Het Halting Probleem
2025

Niet Beslisbare Problemen

Bestaan er, op het eerste gezicht toegankelijk problemen, waarvoor geen algoritme, dat het probleem oplost, mogelijk is?

Of anders gezegd: wat zijn de grenzen van wat je met een computer / algoritme kan doen?

Deze vraag is zinvol omdat we weten dat computermodellen die Turing volledig zijn, equivalent zijn. Dus als het met de ene taal/ machine niet kan, kan het nergens!

Niet Beslisbare Problemen

Het standaard voorbeeld van zo'n probleem is het Halting probleem (Stop probleem).

We zullen dit probleem in termen van Python formuleren.
Bekijk eerst de volgende 2 Python functies `stop(n)` en `loop(n)`
Het is duidelijk dat het eerste altijd eindigt en het tweede nooit:

```
def stop(n):
  return n

def loop(n):
  while n < n + 1:
    n = n +2
```

Niet Beslisbare Problemen

Maar stel dat we een onbekende Python f(n) hebben, waarvan we niet weten hoe deze in elkaar zit.

Dan kunnen we hooguit proberen voor verschillende waarden van n en kijken of de functie dan eindigt:

```
def test():
  for i in range(1000000):
    f(i)
  return True
```

Kunnen we op grond van dit iets over f(n) zeggen?

Niet Beslisbare Problemen

Nee, kies f(n) als volgt:

```
def f(n):
  if n !=  1234567:
    return n
  else:
    loop()
```

Niet Beslisbare Problemen

Maar zelfs als we in de definitie van een functie kunnen kijken is het lastig om iets te zeggen over de eindigheid van de functie.
Bekijk hiertoe de collatz functie:

```
def collatz(n):
  while n > 1:
    if n % 2 == 0:  # n even
      n = n /2
    else:
      n = 3*n + 1
  return True
```

Het vermoeden is dat dit altijd eindigt, maar of dit echt zo is, is een open wiskundig probleem

Halting Probleem

Bestaat er een functie `halt` die gegeven een andere functie `f` en een argument n besluit of `f(n)` termineert?

Uit het voorafgaande weten we dat deze vraag alleen zinvol is als de beschrijving van `f` kunnen inspecteren.

Laten we deze `rep_f` noemen.

De vraag is nu: bestaat er een functie `halt` zodat:

```
halt(rep_f,n) = if f(n) termineert: True else False
```

Halting Probleem

We kunnen de vraag zo stellen, omdat we al weten dat we in Python een interpreter voor Python kunnen maken.

Python is immers Turing volledig.

De `halt` functie moet in staat zijn `rep_f` de laten runnen!
We noemen deze functie `eval`:
`eval(rep_f,n)` gedraagd zich als `f(n)`

De eigenschap dat je in staat bent om binnen een formalisme programma's geschreven in het formalisme te executeren is **essentieel** in de formulering van het Halting probleem.

Halting Probleem

Het feit dat het Collatz probleem niet is opgelost doet ons vermoeden dat de halt functie wel niet zal kunnen bestaan.

Alleen in specifieke situaties valt er iets over het terminatie gedrag van een functie te zeggen.

Maar om te bewijzen dat het iha niet kan is niet eenvoudig!

Hiertoe moeten we een paradox creëren.

Paradoxen

In het bewijs creëren we een paradox vergelijkbaar met de volgende uitspraken:

Deze zin is niet waar!

Beroemd voorbeeld
Iemand is ter dood veroordeeld, maar mag zelf kiezen of hij wordt onthoofd of opgehangen. Hij mag een uitspraak doet. Als de uitspraak waar is wordt hij onthoofd. Als hij niet waar is wordt hij opgehangen.

Wat zou jij zeggen?

Paradoxen

Beroemd voorbeeld
Iemand is ter dood veroordeeld, maar mag zelf kiezen of hij wordt onthoofd of opgehangen. Hij mag een uitspraak doet. Als de uitspraak waar is wordt hij onthoofd. Als hij niet waar is wordt hij opgehangen.

Wat zou jij zeggen?

Ik word opgehangen!

Als hij wordt opgehangen had hij onthoofd moeten worden en omgekeerd

Bewijs van Halting: `halt` functie

Het bewijs is uit het ongerijmde. Dus we nemen aan dat de functie `halt` wel bestaat en we proberen tot een tegenspraak te komen. `halt` heeft dus de volgende eigenschap:

```
def halt(rep_f,n):
   if eval(rep_f,n) stopt: # f(n) stopt
      return True
   else:
      return False
```

Bewijs Halting: de paradox!

Definieer nu de functie p die als argument een representatie voor een functie f heeft en een argument:

```
def p(rep_f):
    if halt(rep_f,rep_f): # f(rep_f) termineert
        loop()
    else:
        stop()
```

Dus p(rep_f) termineert als f(rep_f) niet termineert en omgekeerd

Bewijs Halting: de paradox!

Als halt bestaat dan moet ook p bestaan.
In het bijzonder bestaat dan ook rep_p de representatie van p in Python. Bekijk nu:

```
p(rep_p)
```

Er zijn nu 2 opties:
`halt(rep_p,rep_p)` = True, maar dan eindigt `p(rep_p)` niet
`halt(rep_p,rep_p)` = False, dan eindigt `p(rep_p)` wel

In beide gevallen hebben we een tegenspraak vergelijkbaar met de leugenparadox!
Conclusie: `halt` kan klaarblijkelijk niet bestaan!

Post Correspondence Probleem

Eenvoudig voorbeeld van een onbeslisbaar probleem (probleem waar geen algoritme voor kan bestaan)
Domino steentjes met boven en onder een string.
Vind een ordening van de steentjes zo dat boven en onder dezelfde string staat of zeg anders dat het onmogelijk is.
Hierbij mogen steentjes meerdere keren gebruikt worden!

Zie beschrijving wikipedia

In het bewijs laat men zien dat het simuleren van een willekeurige Turingmachine gecodeerd kan worden mbv domino steentjes en dat het vinden van een correspondentie neer komt op de oplossing van het Halting probleem. Aangezien dat niet mogelijk is, is het ook niet mogelijk iha het Post probleem op te lossen!

Diophantische Vergelijkingen

Beschouw vergelijkingen van het type:

$a*x^n + b*y^m + c*z^k = 0$

met a b c n m k gehele getallen.

Bv $x^3 + y^3 + z^3 = 42$

Bestaan er een oplossing met gehele getallen x,y,z?

Nog onbekend voor 42 (geldt wel voor andere n < 100).

Diophantische Vergelijkingen

Er kan geen methode bestaan die voor een willekeurige Diophantische vergelijking kan zeggen of deze een oplossing heeft (Negatief antwoord op Hilberts 10e probleem)