Selbsttestende und selbstkorrigierende Programme

Seminarvortrag Ergebnisprüfung 14.10.2003

Patrick Lauer

Einordnung

- Verifikation:
  - Nachteile:
    - sehr aufwändig
    - erfordert präzise Aufgabenstellung
    - schwer automatisierbar
    - kann Laufzeitfehler (Hardware, Compiler ...) nicht erkennen weil vor Programmlauf (offline) verifiziert wird
  - Vorteile:
    - exakte, garantierte Ergebnisse

Einordnung

- Tests:
  - Vorteile:
    - relativ einfach zu implementieren
    - meistens effizient (=schnell)
  - Nachteile:
    - nicht alle Werte werden getestet
    - Qualität der Tests ist abhängig von den Programmierern
    - (Tests können selbst fehlerhaft sein)

Einrichtung

- Ergebnisprüfung
  - nicht Implementation, sondern Ergebnis wird getestet
  - online (im Betrieb)
  - Vorteile:
    - resistent gegen Hardwarefehler, Compilerfehler
    - Implementation ähnlich komplex wie Tests
  - Nachteile:
    - Vergrösserung der Laufzeit um einen konstanten Faktor
    - (bzw. Hardwareaufwand grösser)

Ergebnisprüfung

- Ansatz:
  - Oft ist das Überprüfen eines Ergebnises einfacher als die Berechnung
  - Bsp.: Faktorisierung grosser Zahlen
  -
  - Berechnung: O(2^n) <-> Testen: O(n^2)
  -
  -
  -

Definitionen

- f(x) sei die zu berechnende Funktion mit einer Zeitkomplexität o(T(n))
- P(x) ist ein Programm, das f(x) für viele Werte aus dem Definitionsbereich berechnet
- C ist ein Programm, dass mit hoher Wkeit entscheiden kann, ob P(x) = f(x)

einfache Prüfer-Definition

-
- Eingabe: Wertpaar (x,y)
- Korrekte Ausgabe: y= f(x) = P(x) wird akzeptiert
- Zuverlässigkeit: ? (x,y) ist die Fehlerwahrscheinlichkeit ? p_c
- Little-o-rule: C ist zeitbeschränkt auf O(C) ? o(T(n))

einfache Prüfer-Beispiel

- Faktorisierung (angepasst)
  - P(x)=(y1,y2)
  - Eingabe: (x, (y1,y2)) wobei x = y1* y2 mit x,yi,k ? |N
  - korrekte Ausgabe: (x,(y1,y2)) wird akzeptiert ? x=y1*y2
  - Zuverlässigkeit: Korrekt mit W'keit nahe 1
  - little-o rule: O(C)= O(n^2) ? o(T(n)) (=O(2^n))

komplexe Prüfer

- Def. ähnlich wie einfache Prüfer
- darf aber P als Unterprogramm aufrufen
- Bsp.: selbsttestende Multiplikation
  - Annahme: Addition ist sicherer und schneller als Multiplikation
  - mehrfache Berechnung der Multiplikation erhöht die Sicherheit

komplexe Prüfer-Definition

- Eingabe: Wertpaar (x,y)
- Korrekte Ausgabe: y= f(x) = P(x)
- Zuverlässigkeit: ? (x,y) ist die Fehlerwahrscheinlichkeit ? p_c
- Little-o-rule: C ist zeitbeschränkt auf o(T(n)) (Aufrufe von P werden als O(1) betrachtet)
  - > Laufzeit wird um einen konstanten Faktor vergrössert

komplexe Prüfer / selbstkorrigierende Programme

selbsttestende Multiplikation: gesucht: y=(w*x)

(Fehlerw'keit z.B. 1/100)

- erzeuge 2 Zufallszahlen r1 und r2
- berechne y' = P(w-r1, x-r2) + P(w-r1, r2) + P(r1, x-r2) + P(r1,r2)
-
- wenn y=y' dann akzeptiere
-

selbstkorrigierende Multiplikation

-
- 4 Mult. --> 4/100 W'keit eines Fehlers unabhängig von der Eingabe
- 4 Mult. --> 4facher Aufwand
- durch wiederholte Iteration kann die Fehler w'keit beliebig reduziert werden

komplexe Prüfer-Beispiel

- Graph-Isomorphie
  - Problem: gegeben zwei Graphen A und B, sind diese isomorph?
  - Prüfer: Berechne P(A,B) (Ausgabe: Ja/Nein)
    - Wenn Ja: Wenn P(B,A) = Ja --> akzeptieren
      - alternativ: berechne expliziten Isomorphismus
    - Wenn Nein:
      - Wiederhole k-mal:
        - Wähle zufällig Permutation A' / B' von A oder B
        - Berechne P(A,A') oder P(B, B')
          - Wenn P(x,x') = Nein --> verwerfen

Graphisomorphie

- Eingabe: zwei Graphen A und B
- Korrekte Ausgabe: Ja wenn A ? B, nein sonst
- Zuverlässigkeit: ? (A,B) ist die Fehlerwahrscheinlichkeit ? 1/2^k bei k Tests
- Little-o-rule: C ist zeitbeschränkt auf k*O(P)

Graphisomorphie

- Korrektheit:
  - A ? B, P akzeptiert, B? A
  - A ? B, P akzeptiert --> P(B,A)=Ja hinreichend unwahrscheinlich
  - A ? B, P verwirft, Permutationen werden richtig akzeptiert/verworfen
  - A ? B, P verwirft --> k Permutationen werden jeweils mit W'keit 1/2 falsch erkannt --> W=1/2^k

Korrektheit von Prüfern

- Mögliche Zustände:
  - korrektes Ergebnis wird akzeptiert
  - falsches Ergebnis wird verworfen
  - korrektes Ergebnis wird verworfen
  - falsches Ergebnis wird akzeptiert
- nur der letzte Zustand ist problematisch!

Diskussion: Vor/Nachteile von Prüfern

- Mehraufwand?
- Präzisionsprobleme?
- Korrektheit des Prüfers?

Buggy checkers

- Nur das Akzeptieren einer inkorrekten Berechnung ist problematisch
- Komplexität der checker ist meist deutlich geringer als die der Programme
- z.B. Prüfer für Sortierprogramme fehlerresistent:
  - berechne h(x1)+h(x2) + ... = h(y1)+h(y2)+...
  - bei Fehler sehr unwahrscheinlich dass Hashwerte übereinstimmen

Variationen von Prüfen

- teilweise Tests (nur kritische Bereiche)
- Laufzeittests (Abschätzen des Aufwandes)
- Tests durch interaktive Beweise
  - Graphisomorphismen:
    - A ? B --> sukzessive Reduktion zeigt expliziten Isomorphismus
    - A ? B --> generiere Permutation C von A oder B
    - --> W'keit 1/2 dass richtiger Iso. gezeigt wird

Variationen von Prüfern

- angepasste Ein/Ausgabe
  - Sortierprogramm:
    - Eingabe Array von Zahlen x_1 ... x_n
    - Ausgabe Array von Tupeln (y_1, m_1), ... (y_n, m_n)
    - --> m ist die Position des Elements
    -
    - Prüfer: ist y_1 gleich wie das m_i-te Element der Eingabe?
    - --> konstanter Aufwand pro Element, linearer Gesamtaufwand

Variationen von Prüfen

- schwache oder seltene Tests
  - Bsp.: Suchprogram: Ist eine Zahl n in einer Menge enthalten?
  - Wenn nicht enthalten wähle zufällig ein Element aus der Menge und vergleiche es mit n
  - 1/n W'keit ein defektes Programm zu finden
- "Batch tests"
  - mehrere Ergebnisse auf einmal überprüfen kann Zeit sparen, aber Korrektur schwieriger

Variationen von Prüfern

- "teure" Selbstkorrektoren
  - im Falle eines Fehlers wird ein langsamer, einfacher Algorithmus benutzt
  - Bsp.: Matrixmultiplikation
    - bester bekannter Alg.: O(n^2.38)
    - naiver (fehlerfreier) Algorithmus: O(n^3)
    - Prüfer: O(n^2)

Überblick

- Prüfer sind eine vergleichsweise günstige Methode die Korrektheit eines Programmes zu testen
- Laufzeitvergrösserung, aber höhere Sicherheit
- einfache P. : testen Ergebnis unabhängig
- komplexe P.: benutzen Programm zum testen
- Korrektoren: erkennen Fehler und korrigieren

Fragen?

- ?