Berechenbarkeit

Motivation

Computerviren, Würmer und Trojanische Pferde verursachen jährlich Schäden in Milliardenhöhe.

Typische Virenscanner finden nur bekannte Viren, aber keine Neuentwicklungen.

Kann man einen Virenscanner schreiben, der nicht syntaktische Details bekannter Viren sondern die Viruseigenschaft an sich erkennt?

Motivation

Allgemeiner:

> Kann man ein Analyseprogramm schreiben, das Programme daraufhin untersucht, ob sie gewisse Eigenschaften besitzen?

Mögliche Anwendungen:

> Untersuchung auf Viren,
> Terminierungsbeweise für Programme,
> Korrektheitsbeweise für Programme,
> Äquivalenzbeweise für Programme
>   (haben zwei Programme dasselbe Ein-/Ausgabeverhalten?)

Zunächst: Spezialfall Terminierung (Halteproblem).

Einschub: indirektes Beweisen

Januar 2001	Mo	Di	Mi	Do	Fr	Sa	So
	1	2	3	4	5	6	7
	8	9	10	11	12	13	14
	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30	31				

Heute ist der 18.01.2001.	$\Rightarrow$	Heute ist Donnerstag.
wahr	$\Rightarrow$	wahr
Heute ist der 04.01.2001.	$\Rightarrow$	Heute ist Donnerstag.
falsch	$\Rightarrow$	wahr
Heute ist der 15.01.2001.	$\Rightarrow$	Heute ist Montag.
falsch	$\Rightarrow$	falsch

Einschub: indirektes Beweisen

Wenn aus einer wahren Aussage durch korrekte Schlüsse
eine Aussage X folgt, dann ist X ebenfalls wahr.

Wenn aus einer Aussage Y durch korrekte Schlüsse
eine falsche Aussage folgt, dann ist Y ebenfalls falsch.

Indirektes Beweisen einer Aussage X:

> Nimm an, daß $\neg X$ gilt.
> Wenn man aus $\neg X$ eine falsche Aussage folgern kann,
> dann ist die Annahme $\neg X$ ebenfalls falsch.
> Also ist X wahr.

Halteproblem

Gibt es ein Java-Programm M0, das herausfindet, ob ein Programm x mit der Eingabe in der Datei y terminiert?

M0 soll folgende Eigenschaften besitzen:

> M0 wird aufgerufen als `java M0 x.java y`.
> M0 terminiert garantiert.
> M0 gibt entweder `ja` oder `nein` aus (und sonst nichts).
> `java x y` terminiert
> $\Rightarrow$ `java M0 x.java y` gibt `ja` aus.
> `java x y` terminiert nicht
> $\Rightarrow$ `java M0 x.java y` gibt `nein` aus.
> (Programmabsturz zählt als Terminierung.)

Halteproblem

Nehmen wir an, es gäbe ein derartiges Programm M0.
Dann können wir den Quellcode von M0 wie folgt abändern:
Wir fügen eine neue Klasse M1 hinzu:

```
public class M1 {
    static public void main(String[] arg) {
        String[] argNeu = new String[2];
        argNeu[0] = arg[0];
        argNeu[1] = arg[0];
        M0.main(argNeu);
    }
}
```

M1 wird aufgerufen als `java M1 x.java`.
M1 terminiert garantiert.
`java M1 x.java` liefert das gleiche Ergebnis wie `java M0 x.java x.java`.

Halteproblem

Wir ändern den Quellcode von M1, M0 (und den anderen davon benutzten Klassen) wie folgt ab: Vor jeder Ausgabeanweisung wie beispielsweise

```
System.out.println(s);
```

fügen wir ein:

```
if (s.startsWith("j")) {
    while (true) {
        // Endlosschleife
    }
}
```

Außerdem wird die Klasse M1 in M2 umbenannt.

M2 wird aufgerufen als `java M2 x.java`.
`java M1 x.java` gibt `ja` aus $\Rightarrow$ `java M2 x.java` terminiert nicht.
`java M1 x.java` gibt `nein` aus $\Rightarrow$ `java M2 x.java` terminiert.

Halteproblem

`java x y` terminiert ⇒ `java M0 x.java y` gibt `ja` aus.
`java x y` terminiert nicht ⇒ `java M0 x.java y` gibt `nein` aus.
`java M1 x.java` hat gleiche Ausgabe wie `java M0 x.java x.java`.
`java M1 x.java` gibt `ja` aus ⇒ `java M2 x.java` terminiert nicht.
`java M1 x.java` gibt `nein` aus ⇒ `java M2 x.java` terminiert.

Was passiert, wenn `java M2 M2.java` aufgerufen wird?

`java M2 M2.java` terminiert
⇒ `java M0 M2.java M2.java` gibt `ja` aus
⇒ `java M1 M2.java` gibt `ja` aus
⇒ `java M2 M2.java` terminiert nicht.

`java M2 M2.java` terminiert nicht
⇒ `java M0 M2.java M2.java` gibt `nein` aus
⇒ `java M1 M2.java` gibt `nein` aus
⇒ `java M2 M2.java` terminiert.

Halteproblem

Konsequenz:

`java M2 M2.java` terminiert
⇔ `java M2 M2.java` terminiert nicht.

Diese Aussage ist zweifellos falsch.

⇒ Unsere Annahme, daß es ein Java-Programm M0 mit den oben angegebenen Eigenschaften gibt, ist falsch.

⇒ Das Halteproblem ist unentscheidbar:
Es gibt kein Java-Programm M0, das herausfindet, ob ein Programm x mit der Eingabe in der Datei y terminiert.

Andere Programmiersprachen/Kalküle

Java/Pascal/C-Programme kann man in den Maschinencode eines konkreten Prozessors übersetzen bzw. von einem Maschinencode-Programm interpretieren lassen.

⇒ Alles was man in Java/Pascal/C berechnen kann, kann man prinzipiell auch in Maschinencode berechnen.

Die Funktionsweise eines konkreten Prozessors kann man mit einem Java/Pascal/C-Programm simulieren.

⇒ Alles was man in Maschinencode berechnen kann, kann man prinzipiell auch in Java/Pascal/C berechnen.

Andere Programmiersprachen/Kalküle

Konsequenz:

Alle üblichen Programmiersprachen haben die gleiche Ausdruckskraft.
Sie sind „berechnungsuniversell“.

Das Halteproblem für Maschinencode/Java/Pascal/C-Programme ist unentscheidbar.

Andere Programmiersprachen/Kalküle

Die Voraussetzungen für Berechnungsuniversalität sind gering:

while-Schleifen + if/then/else + Zuweisung
+ ausreichend viel frei-adressierbarer Speicher

(alternativ statt while-Schleifen + if/then/else: bedingtes goto)

Rekursive Funktionen (Parameter: beliebig große ganze Zahlen)
+ Addition + if/then/else
+ ausreichend viel Platz auf dem Stack

(alternativ statt ganze Zahlen/Addition: Strings/Konkatenation)

Andere Programmiersprachen/Kalküle

Aber: „saubere“ for-Schleifen statt while-Schleifen reichen nicht aus:

```
for (i = 1;  i <= n;  i++) {
  // i und n werden im Schleifenrumpf
  // nicht verändert.
}
```


Andere Programmiersprachen/Kalküle

Turingmaschinen

Bestandteile:

Unendlich langes Band („Turingband“), auf dem Zeichen aus einer endlichen Menge $\{a_0, \ldots, a_n\}$ stehen können;
a_0 ist ein Leerzeichen;
auf dem Band sind nur endlich viele Zeichen ungleich a_0.

Lesekopf: zeigt auf eine Stelle des Bandes;
kann dort das Zeichen lesen oder es überschreiben;
kann ein Zeichen weiter nach links oder rechts wandern.

Andere Programmiersprachen/Kalküle

Turingmaschinen

Bestandteile:

Programm: numerierte Liste von Befehlen.

Befehle:

- Schreibe das Zeichen a_i und gehe zu Befehl j.
- Bewege den Lesekopf ein Zeichen nach links und gehe zu Befehl j.
- Bewege den Lesekopf ein Zeichen nach rechts und gehe zu Befehl j.
- Falls der Lesekopf auf das Zeichen a_i zeigt, gehe zu Befehl j, sonst gehe zu Befehl k.
- Halte an.

Andere Programmiersprachen/Kalküle

Turingmaschinen

Alles was man in Maschinencode/Java/Pascal/C berechnen kann, kann man prinzipiell auch mit einer geeigneten Turingmaschine berechnen.

Turingmaschinen sind berechnungsuniversell.
(Andere Bezeichnung für „berechnungsuniversell“: Turing-vollständig)

Das Halteproblem für Turingmaschinen ist unentscheidbar.

Andere Unentscheidbarkeitsresultate

Gibt es ein Programm Nullcheck, das für jedes beliebige Programm P und jede Eingabe x bestimmt, ob P bei Eingabe x den Wert 0 ausgibt?

Antwort: nein.

Andere Unentscheidbarkeitsresultate

Der Beweis erfolgt durch *Reduktion* des Halteproblems auf das neue Problem:

Wir geben eine Transformation an, die zu jedem Programm P_1 ein Programm P_2 berechnet, so daß gilt:

P_1 hält bei Eingabe x an $\Leftrightarrow$ P_2 gibt bei Eingabe x 0 aus.

Die Transformation ist einfach: Streiche in P_1 alle Ausgabeoperationen und hänge einen neuen Befehl

```
System.out.println("0");
```

am Ende an.

Andere Unentscheidbarkeitsresultate

Angenommen, es gäbe ein Programm Nullcheck mit der oben angegebenen Eigenschaft, dann könnte man das Halteproblem dadurch entscheiden, daß man erst die Transformation durchführt und dann Nullcheck benutzt.

Das Halteproblem ist aber unentscheidbar $\Rightarrow$ Widerspruch.

Also kann es kein Programm mit dieser Eigenschaft geben.

Andere Unentscheidbarkeitsresultate

Satz von Rice:

Jedes Programm P berechnet eine partielle Funktion (Eingabe $\mapsto$ Ausgabe), die wir als f_P bezeichnen.

Sei E eine nicht-triviale Eigenschaft von partiellen Funktionen. (D.h., es gibt Programme P_0 und P_1, so daß f_{P_1} die Eigenschaft E hat und f_{P_0} sie nicht hat.)

Dann existiert kein Programm, das für ein beliebiges P berechnet, ob f_P die Eigenschaft E hat.

Andere Unentscheidbarkeitsresultate

Praktische Konsequenzen:

Es kann keinen absolut sicheren Virenscanner geben.

Berechnungsuniverselle Sprachen als Datenaustauschformate machen die sichere Analyse der Daten unmöglich (z. B.: Macrosprachen von Textverarbeitungen, Postscript, „self-extracting archive“).

Andere Unentscheidbarkeitsresultate

Postsches Korrespondenzproblem:

Gegeben: eine endliche Menge P von Paaren (s, t) von Strings.

Frage: Existiert eine Folge von Paaren $(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)$, so daß jedes Paar (s_i, t_i) in P enthalten ist, und so daß die Konkatenation der Strings $s_1, s_2, \ldots, s_n$ gleich der Konkatenation der Strings $t_1, t_2, \ldots, t_n$ ist?
(Paare aus P dürfen mehrfach verwendet werden.)

Andere Unentscheidbarkeitsresultate

Postsches Korrespondenzproblem:

Das Postsche Korrespondenzproblem ist unentscheidbar, d. h., es existiert kein Algorithmus, der für jede beliebige Menge P berechnet, ob eine derartige Folge von Paaren existiert.

Andere Unentscheidbarkeitsresultate

Eckdominoproblem:

Gegeben: eine endliche Menge D von quadratischen „Dominosteinen“ mit Beschriftungen an den vier Kanten. (Die Steine haben feste Orientierung; Umdrehen oder Rotieren sind nicht möglich.)

Frage: Kann man ausgehend von einem Eckstein d_0 einen kompletten Quadranten mit dieser Art von Dominosteinen parkettieren, d. h., kann man *beliebig weit* nach oben und rechts Steine so anlegen, daß die aneinandergrenzenden Seiten gleich beschriftet sind? (Es dürfen beliebig viele Duplikate der Steine aus D verbaut werden, aber keine anders beschrifteten Steine als die in D vorhandenen.)

Andere Unentscheidbarkeitsresultate

Eckdominoproblem:

Das Eckdominoproblem ist unentscheidbar, d. h., es existiert kein Algorithmus, der für jede beliebige Menge D und jedes beliebige $d_0 \in D$ ermittelt, ob eine Parkettierung des kompletten Quadranten möglich ist.

Andere Unentscheidbarkeitsresultate

Hilberts zehntes Problem:

Gegeben: eine ganze Zahl k und ein Polynom $p(x_1, \ldots, x_k)$ in k Variablen mit ganzzahligen Koeffizienten.

Frage: Existiert eine ganzzahlige Nullstelle von p, d. h., existieren irgendwelche ganzen Zahlen $n_1 \ldots, n_k$, so daß $p(n_1 \ldots, n_k) = 0$ gilt?

Hilberts zehntes Problem ist unentscheidbar, d. h., es existiert kein Algorithmus, der für jede beliebige k und p ermittelt, ob p eine ganzzahlige Nullstelle besitzt.

Andere Unentscheidbarkeitsresultate

Hilberts zehntes Problem:

Dies ist das zehnte einer Liste von 23 (damals) ungelösten Problemen, die der Mathematiker David Hilbert 1900 auf dem Internationalen Mathematiker-Kongreß in Paris vorstellte. Der Beweis wurde 1970 von dem damals 22jährigen Yuri Matiyasevich gefunden.