e_2) \downarrow \wedge M_2(e_1, e_2) \downarrow \wedge M_1(e_1, e_2) =_1 M_2(e_1, e_2)]$.

1. Input e_1
2. For $(e_2, s) \in \mathbb{N} \times \mathbb{N}$
   2.1 Run $M_1(e_1, e_2)$ and $M_2(e_1, e_2)$ for s steps.
   2.2 If they both $\downarrow$ and agree on first spot, output that spot. Else go to next (e_2, s).

This algorithm computes HALT because of the case we are in.

Case 2

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2) \downarrow \wedge M_2(e_1, e_2) \downarrow \wedge$ agree on first spot].

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge$ agree on first spot].
Case 2 will be negation of Case 1.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge$ agree on first spot$]$.
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1, e_2)\uparrow \vee M_2(e_1, e_2)\uparrow \vee M_1(e_1, e_2)\downarrow \neq_1 M_2(e_1, e_2)]$.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge$ agree on first spot].
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1, e_2)\uparrow \vee M_2(e_1, e_2)\uparrow \vee M_1(e_1, e_2)\downarrow \neq_1 M_2(e_1, e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2) \downarrow \wedge M_2(e_1, e_2) \downarrow \wedge$ agree on first spot].
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1, e_2) \uparrow \vee M_2(e_1, e_2) \uparrow \vee M_1(e_1, e_2) \downarrow \neq_1 M_2(e_1, e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

1. Input e_2 (Yes I intentionally use e_2.)

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge$ agree on first spot$]$.
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1, e_2)\uparrow \vee M_2(e_1, e_2)\uparrow \vee M_1(e_1, e_2)\downarrow \neq_1 M_2(e_1, e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

1. Input e_2 (Yes I intentionally use e_2.)
2. Run $M_1(e_1, e_2)$ and $M_2(e_1, e_2)$ at the same time until one of them halts and has b as the first component.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2) \downarrow \wedge M_2(e_1, e_2) \downarrow \wedge$ agree on first spot$]$.
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1, e_2) \uparrow \vee M_2(e_1, e_2) \uparrow \vee M_1(e_1, e_2) \downarrow \neq_1 M_2(e_1, e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

1. Input e_2 (Yes I intentionally use e_2.)
2. Run $M_1(e_1, e_2)$ and $M_2(e_1, e_2)$ at the same time until one of them halts and has b as the first component.
   Because of the case we are in, the other one **cannot** halt and have b as the second component.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1,e_2)\downarrow \wedge M_2(e_1,e_2)\downarrow \wedge$ agree on first spot$]$.
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1,e_2)\uparrow \vee M_2(e_1,e_2)\uparrow \vee M_1(e_1,e_2)\downarrow \neq_1 M_2(e_1,e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

1. Input e_2 (Yes I intentionally use e_2.)
2. Run $M_1(e_1,e_2)$ and $M_2(e_1,e_2)$ at the same time until one of them halts and has b as the first component.
   Because of the case we are in, the other one **cannot** halt and have b as the second component.
   Hence this is the correct answer.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1,e_2)\downarrow \wedge M_2(e_1,e_2)\downarrow \wedge$ agree on first spot].
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1,e_2)\uparrow \vee M_2(e_1,e_2)\uparrow \vee M_1(e_1,e_2)\downarrow\neq_1 M_2(e_1,e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

1. Input e_2 (Yes I intentionally use e_2.)
2. Run $M_1(e_1,e_2)$ and $M_2(e_1,e_2)$ at the same time until one of them halts and has b as the first component.
   Because of the case we are in, the other one **cannot** halt and have b as the second component.
   Hence this is the correct answer.
3. Output the second component.

Case 2

Recall: **Case 1** $(\forall e_2)(\exists e_1)$
$[M_1(e_1, e_2)\downarrow \wedge M_2(e_1, e_2)\downarrow \wedge$ agree on first spot].
Case 2 will be negation of Case 1.
Case 2 $(\exists e_1)(\forall e_2)$
$[M_1(e_1, e_2)\uparrow \vee M_2(e_1, e_2)\uparrow \vee M_1(e_1, e_2)\downarrow \neq_1 M_2(e_1, e_2)]$.
We use e_1 and $b = \mathrm{HALT}(e_1)$ as parameters in the algorithm.

1. Input e_2 (Yes I intentionally use e_2.)
2. Run $M_1(e_1, e_2)$ and $M_2(e_1, e_2)$ at the same time until one of them halts and has b as the first component.
   Because of the case we are in, the other one **cannot** halt and have b as the second component.
   Hence this is the correct answer.
3. Output the second component.

This algorithm computes HALT because of the case we are in.

The Proof is Nonconstructive!

Given an alleged algorithm for $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ that makes only one query, the proof **does not** tell you how to create an algorithm for HALT.

The Proof is Nonconstructive!

Given an alleged algorithm for $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ that makes only one query, the proof **does not** tell you how to create an algorithm for HALT.

However, it does tell you how to create an infinite number of programs, one of which solve HALT.

The Proof is Nonconstructive!

Given an alleged algorithm for $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ that makes only one query, the proof **does not** tell you how to create an algorithm for HALT.

However, it does tell you how to create an infinite number of programs, one of which solve HALT.

So the proof is nonconstructive.

The Proof is Nonconstructive!

Given an alleged algorithm for $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ that makes only one query, the proof **does not** tell you how to create an algorithm for HALT.

However, it does tell you how to create an infinite number of programs, one of which solve HALT.

So the proof is nonconstructive.

Could there be a constructive proof? No.

The Proof is Nonconstructive!

Given an alleged algorithm for $\mathrm{HALT}(e_1)\mathrm{HALT}(e_2)$ that makes only one query, the proof **does not** tell you how to create an algorithm for HALT.

However, it does tell you how to create an infinite number of programs, one of which solve HALT.

So the proof is nonconstructive.

Could there be a constructive proof? No.
Proven by Gasarch in 1990.

Summary of What I’ve Told You

Summary of What I've Told You

- 3-queries-to-HALT **can** be computed with 2-queries-to-HALT.

Summary of What I've Told You

- 3-queries-to-HALT **can** be computed with 2-queries-to-HALT.
- 2-queries-to-HALT **cannot** be computed with 1-query-to-X for any X.

Summary of What I've Told You

- 3-queries-to-HALT **can** be computed with 2-queries-to-HALT.
- 2-queries-to-HALT **cannot** be computed with 1-query-to-X for any X.
- **Konstantine's Theorem** CANNOT do 3-queries-to-HALT with 2-queries-to-X if you insist that even incorrect answers lead to converging.

Summary of What I've Told You

- 3-queries-to-HALT **can** be computed with 2-queries-to-HALT.
- 2-queries-to-HALT **cannot** be computed with 1-query-to-X for any X.
- **Konstantine's Theorem** CANNOT do 3-queries-to-HALT with 2-queries-to-X if you insist that even incorrect answers lead to converging.

Hence we know the exact query complexity of 3-queries-to-HALT.

What More is Known

What More is Known

- $2^n - 1$-queries-to-HALT **can** be computed with n-queries-to-HALT.

What More is Known

- $2^n - 1$-queries-to-HALT **can** be computed with n-queries-to-HALT.
  Use Binary Search to find out how many halt and then run them to see which ones halt.

What More is Known

- $2^n - 1$-queries-to-HALT **can** be computed with n-queries-to-HALT.
  Use Binary Search to find out how many halt and then run them to see which ones halt.
- 2^n-queries-to-HALT **cannot** be computed with n-queries-to-X.

What More is Known

- $2^n - 1$-queries-to-HALT **can** be computed with n-queries-to-HALT.
  Use Binary Search to find out how many halt and then run them to see which ones halt.
- 2^n-queries-to-HALT **cannot** be computed with n-queries-to-X.
  Could do this in class if had more time.

What More is Known

- $2^n - 1$-queries-to-HALT **can** be computed with n-queries-to-HALT.
  Use Binary Search to find out how many halt and then run them to see which ones halt.
- 2^n-queries-to-HALT **cannot** be computed with n-queries-to-X.
  Could do this in class if had more time.
- **Konstantine's Theorem** If you want to compute m-queries to HALT and you insist that even incorrect answers lead to converging then **requires** m queries.

First Step in Proof about 2^n

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.
Proof For $\tau \in \{0,1\}^n$ let $M^\tau(x)$ do the computation of f but answer the ith query with σ_i.

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.

Proof For $\tau \in \{0,1\}^n$ let $M^\tau(x)$ do the computation of f but answer the ith query with σ_i.

Thm $\mathrm{HALT}(e_1)\cdots\mathrm{HALT}(e_{2^n})$ cannot be computed with n queries.

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.
Proof For $\tau \in \{0,1\}^n$ let $M^\tau(x)$ do the computation of f but answer the ith query with σ_i.
Thm $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n})$ cannot be computed with n queries.
Beginning of Proof
Assume, BWOC that $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n})$ can be computed with n queries. By Lemma $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n}) \in \mathrm{EN}(2^n)$.

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.
Proof For $\tau \in \{0,1\}^n$ let $M^\tau(x)$ do the computation of f but answer the ith query with σ_i.
Thm $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n})$ cannot be computed with n queries.
Beginning of Proof
Assume, BWOC that $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n})$ can be computed with n queries. By Lemma $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n}) \in \mathrm{EN}(2^n)$.
OH! Lets CHANGE the problem

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.
Proof For $\tau \in \{0,1\}^n$ let $M^\tau(x)$ do the computation of f but answer the ith query with σ_i.
Thm $\mathrm{HALT}(e_1)\cdots\mathrm{HALT}(e_{2^n})$ cannot be computed with n queries.
Beginning of Proof
Assume, BWOC that $\mathrm{HALT}(e_1)\cdots\mathrm{HALT}(e_{2^n})$ can be computed with n queries. By Lemma $\mathrm{HALT}(e_1)\cdots\mathrm{HALT}(e_{2^n}) \in \mathrm{EN}(2^n)$.
OH! Lets CHANGE the problem
Thm For all m, $\mathrm{HALT}(e_1), \cdots, \mathrm{HALT}(e_m)$ is notin $\mathrm{EN}(m)$.

First Step in Proof about 2^n

Def A function $f \in \mathrm{EN}(m)$ if there exists m Turing Machines $M_1, \ldots, M_m$ such that

$$(\forall x)[f(x) \in \{M_1(x), \ldots, M_m(x)\}].$$

(So at least one of the TM's halts and outputs the right answer.)

Lemma If f can be computed with n q's to X then $f \in \mathrm{EN}(2^n)$.
Proof For $\tau \in \{0,1\}^n$ let $M^\tau(x)$ do the computation of f but answer the ith query with σ_i.
Thm $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n})$ cannot be computed with n queries.
Beginning of Proof
Assume, BWOC that $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n})$ can be computed with n queries. By Lemma $\mathrm{HALT}(e_1) \cdots \mathrm{HALT}(e_{2^n}) \in \mathrm{EN}(2^n)$.
OH! Lets CHANGE the problem
Thm For all m, $\mathrm{HALT}(e_1), \cdots, \mathrm{HALT}(e_m)$ is notin $\mathrm{EN}(m)$.
The proof is by induction on m. Omitted but could do.

More Has Been Studied

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1) \cdots Ae_m) \notin \mathrm{EN}(m)]$.

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT.

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.
   $\mathrm{INF}(e_1)\cdots\mathrm{INF}(e_n)$ requires n queries.

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.
   $\mathrm{INF}(e_1)\cdots\mathrm{INF}(e_n)$ requires n queries.
4. Number-of-q's is a complexity measure.

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.
   $\mathrm{INF}(e_1)\cdots\mathrm{INF}(e_n)$ requires n queries.
4. Number-of-q's is a complexity measure.
   Example

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.
   $\mathrm{INF}(e_1)\cdots\mathrm{INF}(e_n)$ requires n queries.
4. Number-of-q's is a complexity measure.
   Example How many queries does it take to find the chromatic number of an infinite graph?

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.
   $\mathrm{INF}(e_1)\cdots\mathrm{INF}(e_n)$ requires n queries.
4. Number-of-q's is a complexity measure.
   Example How many queries does it take to find the chromatic number of an infinite graph?
5. q's-to-SAT in Poly Time has been studied. Some results similar. But the following is different:

More Has Been Studied

Thm Let A be undec. $(\forall m)[A(e_1)\cdots Ae_m) \notin \mathrm{EN}(m)]$.
The following have been studied:

1. Parallel q's. Our 3-for-2 Alg was sequential.
2. Algs where all query-paths $\downarrow$ (Konstantine's Issue).
3. Sets other than HALT. **Example**
   $\mathrm{INF} = \{e : (\forall x)(\exists y, s)[M_{e,s)\downarrow}\}$ is Π_2-complete.
   $\mathrm{INF}(e_1)\cdots\mathrm{INF}(e_n)$ requires n queries.
4. Number-of-q's is a complexity measure.
   Example How many queries does it take to find the chromatic number of an infinite graph?
5. q's-to-SAT in Poly Time has been studied. Some results similar. But the following is different:
   If $\mathrm{SAT}(\phi_1)\cdots\mathrm{SAT}(\phi_k)$ can be computed in poly time with $k-1$ queries to X then $\Sigma_2^p = \Pi_2^p$, so we think not.

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**
Well. . . its not as though I've wrote a book on it.

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**

Well. . . its not as though I've wrote a book on it.

`https://www.amazon.com/Bounded-Queries-Recursion-Progress-Computer-ebook/dp/B000W98WU4?ref_=ast_author_mpb`

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**

Well... its not as though I've wrote a book on it.

https://www.amazon.com/Bounded-Queries-Recursion-Progress-Computer-ebook/dp/B000W98WU4?ref_=ast_author_mpb

Oh! I did!

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**
Well... its not as though I've wrote a book on it.
`https://www.amazon.com/Bounded-Queries-Recursion-Progress-Computer-ebook/dp/B000W98WU4?ref_=ast_author_mpb`

Oh! I did!
Published in 1991. Sold about 1000 copies, the last two in 2014.

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**
Well... its not as though I've wrote a book on it.
`https://www.amazon.com/Bounded-Queries-Recursion-Progress-Computer-ebook/dp/B000W98WU4?ref_=ast_author_mpb`

Oh! I did!
Published in 1991. Sold about 1000 copies, the last two in 2014.

1. I bought a copy since I didn't have one and the Chairman was assembling a display of books by faculty.

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**
Well... its not as though I've wrote a book on it.
`https://www.amazon.com/Bounded-Queries-Recursion-Progress-Computer-ebook/dp/B000W98WU4?ref_=ast_author_mpb`

Oh! I did!
Published in 1991. Sold about 1000 copies, the last two in 2014.

1. I bought a copy since I didn't have one and the Chairman was assembling a display of books by faculty.
   Amazon asked me if I wanted to write a review, so I wrote one one which is still there.

Is This a Bill-Topic?

Jeremy wants to know: **Is this a Bill-Topic?**
Well... its not as though I've wrote a book on it.
`https://www.amazon.com/Bounded-Queries-Recursion-Progress-Computer-ebook/dp/B000W98WU4?ref_=ast_author_mpb`

Oh! I did!
Published in 1991. Sold about 1000 copies, the last two in 2014.

1. I bought a copy since I didn't have one and the Chairman was assembling a display of books by faculty.
   Amazon asked me if I wanted to write a review, so I wrote one one which is still there.
2. Tell story about Adam Winkler buying a copy.