„Hilberts Albtraum“

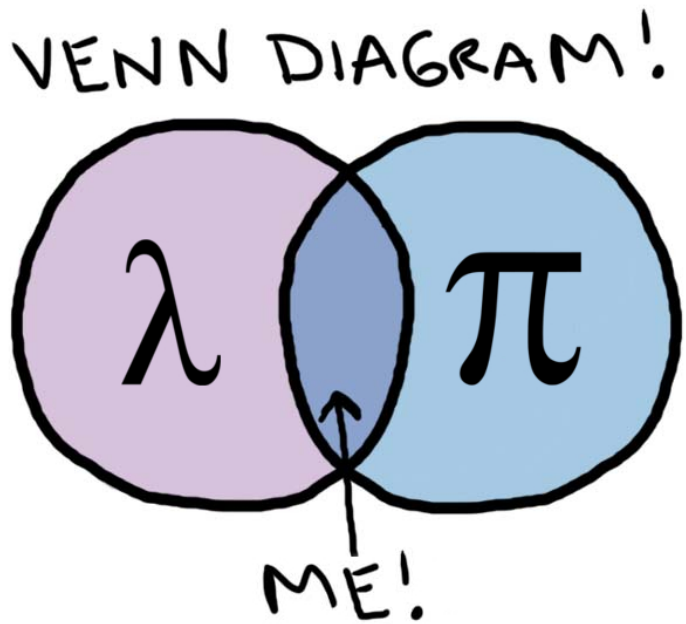
(The answer may *not* be out there!)

Jonas Betzendahl

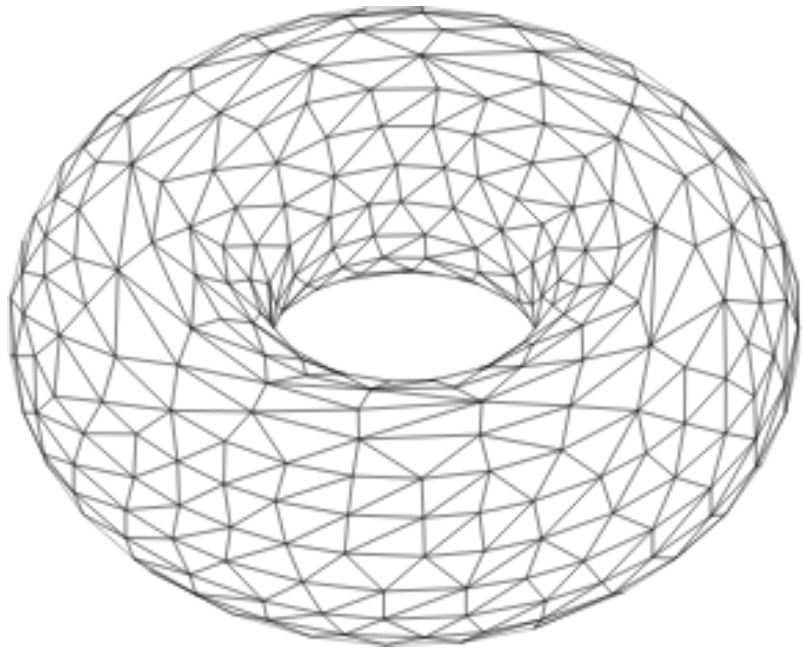
`@jbetzend`

Mein Thema

Wie kann ich einem Computer beibringen,
mit mathematischen Beweisen umzugehen?



„Venn Diagram“ by unknown artist, modified, Image under Fair Use

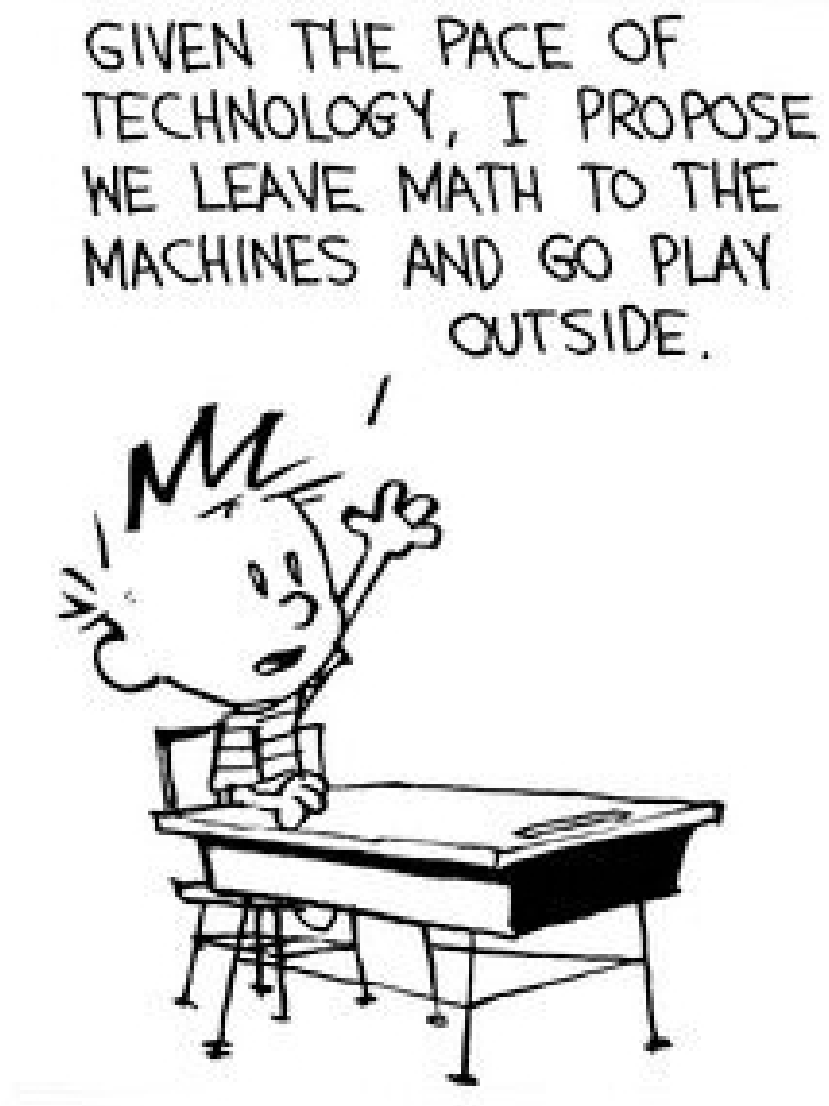


Mathematik ist eine Wissenschaft
fast wie jede andere. . .

„3D Torus . . .“ by GDJ, `www.openclipart.org`, licensed `CC-0`

Math is the reverse of comedy. The anti-joke.
We'll tell you the punchline first, then laboriously explain to you why it was the right punchline.

Joseph Maher, College of Staten Island



„Calvin & Hobbes, 25 NOV 1992“ by Bill Waterson, Image under Fair Use

David Hilbert

Mathematiker (1862-1943) aus Königsberg, bekannt für seine Grundlagenforschung und Problemsammlungen.

Hilberts Traum:

Ein Computer, dem ich eine beliebige mathematische Aussage reichen kann und der mir sagt, ob sie stimmt.

Das ist leicht zu schaffen, wenn wir das *Halteproblem* lösen!

(z.B. durch Auflistung aller Folgerungen aus den Axiomen)

Halteproblem:

Gegeben nur den Quellcode und den Input eines Programms, sage vorher ob dieses Programm jemals „fertig wird“ oder endlos weiter läuft.

Manchmal ist es sehr leicht zu sehen, ob ein Programm jemals halten wird oder nicht:

```haskell
-- Dieses Programm hält quasi sofort
main :: IO ()
main = print $ plus (3,4)
  where
    plus :: (Int, Int) -> Int
    plus (x,y) = x + y
```

Manchmal ist es sehr leicht zu sehen, ob ein Programm jemals halten wird oder nicht:

```haskell
-- Dieses Programm hält quasi sofort
main :: IO ()
main = print $ plus (3,4)
  where
    plus :: (Int, Int) -> Int
    plus (x,y) = x + y
```

```haskell
-- Dieses Programm läuft "für immer"
main :: IO ()
main = forever $ print "lol, infinite loop!"
```

Manchmal ist es aber auch nahezu unmöglich!

```haskell
-- Nobody knows if this ever halts...
main :: IO ()
main = do let results = filter isPerfect [1,3..]
          case results of
            [] -> print " No odd perfect numbers!"
            _  -> print "Yes odd perfect numbers!"

divisors :: Int -> [Int]
divisors n = filter (\x -> n `rem` x == 0) [1..n]

isPerfect :: Int -> Bool
isPerfect n = (sum . divisors) n == n + n
```

Alan Turing

Mathematiker (1912-1954) aus London. Half in Bletchley Park, den dt. *Enigma*-Code zu lösen.

Bewies in seiner Doktorarbeit, dass das Halteproblem *nicht lösbar sein kann* (zumindest im allgemeinen Fall)!

Alan Turing

Mathematiker (1912-1954) aus London. Half in Bletchley Park, den dt. *Enigma*-Code zu lösen.

Bewies in seiner Doktorarbeit, dass das Halteproblem *nicht lösbar sein kann* (zumindest im allgemeinen Fall)!

Aber nehmen wir mal an, es *wäre lösbar*...

Ein magisches Halte-Orakel (1)

Angenommen wir haben ein magisches Orakel, was uns für Quellcode und Input immer sagt, ob ein Programm hält oder nicht.

Ein magisches Halte-Orakel (2)

Angenommen wir haben ein magisches Orakel, was uns für Quellcode und Input immer sagt, ob ein Programm hält oder nicht.

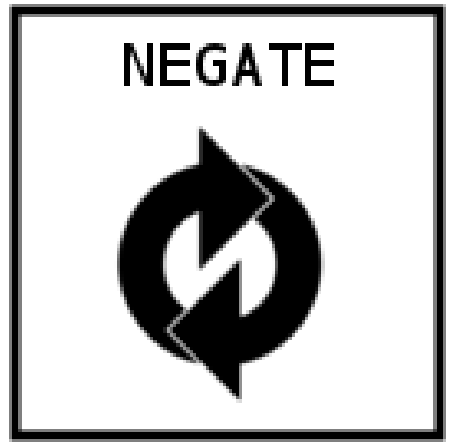
Ein magisches Halte-Orakel (3)

Angenommen wir haben ein magisches Orakel, was uns für Quellcode und Input immer sagt, ob ein Programm hält oder nicht.

Ein magisches Halte-Orakel (4)

Angenommen wir haben ein magisches Orakel, was uns für Quellcode und Input immer sagt, ob ein Programm hält oder nicht.

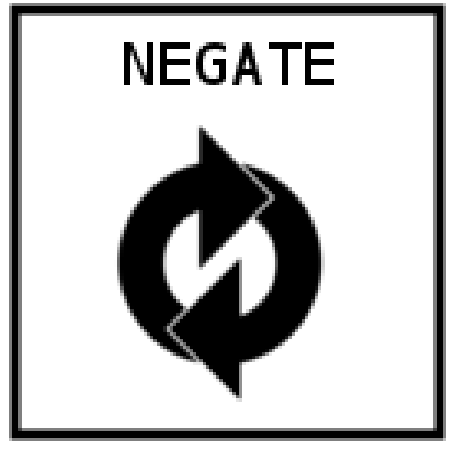
Negierer (1)



Dazu kommt ein Negierer, der ein Haltergebnis umdreht!

Negierer (2)

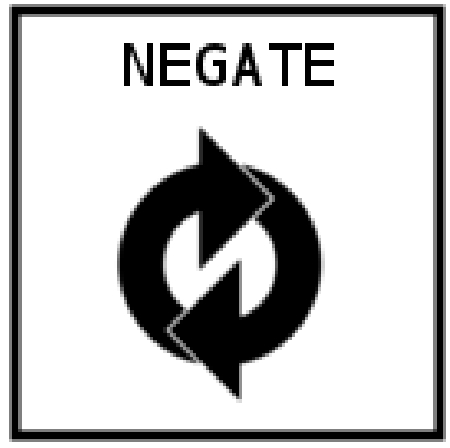
Hält ->



-> Hält nicht!

Dazu kommt ein Negierer, der ein Haltergebnis umdreht!

Negierer (3)

Hält nicht! -> 

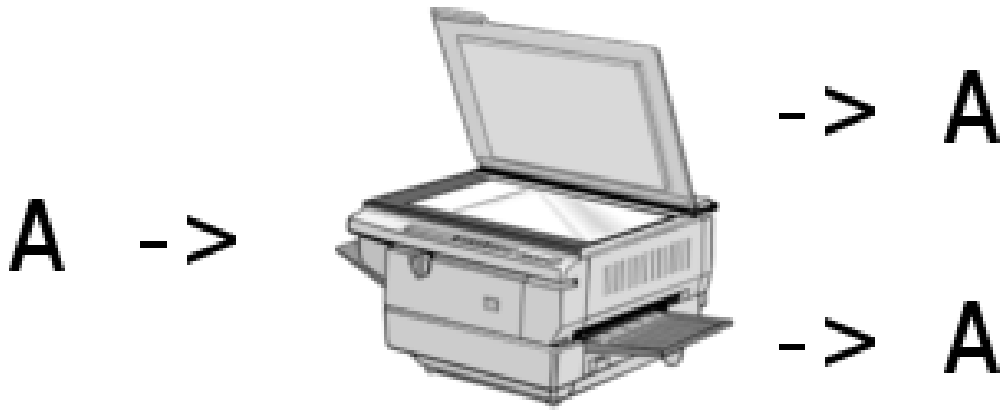
-> Hält!

Dazu kommt ein Negierer, der ein Haltergebnis umdreht!

Duplizierer (1)

Wir haben auch einen Duplizierer, der seine Eingabe verdoppelt!

Duplizierer (2)



Wir haben auch einen Duplizierer, der seine Eingabe verdoppelt!

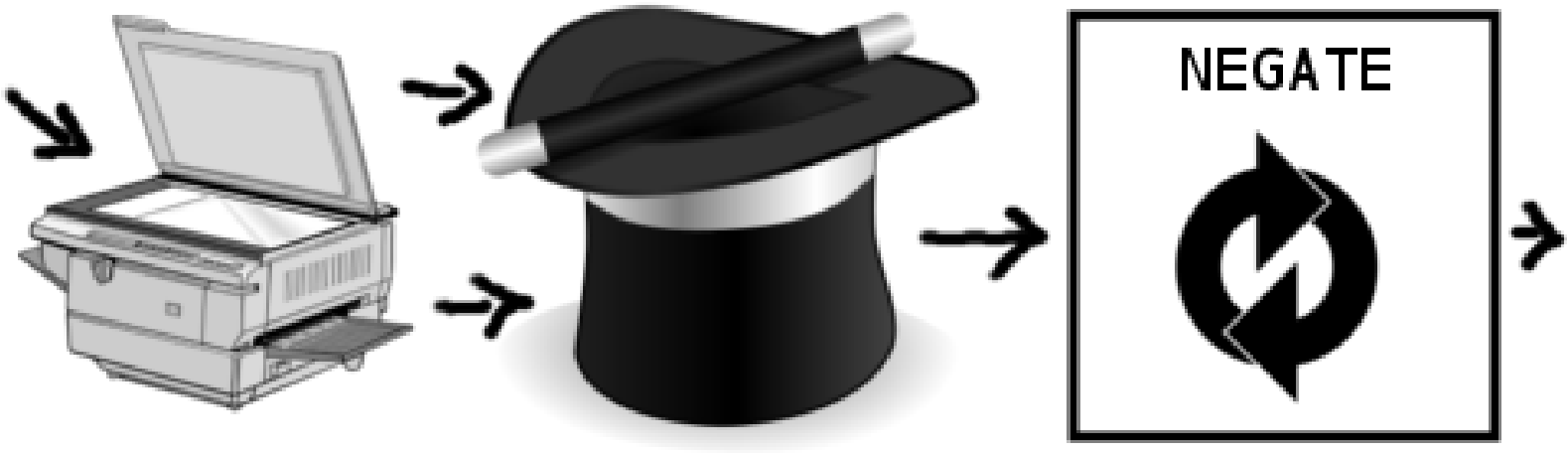
Duplizierer (3)

```
plus ->
```

```
-> plus
-> plus
```

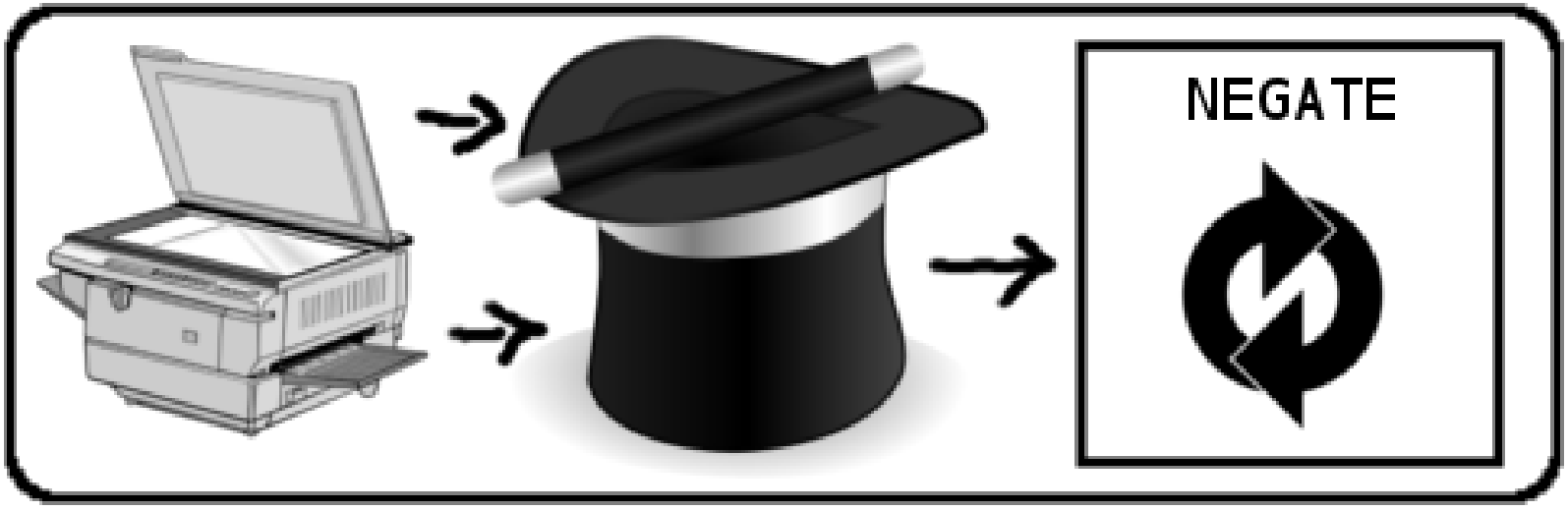
Wir haben auch einen Duplizierer, der seine Eingabe verdoppelt!

Unite and conquer! (1)



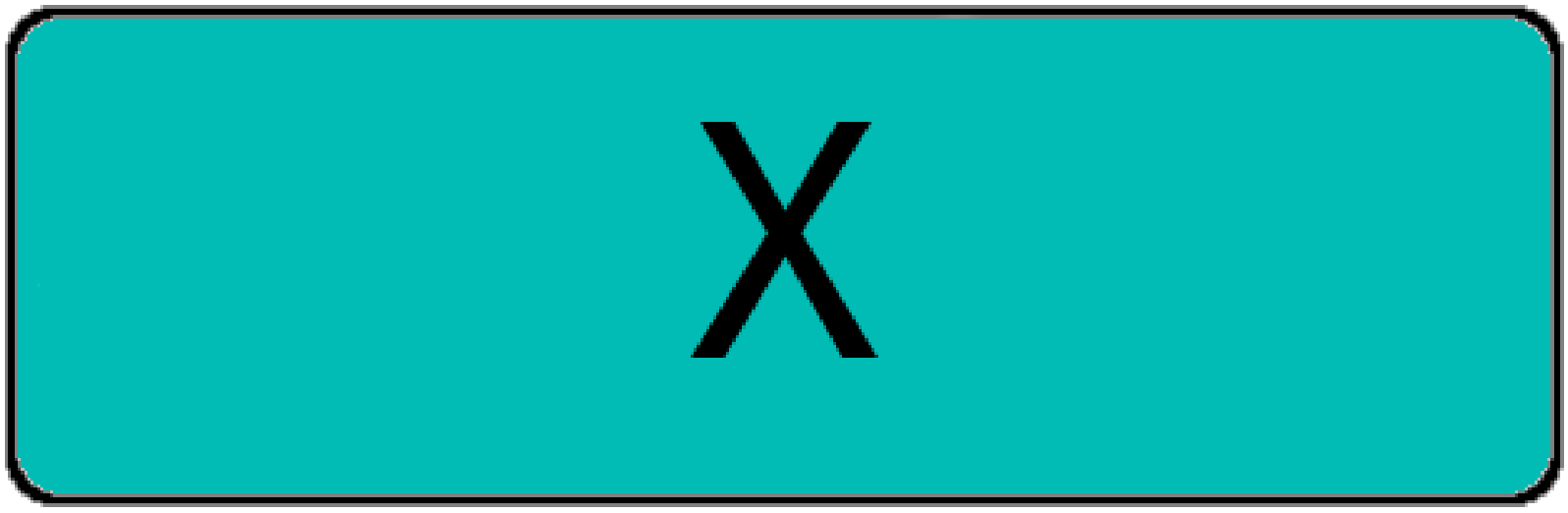
Manche ahnen es vielleicht schon! Alles das können wir jetzt zusammen stecken und hintereinander schalten!

Unite and conquer! (2)



Insbesondere können wir diese Aneinanderreihung als *eine Maschine* betrachen. Nennen wir sie `X`.

Unite and conquer! (3)

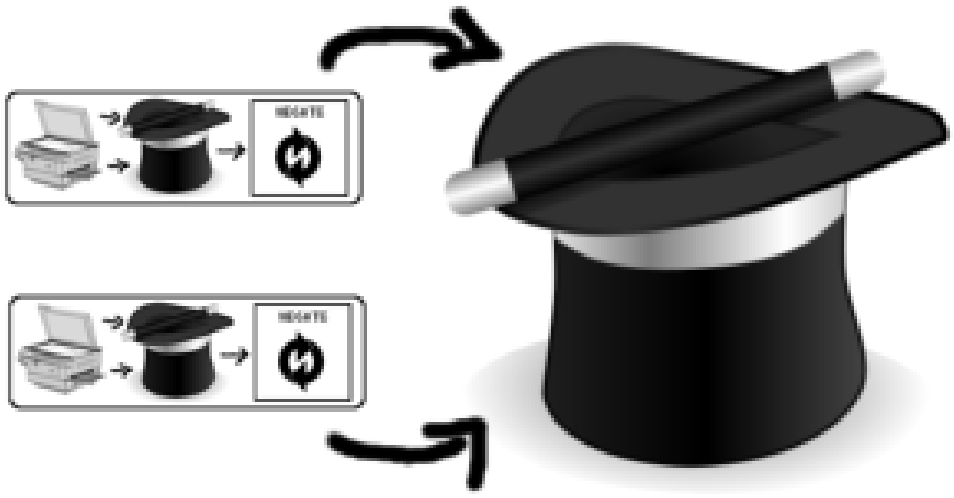


Insbesondere können wir diese Aneinanderreihung als *eine* *Maschine* betrachen. Nennen wir sie `X`.

Going Meta! (1)

Jetzt nur keine Panik bekommen!
Aber was passiert, wenn wir den Quellcode von `X` an `X` geben?

Going Meta! (2)



Der interessante Teil der Frage ist, was das Orakel ausspuckt.
Hier gibt es aber überhaupt nur zwei Möglichkeiten!

Fall 1: Orakel sagt "Hält!"

Entweder das Orakel sagt das `X` mit `X` als Inpt hält. Das geht aber nicht, weil dann der Negierer dafür sorgt, dass es *nicht* hält.

Fall 2: Orakel sagt "Hält nicht!"

Oder das Orakel sagt `X` mit `X` als Inpt hält *nicht*! Das bringt aber das gleiche Problem: dank des Negierers hält `X` eben doch!

Widerspruch

Was bleibt uns von unserer erdachten magischen Maschine?
Jeder mögliche Weg führt zum Widerspruch!

Widerspruch

Was bleibt uns von unserer erdachten magischen Maschine?
Jeder mögliche Weg führt zum Widerspruch!

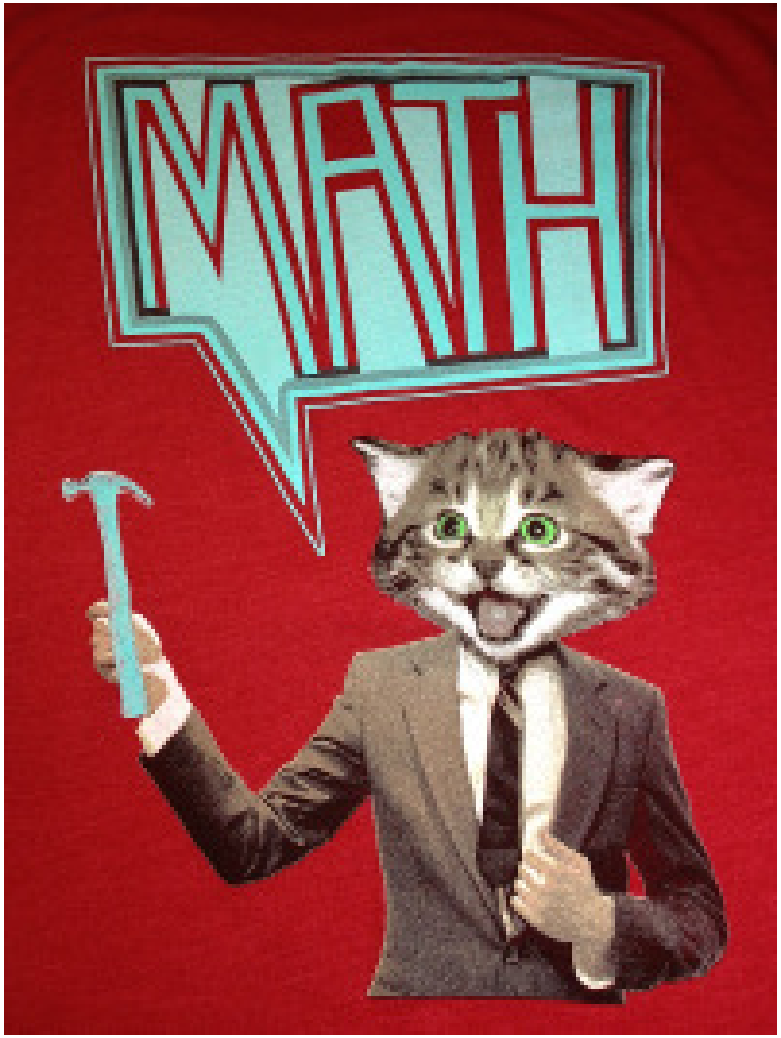
Damit ist bewiesen: so ein Orakel *kann* es nicht geben!
Das Halteproblem ist (im Allgemeinen) unlösbar!

Widerspruch

Was bleibt uns von unserer erdachten magischen Maschine?
Jeder mögliche Weg führt zum Widerspruch!

Damit ist bewiesen: so ein Orakel *kann* es nicht geben!
Das Halteproblem ist (im Allgemeinen) unlösbar!

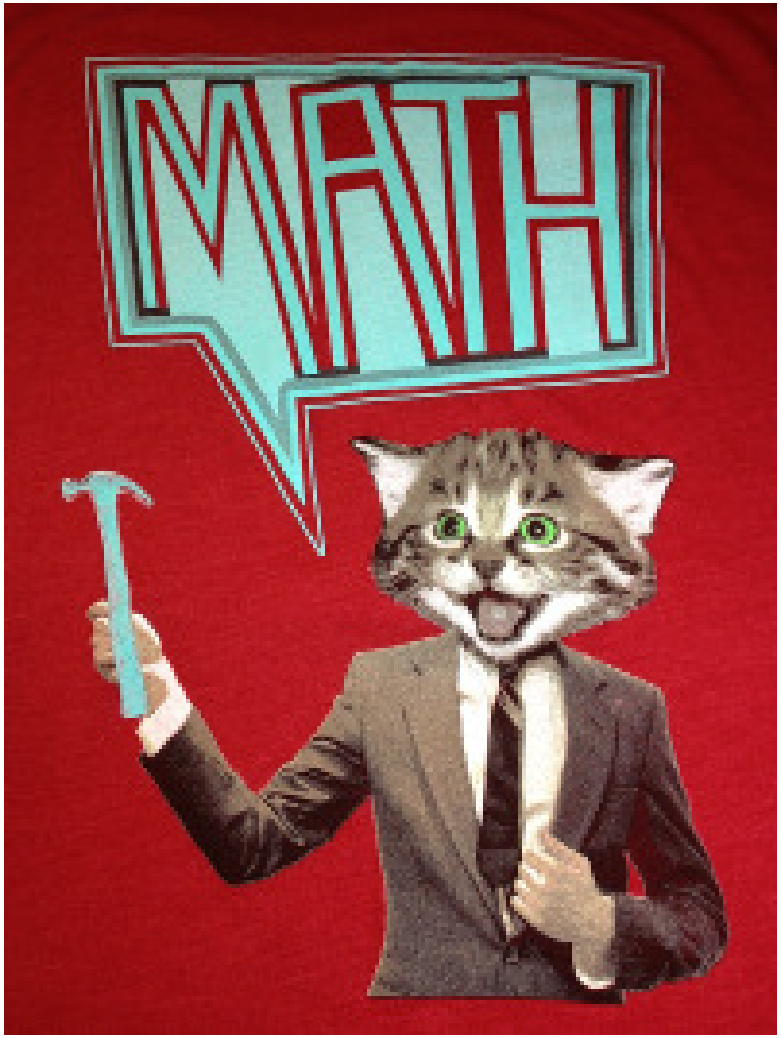
Schlusswort



Was haben wir heute gelernt?

„Red Math Cat“ by Jon Wye, Image under Fair Use

Schlusswort

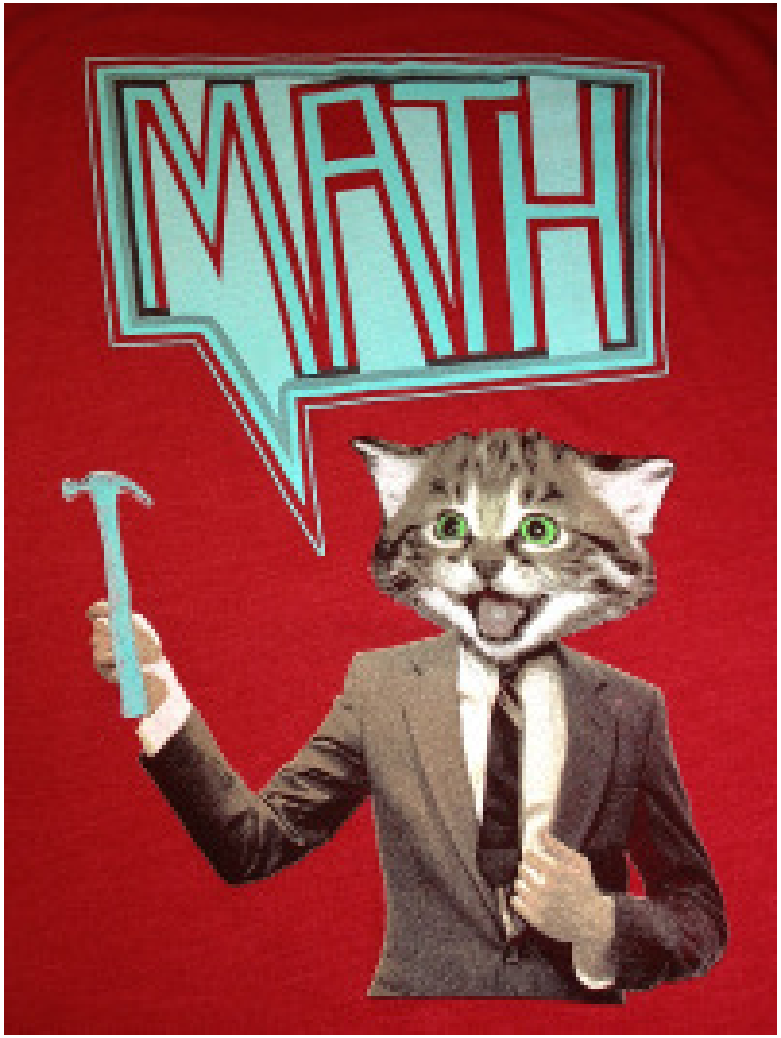


Was haben wir heute gelernt?

- Die Welt ist groß und ungewiss!

„Red Math Cat“ by Jon Wye, Image under Fair Use

Schlusswort

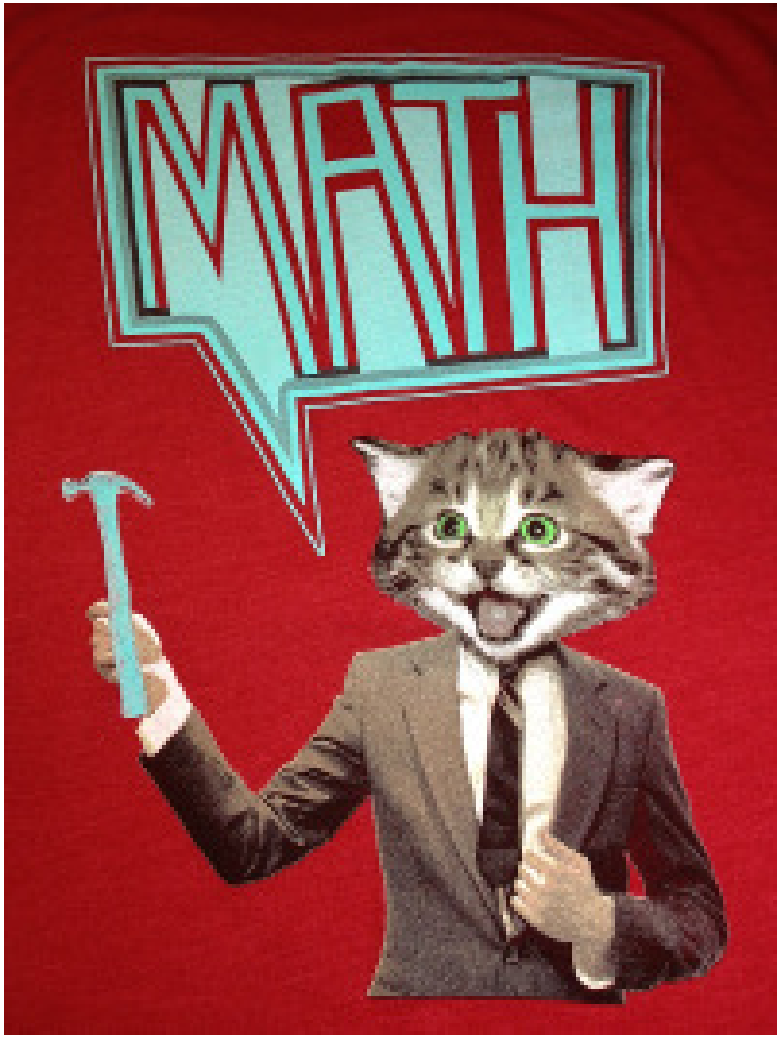


Was haben wir heute gelernt?

- Die Welt ist groß und ungewiss!
- Mathe macht Spaß!

„Red Math Cat“ by Jon Wye, Image under Fair Use

Schlusswort

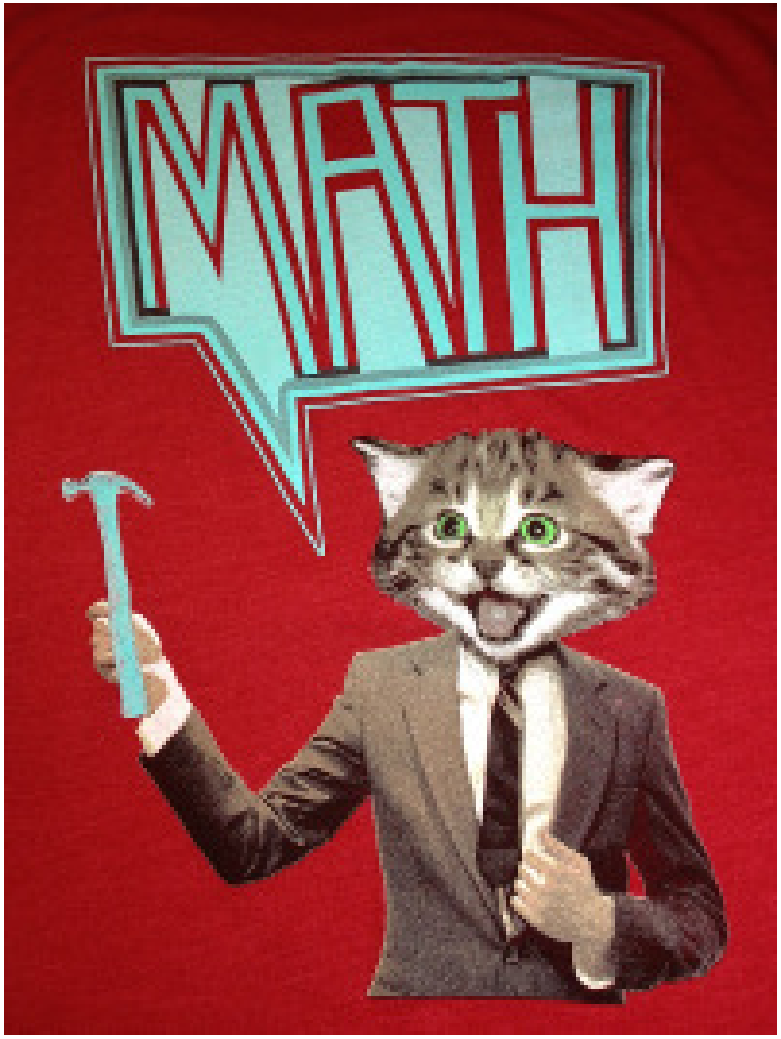


Was haben wir heute gelernt?

- Die Welt ist groß und ungewiss!
- Mathe macht Spaß!
- Wir können jetzt das Halteproblem erklären und wissen, dass es keine allgemeine Lösung gibt.

„Red Math Cat“ by Jon Wye, Image under Fair Use

Schlusswort



Was haben wir heute gelernt?

- Die Welt ist groß und ungewiss!
- Mathe macht Spaß!
- Wir können jetzt das Halteproblem erklären und wissen, dass es keine allgemeine Lösung gibt.

Bitte bleiben
Sie neugierig!

„Red Math Cat“ by Jon Wye, Image under Fair Use