TUHH
ALGO

# The Halting Problem

Prof. Matthias Mnich
Hamburg University of Technology
Institute for Algorithms and Complexity

# Welcome to today's lecture

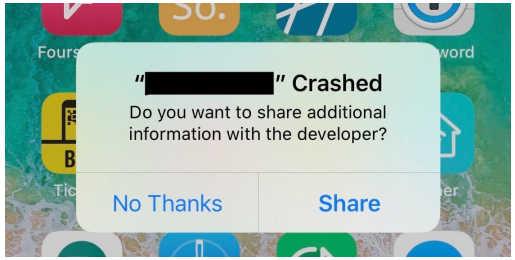
You have the following **background knowledge**:

- You are well-versed with the concept of Turing machine.
- You are well-versed with the concept of recursive sets.
  → Discrete algebraic structures (Prof. Wiehe)

Our **learning goals** for today:

- You get to know the halting problem.
- You understand, why the halting problem is not decidable on Turing machines.


# The crash (of your smartphone)




There are apps which can freeze your phone, either by malicious intent of the developer or by neglect.

E.g., the app could try to execute an illegal mathematical operation, like a division 5 / 0.

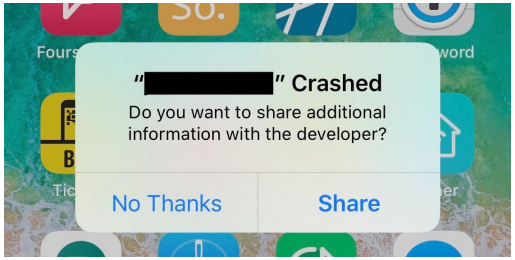
When a modern computer will meet such command, this will lead to a crash, which in the worst case will take the rest of the operating system with it.

Your phone will freeze, and there is nothing you can do but to reset your phone.

Why does this have to happen?

Can one not test such behaviour beforehand?

# The *Freeze* app

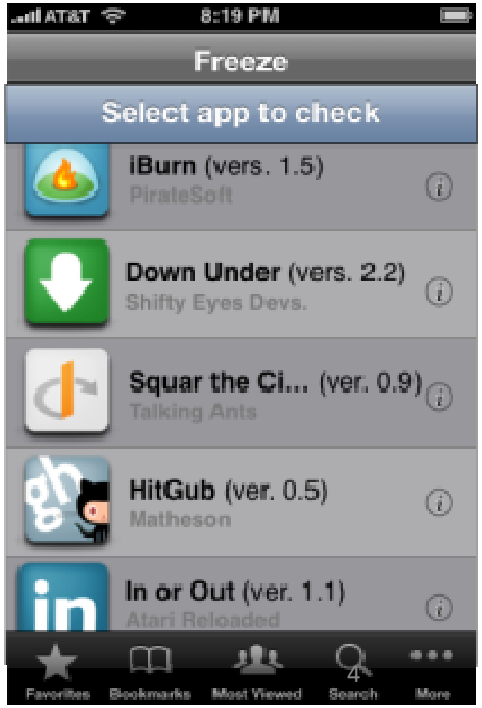



Imagine that someone developed an app *Freeze*, which can check a suspicious app whether it will freeze your phone or not.

*Freeze* would run experiments with the suspicious app, without really freezing your phone.

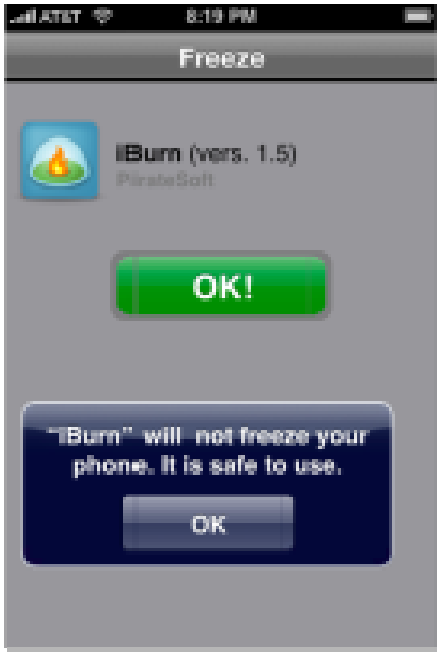
Our Freeze could derive the behaviour of the suspicious app by analyzing its source code – it is not important for us how *Freeze* keeps what it promises.

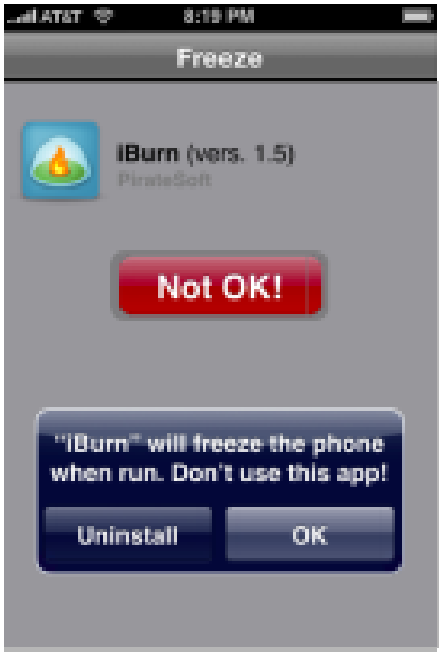
*Freeze* is really easy to use.
First select an app from the list of apps installed on your phone:



# The *Freeze* app

Then *Freeze* runs its analysis, which can take a few minutes, and then displays ist result to you:



# The Paradox app


As soon as *Freeze* appears in the app store, I will build a new app: *Paradox*.
For ist development, I will only need 3 minutes:

```
set Result to
    tell application "Freeze" to
        check application "Paradox"
    end tell
if Result is "OK" then
    display 5/0
else display "Freeze detected that Paradox freezes"
end if
```

# The Paradox app


```
set Result to
    tell application "Freeze" to
        check application "Paradox"
    end tell
if Result is "OK" then
    display 5/0
else display "Freeze detected that Paradox freezes"
end if
```

Download *Paradox* 

and use it;
*Freeze* will protect your phone!

# Is the Paradox app malicious?

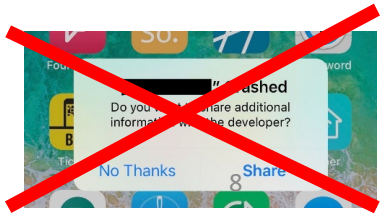

Either Paradox is a malicious app, which freezes the phone, or it is not.

We will show that **both cases are logically impossible.**

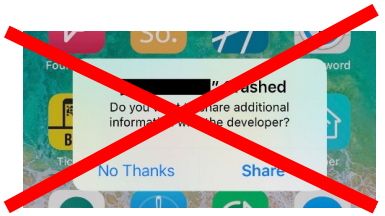
Case 1: Paradox freezes the phone.
Freeze will recognize this, therefore Paradox will return „Freeze detected that Paradox will freeze your phone“ and will terminate normally. So the phone in fact did not freeze, which contradicts our assumption.

Case 2: Paradox will not freez the phone.
Freeze will recognize this, thus Paradox will continue with the division 5/0 and thus freeze the phone. This contradicts our assumption.

So our initial assumption was incorrect. **Thus, an app like freeze cannot exist.** „There is no app for that.“



# The Halting Problem




**An app like Freeze cannot exist.**

This is a consequence of a famous work by Turing for the Halting problem.

**The Halting problem formalizes the question, whether an algorithm will terminate.**

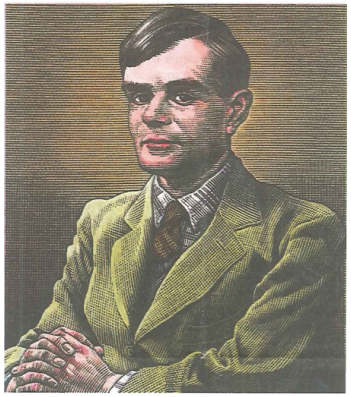
Although this question can be easily answered for many algorithms, Turing showed (with his model of the Turing machine) that no algorithm can decide this question for all possible algorithms and arbitrary inputs.

**Therefore, the Halting problem is algorithmically undecidable.**

# The Halting problem is undecidable.

If the Halting problem would be decidable then there would exist a Turing machine $H$, which for any Turing machine $T$ on input $w$ decides if $T$ eventually stops or runs infinitely long.

The input for $H$ consists of an encoding $c(T)$ of the Turing machine $T$ and an encoding of *its* input $w$.



**Turing's Theorem.**

Such a Turing machine $H$ cannot exist.

Alan M. Turing. *On computable numbers, with an application to the Entscheidungsproblem.* Proceedings of the London Math. Soc., 2, 42 (1937), S. 230–265.

# The Halting Problem is undecidable.

If the Halting problem would be decidable then there would exist a Turing machine $H$, which for any Turing machine $T$ on input $w$ decides if $T$ eventually stops or runs infinitely long.

**Turing's Theorem.**

Such a Turing machine $H$ cannot exist.

A language $L$ is *recursively enumerable* if there is a Turing machine which accepts all words in $L$ and does not accept any words outside of $L$. Probably one has to wait infinitely long to receive the answer.

The Halting problem is decidable, if the Halting problem as well as ist complement are recursively enumerable.

# The Halting Problem is undecidable.

If the Halting problem would be decidable then there would exist a Turing machine $H$, which for any Turing machine $T$ on input $w$ decides if $T$ eventually stops or runs infinitely long.

The Halting problem is decidable, if the Halting problem as well as ist complement are recursively enumerable.

Let $H$ be a hypothetical Turing machine which on input an encoding $c(T)$ of a Turing machine $T$ and a word $w$

- returns a 0 if $T$ on input $w$ does not stop,
- returns a 1 if $T$ on input $w$ stops.

# The Halting problem is undecidable.

The Halting problem is decidable, if the Halting problem as well as ist complement are recursively enumerable.

Let $H$ be a hypothetical Turing machine which on input an encoding $c(T)$ of a Turing machine $T$ and a word $w$

- returns a 0 if $T$ on input $w$ does not stop,
- returns a 1 if $T$ on input $w$ stops.

Let $F^c$ be the Turing machine which starts $H$ on input $(c(T), w)$:

- If $H$ returns a 0 then $F^c$ returns a 1.
- If $H$ returns a 1 then $F^c$ enters an infinite loop.

Ergo: if $H$ decides the Halting problem then $F^c$ is recursively enumerable.

# The Halting problem is undecidable.

The Halting problem is decidable, if the Halting problem as well as ist complement are recursively enumerable.

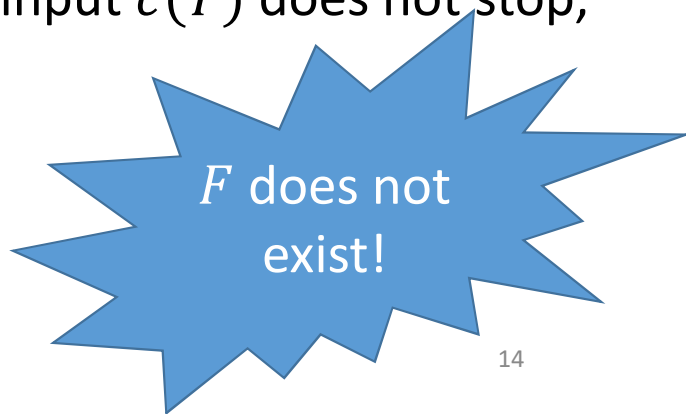
Let $L$ be the language of all Turing machines which do not halt when they receive their own encoding as their input.

Assume a hypothetical Turing machine $F$ which decides $L$.

Ergo: on input $c(T)$ machine $F$ returns value 1 if $T$ on input $c(T)$ does not stop, and which does not stop if $T$ on input $c(T)$ stops.

Thus:

- $F$ stops on input $c(F)$ with output 1, if $F$ on input $c(F)$ does not stop
- $F$ does not stop if $F$ stops on input $c(F)$.



# The Halting problem is undecidable.

$L$ = {TM, which do not halt when receiving their own encoding as input}

The Halting problem is decidable, if the Halting problem as well as ist complement are recursively enumerable.

If the complement of $L$ is recursively enumerable, then so would be $L$:

Let $F^c$ be a hypothetical Turing machine which on input $c(T)$ of a T[illegible]g machine $T$ and word $w$

- outputs a 1 if $T$ on input $w$ does not stop,
- and does not stop when $T$ stops.

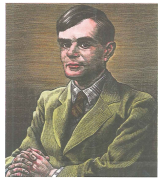
As $F$ does not exist, neither does $F^c$!

We construct a Turing machine $F$, which on input $c(T)$ of $T$ simply starts $F^c$ on input $(c(T), w = c(T))$.

# The Halting problem is undecidable.

The Halting problem is decidable, if the Halting problem as well as ist complement are recursively enumerable.

We have shown: the Halting problem and its complement are not recursively enumerable. Ergo there is no Turing machine for the halting problem.



**Turing's Theorem.**

The Halting problem is algorithmically undecidable.

This result plays a fundamental role in the computability theory.

# Learning goal check

All materials available in the e-learning portal.

Our **Learning Goals** for today:

- You get to know the halting problem.
- You understand why the halting problem is not decidable on Turing machines.


Chapter 8.2

Any questions?