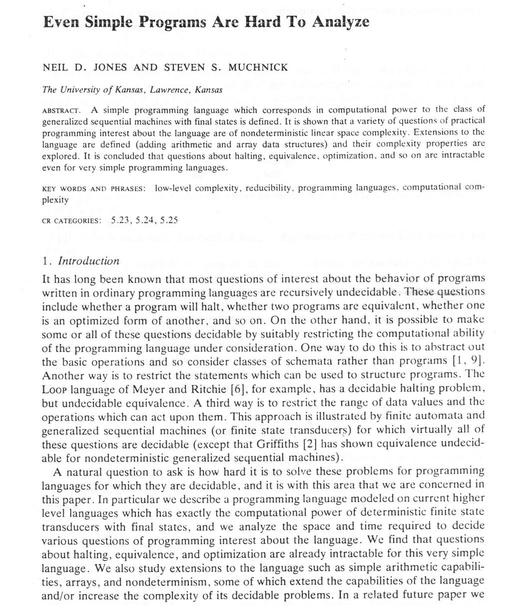
Simple Programs that are Hard to Analyze

Amir M. Ben-Amram

Neil Jones Workshop

June 2014



Even Simple Programs Are Hard To Analyze

NEIL D. JONES AND STEVEN S. MUCHNICK

The University of Kansas, Lawrence, Kansas

ABSTRACT. A simple programming language which corresponds in computational power to the class of generalized sequential machines with final states is defined. It is shown that a variety of questions of practical programming interest about the language are of nondeterministic linear space complexity. Extensions to the language are defined (adding arithmetic and array data structures) and their complexity properties are explored. It is concluded that questions about halting, equivalence, optimization, and so on are intractable even for very simple programming languages.

KEY WORDS AND PHRASES: low-level complexity, reducibility, programming languages, computational complexity

CR CATEGORIES: 5.23, 5.24, 5.25

1. *Introduction*

It has long been known that most questions of interest about the behavior of programs written in ordinary programming languages are recursively undecidable. These questions include whether a program will halt, whether two programs are equivalent, whether one is an optimized form of another, and so on. On the other hand, it is possible to make some or all of these questions decidable by suitably restricting the computational ability of the programming language under consideration. One way to do this is to abstract out the basic operations and so consider classes of schemata rather than programs [1, 9]. Another way is to restrict the statements which can be used to structure programs. The LOOP language of Meyer and Ritchie [6], for example, has a decidable halting problem, but undecidable equivalence. A third way is to restrict the range of data values and the operations which can act upon them. This approach is illustrated by finite automata and generalized sequential machines (or finite state transducers) for which virtually all of these questions are decidable (except that Griffiths [2] has shown equivalence undecidable for nondeterministic generalized sequential machines).

A natural question to ask is how hard it is to solve these problems for programming languages for which they are decidable, and it is with this area that we are concerned in this paper. In particular we describe a programming language modeled on current higher level languages which has exactly the computational power of deterministic finite state transducers with final states, and we analyze the space and time required to decide various questions of programming interest about the language. We find that questions about halting, equivalence, and optimization are already intractable for this very simple language. We also study extensions to the language such as simple arithmetic capabilities, arrays, and nondeterminism, some of which extend the capabilities of the language and/or increase the complexity of its decidable problems. In a related future paper we

JACM, 1978

JACM

The Complexity of Finite Memory Programs with Recursion

NEIL D. JONES

University of Aarhus, Aarhus, Denmark

AND

STEVEN S. MUCHNICK

The University of Kansas, Lawrence, Kansas

ABSTRACT. In order to study the effects of recursion on the complexity of program analysis, a finite memory machine with recursive calls is defined, as well as two parameter passing mechanisms which extend the power of the language. Close upper and lower bounds on the complexity of determining whether a program accepts the empty language are given for each of the three program models. It is shown that such questions as acceptance of the empty set, equivalence, and so on are intractable even for these relatively simple programs.

KEY WORDS AND PHRASES: computational complexity, program analysis, program equivalence, recursive programs, pushdown automata, call by name, finite memory programs

CR CATEGORIES: 5.22, 5.23, 5.24, 5.25, 5.27

1. *Introduction*

In [3] we studied the computational complexity of a number of questions of both programming and theoretical interest (e.g. halting, looping, equivalence) concerning the behavior of programs written in an extremely simple programming language. These finite memory programs or FMPS (pronounced "fumps") model the behavior of Fortran-like programs with a finite memory whose size can be determined by examination of the program itself. The main results of [3] are that determining halting, equivalence, looping, etc., are all of essentially the same complexity and that such analyses generally require nondeterministic algorithms with tape bounds at least proportional to the amount of memory which the program being analyzed can address, as a function of the size of the program. More precisely, if we define ACCEPT to be the set of all finite memory programs which halt and accept at least one input string, then one of our results is that

$$\text{ACCEPT} \in \text{NSPACE}(n) - \bigcup_{\epsilon>0} \text{NSPACE}(n^{1-\epsilon}).$$

Throughout the remainder of this paper we shall assume that the reader is familiar with the notation, terminology, and methods of [3], although any reader generally acquainted with complexity theory should be able to follow this paper with little difficulty.

JACM, 1977

Even Simple Progr

NEIL D. JONES AND ST

The University of Kansas, Lawren

ABSTRACT. A simple programmi

JACM, 1978

The Complexity of Fi

NEIL D. JONES

University of Aarhus, Aarhus, Denmar

JACM, 1977: Main Contribution

Even Simple Programs are Hard to Analyze

JACM, 1977: Main Contribution

Even Simple Programs are Hard to Analyze

1. A class of simple programs
   - Finite-memory programs

JACM, 1977: Main Contribution

Even Simple Programs are Hard to Analyze

1. A class of simple programs
    - Finite-memory programs
2. Certain decision problems regarding programs
    - Termination: Does the computation always terminate?
    - Reachability: Does it ever reach a certain state?

JACM, 1977: Main Contribution

Even Simple Programs are Hard to Analyze

1. A class of simple programs
   - Finite-memory programs
2. Certain decision problems regarding programs
   - Termination: Does the computation always terminate?
   - Reachability: Does it ever reach a certain state?
3. Classification of the computational complexity of the problems

JACM, 1977: Main Contribution

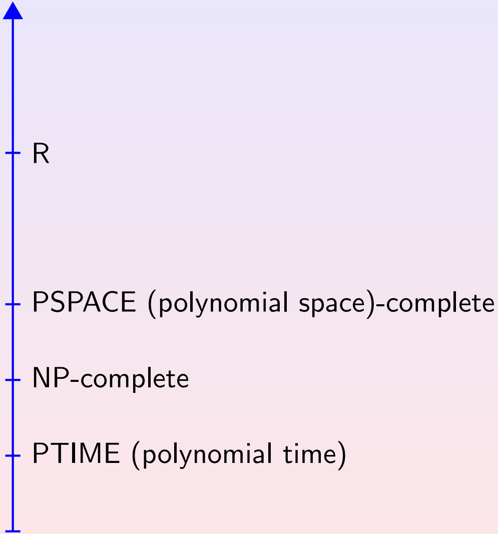
Even Simple Programs are Hard to Analyze

1. A class of simple programs
   - Finite-memory programs
2. Certain decision problems regarding programs
   - Termination: Does the computation always terminate?
   - Reachability: Does it ever reach a certain state?
3. Classification of the computational complexity of the problems

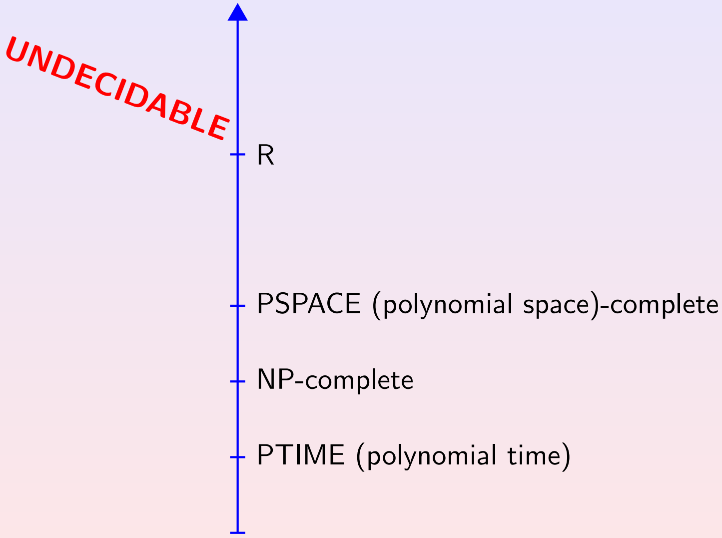
This talk

Some interesting program classes and hardness results

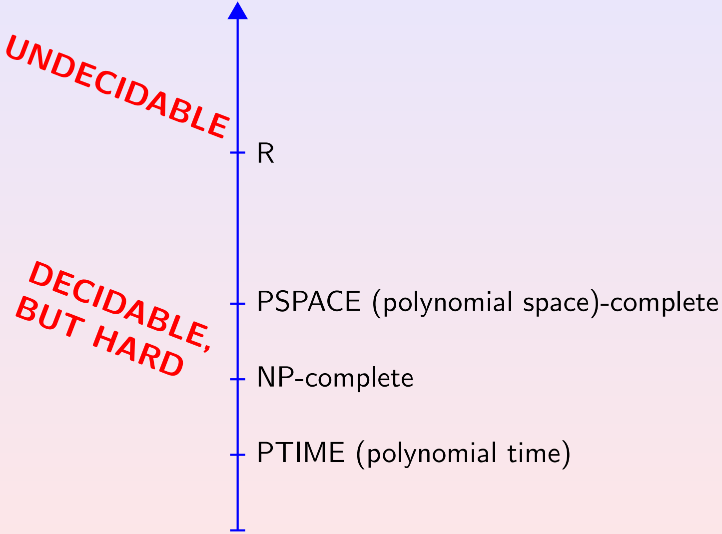
Scale of Computational Hardness



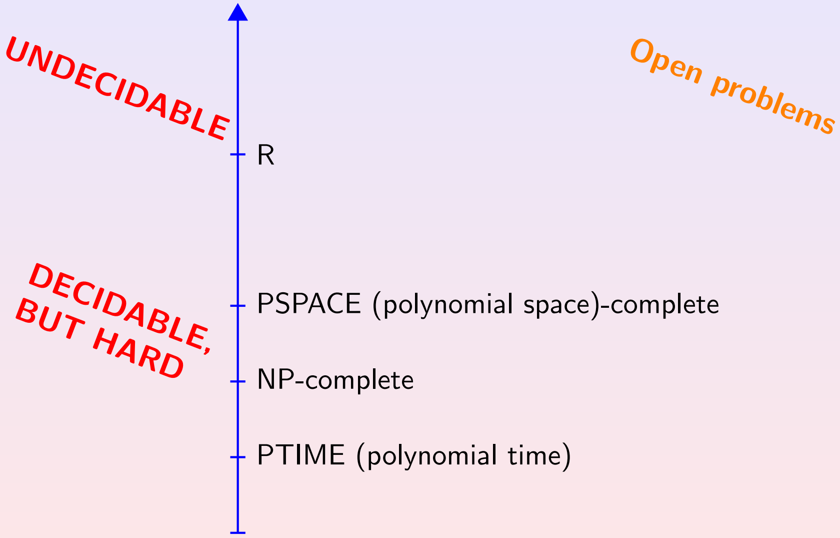
Scale of Computational Hardness



Scale of Computational Hardness



Scale of Computational Hardness



Example 1: Finite Memory Programs

```
1: X := 'a'
2: Y := 'b'
3: if X=Y goto 1
4: end
```

- Programs with BASIC structure (assignments/tests/goto)
  - A finite set of variables.
  - Each variable holds one character.

Example 1: Finite Memory Programs

```
1: X := 'a'
2: Y := 'b'
3: if X=Y goto 1
4: end
```

- Programs with BASIC structure (assignments/tests/goto)
  - A finite set of variables.
  - Each variable holds one character.
- Such a program has a finite state-space so exhaustive search can answer our questions (in polynomial space).

Example 1: Finite Memory Programs

```
1: X := 'a'
2: Y := 'b'
3: if X=Y goto 1
4: end
```

- Programs with BASIC structure (assignments/tests/goto)
  - A finite set of variables.
  - Each variable holds one character.
- Such a program has a finite state-space so exhaustive search can answer our questions (in polynomial space).
- Jones and Muchnick's result: they are PSPACE-complete—as hard as they could get.

Example 2: Counter Programs

```
1: X := 0
2: Y := X+1
3: if X=Y goto 1
4: Y := Y-1
5: end
```

- Values are unrestricted integers and subject to increment, decrement and test.

Example 2: Counter Programs

```
1: X := 0
2: Y := X+1
3: if X=Y goto 1
4: Y := Y-1
5: end
```

- Values are unrestricted integers and subject to increment, decrement and test.
- This language is Turing-complete.

Example 2: Counter Programs

```
1: X := 0
2: Y := X+1
3: if X=Y goto 1
4: Y := Y-1
5: end
```

- Values are unrestricted integers and subject to increment, decrement and test.
- This language is Turing-complete.
- These are not simple programs.

Example 3: Linear Loops

```
while ( x>=0 && y>=0 && 2*z+w >= 3 && ... )
 {
     x := x-y;
     y := y-1;
     ...
}
```

- Just one loop with linear expressions in the assignments and the loop condition.
- Matrix notation:

Example 3: Linear Loops

```
while ( x>=0 && y>=0 && 2*z+w >= 3 && ... )
 {
      x := x-y;
      y := y-1;
      ...
}
```

- Just one loop with linear expressions in the assignments and the loop condition.
- Matrix notation:

$$\texttt{while } A\vec{x} \leq c: \quad \vec{x} \leftarrow B\vec{x} + d$$

$\vec{x}$ is the *state vector*

Example 3: Linear Loops

```
while ( x>=0 && y>=0 && 2*z+w >= 3 && ... )
 {
     x := x-y;
     y := y-1;
     ...
}
```

- Just one loop with linear expressions in the assignments and the loop condition.
- Matrix notation:

$$\texttt{while } A\vec{x} \leq c\text{:} \quad \vec{x} \leftarrow B\vec{x} + d$$

$\vec{x}$ is the *state vector*

- Problem: termination (for all initial states)

Example 3: Linear Loops: results

Decidability of the termination problem is unknown.

Example 3: Linear Loops: results

Decidability of the termination problem is unknown.

B., Genaim and Masud (2012) considered certain *extended forms*

Example 3: Linear Loops: results

Decidability of the termination problem is unknown.

B., Genaim and Masud (2012) considered certain *extended forms*

1. Include the assignment form `Y := isPositive(X)` where

$$\texttt{isPositive}(x) = \begin{cases} 0 & \mathrm{x} \leq 0 \\ 1 & \mathrm{x} > 0 \end{cases}$$

   Termination is undecidable.

Example 3: Linear Loops: results

Decidability of the termination problem is unknown.

B., Genaim and Masud (2012) considered certain *extended forms*

1. Include the assignment form `Y := isPositive(X)` where

$$\texttt{isPositive}(x) = \begin{cases} 0 & \mathrm{x} \leq 0 \\ 1 & \mathrm{x} > 0 \end{cases}$$

   Termination is undecidable.

   - Note: if we could compute products (e.g., $\texttt{isPositive}(X) \star Y$) we could simulate branches quite easily.

Example 3: Linear Loops: results

Decidability of the termination problem is unknown.

B., Genaim and Masud (2012) considered certain *extended forms*

1. Include the assignment form `Y := isPositive(X)` where

$$\texttt{isPositive}(x) = \begin{cases} 0 & \mathrm{x} \leq 0 \\ 1 & \mathrm{x} > 0 \end{cases}$$

   Termination is undecidable.

   - Note: if we could compute products (e.g., $\texttt{isPositive}(X) \star Y$) we could simulate branches quite easily.

2. Termination is also undecidable for loops with a single test:

$$\begin{array}{l} \texttt{while } A\vec{x} \leq c\text{:} \\ \qquad \texttt{if } x_1 > 0 \texttt{ then } \vec{x} \leftarrow B_1\vec{x} \quad \texttt{else } \vec{x} \leftarrow B_2\vec{x} \end{array}$$

Example 3: Linear Loops: Variations

The point-to-point problem

$\vec{x} \leftarrow a;$
$\texttt{while } (\vec{x} \neq b)\ \vec{x} \leftarrow A\vec{x}$

- Starts at a given point, a
- Has to reach point b to terminate

Example 3: Linear Loops: Variations

The point-to-point problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while}\ (\vec{x} \neq b)\ \vec{x} \leftarrow A\vec{x}$$

- Starts at a given point, a
- Has to reach point b to terminate
- Solved in polynomial time (Kannan and Lipton, 1986).

Example 3: Linear Loops: Variations

The point-to-point problem

$\vec{x} \leftarrow a;$
$\texttt{while } (\vec{x} \neq b) \ \vec{x} \leftarrow A\vec{x}$

- Starts at a given point, a
- Has to reach point b to terminate
- Solved in polynomial time (Kannan and Lipton, 1986).

The point-to-hyperplane problem

$\vec{x} \leftarrow a;$
$\texttt{while } (b \cdot \vec{x} \neq 0) \ \vec{x} \leftarrow A\vec{x}$

Example 3: Linear Loops: Variations

The point-to-point problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while}\ (\vec{x} \neq b)\ \vec{x} \leftarrow A\vec{x}$$

- Starts at a given point, a
- Has to reach point b to terminate
- Solved in polynomial time (Kannan and Lipton, 1986).

The point-to-hyperplane problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while}\ (b \cdot \vec{x} \neq 0)\ \vec{x} \leftarrow A\vec{x}$$

- NP-hard (Blondel and Portier, 2002).

Example 3: Linear Loops: Variations

The point-to-point problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while } (\vec{x} \neq b)\ \vec{x} \leftarrow A\vec{x}$$

- Starts at a given point, a
- Has to reach point b to terminate
- Solved in polynomial time (Kannan and Lipton, 1986).

The point-to-hyperplane problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while } (b \cdot \vec{x} \neq 0)\ \vec{x} \leftarrow A\vec{x}$$

- NP-hard (Blondel and Portier, 2002).

Example 3: Linear Loops: Variations

The point-to-point problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while } (\vec{x} \neq b) \ \vec{x} \leftarrow A\vec{x}$$

- Starts at a given point, a
- Has to reach point b to terminate
- Solved in polynomial time (Kannan and Lipton, 1986).

The point-to-hyperplane problem

$$\vec{x} \leftarrow a;$$
$$\texttt{while } (b \cdot \vec{x} \neq 0) \ \vec{x} \leftarrow A\vec{x}$$

- NP-hard (Blondel and Portier, 2002).

Example 3: Linear Loops: Variations

The point-to-point problem

$\vec{x} \leftarrow a;$
$\texttt{while}\ (\vec{x} \neq b)\ \vec{x} \leftarrow A\vec{x}$

- Starts at a given point, a
- Has to reach point b to terminate
- Solved in polynomial time (Kannan and Lipton, 1986).

The point-to-hyperplane problem

$\vec{x} \leftarrow a;$
$\texttt{while}\ (b \cdot \vec{x} \neq 0)\ \vec{x} \leftarrow A\vec{x}$

- NP-hard (Blondel and Portier, 2002).
- Decidability is open!

Example 3: Linear Loops: Further Variations

Cortier (2002): variables hold natural numbers.

Coefficient matrix A must consist of natural numbers;
Transformation $\vec{x} \leftarrow A\vec{x} + b$ is iterated as long as no number becomes negative (there may be negative values in b)

The problem: point-to-point reachability.

Example 3: Linear Loops: Further Variations

Cortier (2002): variables hold natural numbers.

Coefficient matrix A must consist of natural numbers;
Transformation $\vec{x} \leftarrow A\vec{x} + b$ is iterated as long as no number becomes negative (there may be negative values in b)

The problem: point-to-point reachability.

Result: decidable, but the algorithm provided has enormous complexity ($\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\} n$).

Example 4: The Budach Mouse

A problem posed by Lothar Budach at FCT 1979.

The mouse (*Mus Automaticus Budachi*) is a simple finite-state machine, which wanders about the first positive octant of the integer grid.

Its program is a string over $\{E, N\}$, through which is cycles indefinitely.

Example 4: The Budach Mouse

Aleš Pultr; Josef Úlehla
On two problems of mice (1982)

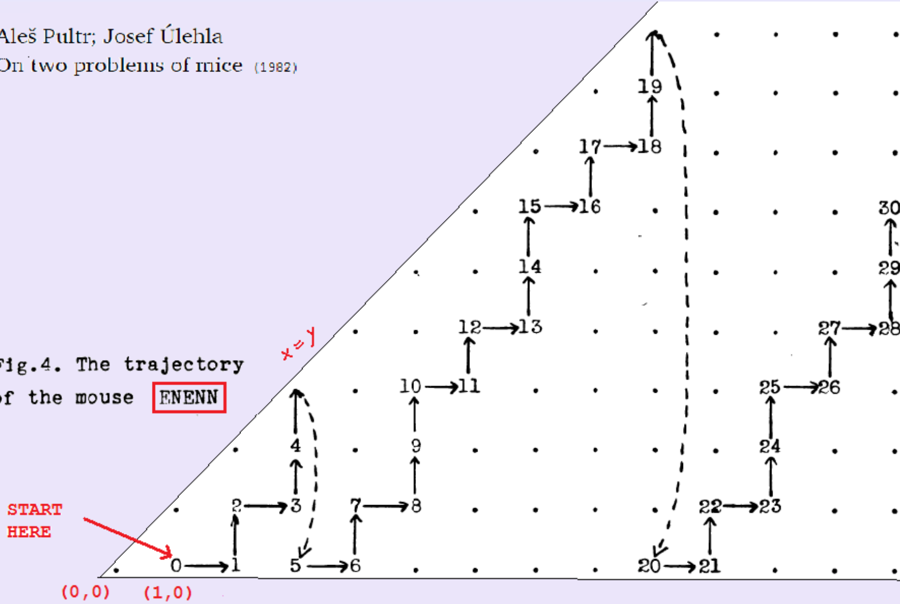


Fig.4. The trajectory of the mouse ENENN

Aleš Pultr; Josef Úlehla
On two problems of mice (1982)

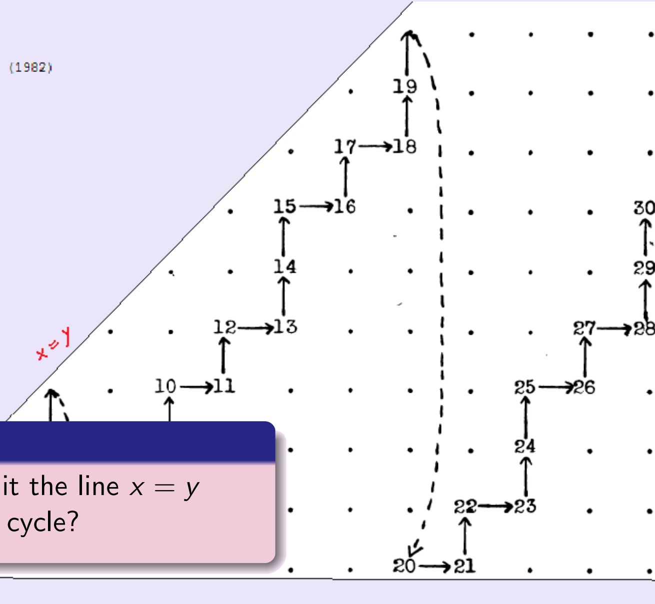


Fig.4. The trajectory of the mouse $\boxed{\text{ENENN}}$

(0,0) (1,0)

Decision problem

Will the mouse ever hit the line $x = y$ when in step k of the cycle?

Example 4: The Budach Mouse

Aleš Pultr; Josef Úlehla
On two problems of mice (1982)

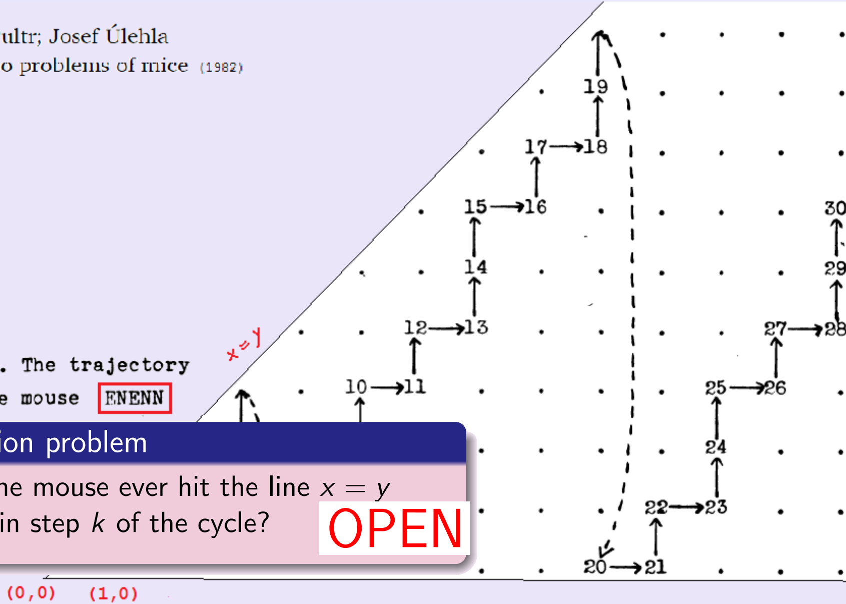


Fig.4. The trajectory of the mouse ENENN

Decision problem

Will the mouse ever hit the line $x = y$ when in step k of the cycle?

OPEN

Conclusion

Conclusion

Neil was right!