"Hilberts Albtraum" (The answer may *not* be out there!)

@jbetzend

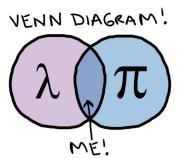




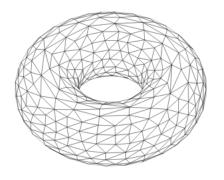


Mein Thema

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



[&]quot;Venn Diagram" by unknown artist, modified, Image under Fair Use



Mathematik ist eine Wissenschaft fast wie jede andere...

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

GIVEN THE PACE OF TECHNOLOGY, I PROPOSE WE LEAVE MATH TO THE MACHINES AND GO PLAY OUTSIDE.



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.

extract_number_and_incr (destination, source) int

"destination; unsigned char ""source; (entract, namber (destination, "source); "source += 2; | tilfndef EXTRACT_MAC-BOS Bunder EXTRACT MUMBER AND INCR Adering EXTRACT MUM. BER AND INCR0fest syd Lestract number and incr0kdest Bust Bendif /* not EXTRACT. MACROS */ #endif /* DEBUG */30/* HIDEBUG is defined. Repex prints many voluminous messages about what it is doing lift the variable 'debug' is nonzero). If linked with the main program in 'iregex.c', you can enter patterns and strings interactively. And if linked with the main program in 'main,c' and the other test files, you can run the already-written tests. */ ##def DEBUG /* We use standard UO for debusping. */ #include < stdio.ho-/"It is useful to test things that "must" be true when debugging. "/ #include -assert to-static int. debug = 0; #define DEBUG_STATEMENT(e) o #define DEBUG_PRINT1(x) if (debug) printf (x) #define DEBUG_PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf (x1, x2, x3) #define DEBLIG_PRINT4(x1, x2, x8, x4) if (debug) printf (x1, x2, x3, x4) #define DE BUG_PRINT_COMPLED_PATTERNIp, s, e)/, if (debug) print_partial_compiled_pattern (s, e) #define DE-BUG PRINT DOUBLE STRING'W ct. cd. c2. c2) \ \ i' (debug) print double ching (w. ct. cd. c2. c2) extern void printchar(); /* Print the fastmap in human-readable form. */ void print. fastmap (fastmap) char *fastmap (unsigned was, a, range = 0; unsigned i = 0; while () < (1 << BiTEWIDTH) (if |fastmap(i++)) [was_a_range = 0; printchar () - 1]; while () < (1 << EFTEWEDTH) && fastmap(i) (was_a_range = 1; i++;) if leas, a range) (printf (**) mintchar (i - 1),)) putchar (inf);) (* Print a compiled pattern string in human-readable form, starting at the START pointer into it and ending just before the pointer END. */ void print_partial_compiled_pattern (start, end) unsigned char "start; unsigned char "end; (int mont, mont2; unsigned char 'p = start; unsigned char 'pend = end; if (start == NULL) | printf ("Inull(or"); return;)." Loop over pattern commands. "/ while (p < pend) (switch live, opcode, f) "p++) (case no_ap: printf ("/no_ap"); break case exacts: mont = "a++: printf ("/exacts/%d"; mont); do (putcher f/% print/har ("p++);) while (-mont); break case start, memory mont = "p++; printf ("/start, memory/%d/%ef, mont. *p++); break; case stop: memory: mont = *p++; printf (*/stop: memory/%d/%d? mont, *p++); break, case displicate: printf ("/duplicate/%d", "p++); break; case anyther: printf ("/anechar"); break case charset case charset not I register intic printf ("Abstratlist" (re-opcode til fla-1) == charset_not 1*_not*: "]; assert (p + *p < pend); for (c = 0; c < *p; c++) (unsigned bit;</p> unsigned char map, byte = p(t + c) putchar (7); for thit = 0; bit < BYTEWDTH; bit++i if Imap byte & (1 << bitl) printchar ic " BYTEINDTH + bit(:) p += 1 + "p; break; I case beg line: printf ("/begline"); break; case endline: printf ("/endline"); break; case on, failure... jump: extract_number_and_incr (&mont, &p); printf ("/on_fallure_jump/0/%d", mont); break case on failure laten string lutter extract number and incr (femore, &e); exist! Clos. failure keep string jump/0/HdC mont; break case dummy failure jump extract number and incr (8mont, 8pt; printf ("Idummy, failure, sump/\$7%d"; mont); break; case push, dummy, failure: printf (")push, dummy, failure"; break; case may

be pop jump extract number and jino Benchi. Bot printf (Wraybe pop jump) (Med) morth, broak case pop, dakenjump extract number, and jino (Benchi, Bot), morth (Voop, balene, jump) (Med), morth broak case jump poet, alt extract, number, and jino Benchi, Bot), printf (Med). Manchmal ist es sehr leicht zu sehen, ob ein Programm jemals halten wird oder nicht:

```
-- 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:

```
-- Dieses Programm hält quasi sofort

main :: IO ()

main = print $ plus (3,4)

where

plus :: (Int, Int) -> Int

plus (x,y) = x + y
```

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

Manchmal ist es aber auch nahezu unmöglich!

```
-- 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)



Ein magisches Halte-Orakel (2)



Ein magisches Halte-Orakel (3)



Ein magisches Halte-Orakel (4)

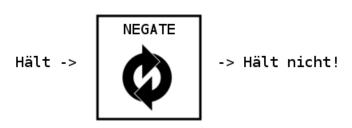


Negierer (1)



Dazu kommt ein Negierer, der ein Haltergebnis umdreht!

Negierer (2)



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)

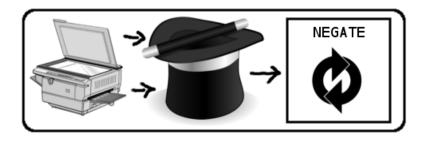
Wir haben auch einen Duplizierer, der seine Eingabe verdoppelt!

Unite and conquer! (1)



Sie ahnen es! Alles das können wir jetzt zusammen stecken!

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)



foo bar

Fall 1: Orakel sagt "Hält!"



foo bar

Fall 2: Orakel sagt "Hält nicht!"



foo bar

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