Programs That Cannot Exist
A Glimpse of Theoretical Computer Science

Steffen Lösch

University of Cambridge - Computer Laboratory
What is a Program?

1. Put 500 ml of water in kettle
2. While the water is not boiling
   1. Leave the kettle on
3. Pour boiling water into a mug
4. Select teabag and look up optimal brewing time
5. Put teabag in the water
6. Until optimal brewing time is over
   1. Leave teabag in water
   2. Stir
7. Remove teabag
What is a Program?

1. Put 500 ml of water in kettle
2. While the water is not boiling
   1. Leave the kettle on
3. Pour boiling water into a mug
4. Select teabag and look up optimal brewing time
5. Put teabag in the water
6. Until optimal brewing time is over
   1. Leave teabag in water
   2. Stir
7. Remove teabag
What is a Program? (2)

1. input scale

2. for i <- 1 to 100
   1. for j <- 1 to 100
      x <- i * scale / 100
      y <- j * scale / 100
      c <- round(x^2 + y^2)
      if c even then
         plot(i,j)

(Example take from "The New Turing Omnibus", A.K. Dewedney)
What is a Program? (2)

1. input scale
2. for i <- 1 to 100
   1. for j <- 1 to 100
      x <- i * scale / 100
      y <- j * scale / 100
      c <- round(x^2 + y^2)
      if c even
         then plot(i,j)
What is a Program? (2)

(Example take from "The New Turing Omnibus", A.K. Dewedney)
1. *input* scale
2. *for* $i \leftarrow 1$ to 100
   1. *for* $j \leftarrow 1$ to 100
      x <- $i \times \text{scale} / 100$
      y <- $j \times \text{scale} / 100$
      c <- round($x^2 + y^2$)
      if c even
         then plot($i,j$)
1. input scale
2. for i <- 1 to 100
   1. for j <- 1 to 100
      x <- i * scale / 100
      y <- j * scale / 100
      c <- round(x^2 + y^2)
      if c even
      then plot(i,j)
What is a Program? (3)

Wallpaper

1. input scale
2. for i <- 1 to 100
   1. for j <- 1 to 100
      x <- i * scale / 100
      y <- j * scale / 100
      c <- round(x^2 + y^2)
      if c even
         then plot(i,j)
Wallpaper

1. input scale
2. for i <- 1 to 100
   1. for j <- 1 to 100
      x <- i * scale / 100
      y <- j * scale / 100
      c <- round(x^2 + y^2)
      if c even then plot(i,j)
What is a Program? (3)

Wallpaper

1. \textit{input} scale
2. \textit{for} i \text{ <- 1 to 100}
   1. \textit{for} j \text{ <- 1 to 100}
   2. \texttt{x <- i * scale / 100}
   3. \texttt{y <- j * scale / 100}
   4. \texttt{c <- round(x^2 + y^2)}
   5. \texttt{if c even}
   6. \texttt{then plot(i,j)}

"15" input

Steffen Lösch
Programs That Cannot Exist
The Halting Problem

Issue: programs can run forever and never stop!

Example:

1. while 1 = 1
   1. x <- x + 1

A solution to the Halting Problem would look like this:

```plaintext
solve \text{halt} \left( \text{input} \right)
{ "true" if \text{prog}(0) \text{ halts} \\
"false" if \text{prog}(0) \text{ does not halt} }

output \left( \text{input} \left( \text{input} \right) \right)
{ "true" if \text{prog}(\text{prog}) \text{ halts} \\
"false" if \text{prog}(\text{prog}) \text{ does not halt} }
```
Issue: programs can run forever and never stop!
The Halting Problem

*Issue:* programs can run forever and never stop!

▶ Example:

1. `while 1 = 1`
   1. `x <- x + 1`
The Halting Problem

**Issue:** programs can run forever and never stop!

- Example:

  1. while 1 = 1
     1. x <- x + 1

A solution to the Halting Problem would look like this:

```plaintext
solve_halt

???
```
The Halting Problem

Issue: programs can run forever and never stop!

Example:

1. while 1 = 1
   1. x <- x + 1

A solution to the Halting Problem would look like this:
The Halting Problem

**Issue:** programs can run forever and never stop!

▶ Example:

1. while 1 = 1
   1. x <- x + 1

A solution to the Halting Problem would look like this:

```
solve_halt

prog | ??? | output

input

{ "true" if prog(0) halts
  "false" if prog(0) does not halt
```
The Halting Problem

**Issue:** programs can run forever and never stop!

Example:

1. \texttt{while } 1 = 1
1. \texttt{x <- x + 1}

A solution to the Halting Problem would look like this:

\begin{center}
\begin{tikzpicture}
  \node (input) at (0,0) {input};
  \node (prog) at (2,0) {prog};
  \node (solve_halt) at (4,0) {solve\textunderscore halt};
  \node (output) at (6,0) {output};

  \draw[->] (input) -- (prog);
  \draw[->] (prog) -- (solve_halt);
  \draw[->] (solve_halt) -- (output);

  \node at (solve_halt) {???};

  \node at (solve_halt) {\begin{cases}
    "true" & \text{if prog(prog) halts} \\
    "false" & \text{if prog(prog) does not halt}
  \end{cases}};
\end{tikzpicture}
\end{center}
The Barber of London

The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?

▶ Yes, he does.
⇒ He is shaved by the barber of London.
⇒ He does not shave himself.
☇

▶ No, he does not.
⇒ He is a man of London who does not shave himself.
⇒ He is shaved by the barber of London (i.e. himself).
☇

Steffen Lösch
Programs That Cannot Exist
Assumption:
The barber of London shaves all (and only) men in London who do not shave themselves.
Assumption:  
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?
The Barber of London

Assumption:
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?
- Yes, he does.

⇒ He is shaved by the barber of London.
⇒ He does not shave himself.
☇

⇒ He is a man of London who does not shave himself.
⇒ He is shaved by the barber of London (i.e. himself).
☇

Steffen Lösch
Programs That Cannot Exist
The Barber of London

**Assumption:**
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?
- Yes, he does.
  - He is shaved by the barber of London.
Assumption:
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?

- Yes, he does.
  ⇒ He is shaved by the barber of London.
  ⇒ He does not shave himself.
The Barber of London

**Assumption:**
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?

- Yes, he does.
  - ⇒ He is shaved by the barber of London.
  - ⇒ He does not shave himself.
- No, he does not.

⇒ He is a man of London who does not shave himself.
⇒ He is shaved by the barber of London (i.e. himself).
The Barber of London

**Assumption:**
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?

- Yes, he does.
  - ⇒ He is shaved by the barber of London.
  - ⇒ He does not shave himself.

- No, he does not.
  - ⇒ He is a man of London who does not shave himself.
**Assumption:**
The barber of London shaves all (and only) men in London who do not shave themselves.

Does he shave himself?

- Yes, he does.
  - ⇒ He is shaved by the barber of London.
  - ⇒ He does not shave himself. ⇓

- No, he does not.
  - ⇒ He is a man of London who does not shave himself.
  - ⇒ He is shaved by the barber of London (i.e. himself). ⇓
The Program of London

Assumption: There is a program \textit{solve halt} solving the Halting Problem.

Then we can build a new program:

1. \textbf{input} \textit{prog}
2. \textbf{if} \textit{solve_halt(prog)} = \textit{true} \\
   \hspace{1em} \textbf{then} \textbf{run forever} \\
   \hspace{1em} \textbf{else} \textbf{halt}

\textbf{barber} \textbf{input} \textbf{prog} \textbf{output} \textbf{""}
Assumption:
There is a program "solve_halt" solving the Halting Problem.
Assumption:
There is a program "solve_halt" solving the Halting Problem.

Then we can build a new program:

1. input prog
2. if solve_halt(prog) = "true"
   1. then run forever
   2. else halt
Assumption:
There is a program "solve_halt" solving the Halting Problem.

Then we can build a new program:

\textbf{barber}

1. \texttt{input prog}
2. \texttt{if solve\_halt(prog) = "true"}
   1. \texttt{then run forever}
   2. \texttt{else halt}
Assumption:
There is a program "solve_halt" solving the Halting Problem.

Then we can build a new program:

```
barber

1. input prog
2. if solve_halt(prog) = "true"
   1. then run forever
   2. else halt
```
The Program of London

Assumption:
There is a program "solve_halt" solving the Halting Problem.

Then we can build a new program:

```
barber
1. input prog
2. if solve_halt(prog) = "true"
   1. then run forever
   2. else halt

input
prog

output
"
```
The Program of London (2)

1. \textit{input} prog
2. \textit{if} solve\_halt(prog) = "true"
   1. \textit{then} run forever
   2. \textit{else} halt

\texttt{barber}

Does barber(barber) halt?

\begin{itemize}
\item \textbf{Yes, it does.} \Rightarrow \texttt{solve\_halt(barber) = "false"} \Rightarrow \texttt{barber(barber) does not halt.}
\item \textbf{No, it does not.} \Rightarrow \texttt{solve\_halt(barber) = "true"} \Rightarrow \texttt{barber(barber) does halt.}
\end{itemize}

So there cannot be a program solving the halting problem!
1. input prog
2. if solve_halt(prog) = "true"
   1. then run forever
   2. else halt

Does barber(barber) halt?
The Program of London (2)

barber

1. input \textit{barber}
2. if solve\_halt \textit{barber} = "true"
   1. then run forever
   2. else halt

Does barber(barber) halt?

Steffen Lösch Programs That Cannot Exist
Does barber(barber) halt?

- Yes, it does.
The Program of London (2)

**barber**

1. *input* barber
2. *if* solve_halt(barber) = "true"
   1. *then* run forever
   2. *else* halt

Does barber(barber) halt?

- Yes, it does.
  ⇒ solve_halt(barber) = "false"
The Program of London (2)

Does barber(barber) halt?

- Yes, it does.
  ⇒ solve_halt(barber) = "false"
  ⇒ barber(barber) does not halt. ✗
The Program of London (2)

Does \( \text{barber}(\text{barber}) \) halt?

- Yes, it does.
  \[ \Rightarrow \text{solve\_halt(\text{barber})} = "false" \]
  \[ \Rightarrow \text{barber}(\text{barber}) \text{ does not halt.} \]

- No, it does not.
The Program of London (2)

Does barber(barber) halt?

- Yes, it does.
  \[ \Rightarrow \text{solve}_\text{halt}(\text{barber}) = "false" \]
  \[ \Rightarrow \text{barber}(\text{barber}) \text{ does not halt. } \checkmark \]

- No, it does not.
  \[ \Rightarrow \text{solve}_\text{halt}(\text{barber}) = "true" \]
Does barber(barber) halt?

- Yes, it does.
  \[ \Rightarrow \text{solve}_\text{halt}(\text{barber}) = "false" \]
  \[ \Rightarrow \text{barber}(\text{barber}) \text{ does not halt.} \]

- No, it does not.
  \[ \Rightarrow \text{solve}_\text{halt}(\text{barber}) = "true" \]
  \[ \Rightarrow \text{barber}(\text{barber}) \text{ does halt.} \]
The Program of London (2)

Does barber(barber) halt?

- Yes, it does.
  ⇒ solve_halts(barber) = "false"
  ⇒ barber(barber) does not halt.

- No, it does not.
  ⇒ solve_halts(barber) = "true"
  ⇒ barber(barber) does halt.

So there cannot be a program solving the halting problem!
So What’s Theoretical Computer Science?
So What’s Theoretical Computer Science?

- Complexity and Computation Theory
So What’s Theoretical Computer Science?

- Complexity and Computation Theory
- Verification and Semantics
So What’s Theoretical Computer Science?

- Complexity and Computation Theory
- Verification and Semantics
- Quantum Computing
So What’s Theoretical Computer Science?

- Complexity and Computation Theory
- Verification and Semantics
- Quantum Computing
- ...