Karlsruher Institut für Technologie

Tutorium 17, #9, Weihnachtstutorium

Max Göckel– *uzkns@student.kit.edu*

Institut für Theoretische Informatik - Grundbegriffe der Informatik

Algorithmen

Ein Algorithmus ist eine Vorschrift um ein Problem zu lösen, dabei überführt ein Algorithmus eine Eingabe in eine Ausgabe.
Diese Vorschrift besteht dabei aus endlich vielen elementaren Einzelschritten, die dann zusammen den Algorithmus formen.

Man kann sich einen Algorithmus wie ein Kochrezept vorstellen das die Zutaten (Eingabe) in ein Endprodukt (Ausgabe) überführt.

Hoare-Kalkül: Grundlagen

Um die Korrektheit eines Algorithmus zu beweisen haben wir das *Hoare-Kalkül*.
Dies beweist die Korrektheit eine Algorithmus indem es Aussagen (sog. "Zusicherungen") über den Zustand der Ausführung macht.

Eine Aussage zeigt dabei die Korrektheit eines Algorithmus bis zu dem jeweiligen Punkt.

Hoare-Kalkül: Tripel

Ein Hoare-Tripel besteht aus zwei Zusicherungen und einem Stück Programmcode:

- {P}: Vorbedingung
  - P soll wahr sein bevor der Programmcode beginnt
- S: Codesegment
  - Ein Stück Algorithmus - eine oder mehrere Zeilen vom Code
- {Q}: Nachbedingung
  - Ist P wahr und wurde S ausgeführt so soll nun Q wahr sein.

Eine Hoare-Zusicherung hat also die Form $\{P\}S\{Q\}$

Hoare-Kalkül: Tripel

Für die Tripel gelten bestimmte Regeln (HT_1, HT_2, HT_3):

HT_1 Gilt $P' \rightarrow P \wedge Q \rightarrow Q'$ und ist $\{P\}S\{Q\}$ korrekt, so ist auch $\{P'\}S\{Q'\}$ korrekt

HT_2 $\{P\}S_1\{Q\}$ und $\{Q\}S_2\{R\}$ sind korrekt $\rightarrow \{P\}S_1S_2\{R\}$ ist korrekt

HT_3 Bei der Zuweisung $x \leftarrow \beta$ kann man aus der Nachbed. eine Vorbed. erstellen indem man x mit β substituiert

Hoare-Kalkül: HT_1

Wir probieren immer eine möglich ungenaue Vorbedingung und eine möglichst genaue Nachbedingung finden.
Wieso?

ngung Unser Algorithmus soll für möglichst viele Eingaben das richtige Ergebnis produzieren
$\Rightarrow$ Möglichst allgemeine Vorbedingung finden

ngung Über das Ergebnis der Überführung soll man so viele Aussagen wie möglich treffen können
$\Rightarrow$ Möglichst genaue Nachbedingung finden

Hoare-Kalkül: Beispiel

$S_1 = b \leftarrow a,$
$S_2 = c \leftarrow b,$
$Q = \{c = 5\}.$

$\{?_1\}$
S_1
$\{?_2\}$
S_2
Q

Finde $?_1, ?_2$ heraus.

Hoare-Kalkül: Beispiel

$S_1 = b \leftarrow a,$
$S_2 = c \leftarrow b,$
$Q = \{c = 5\}.$

$\{?_1\}$
$b \leftarrow a;$
$\{?_2\}$
$c \leftarrow b;$
Q

Finde $?_1, ?_2$ heraus.
$?_1 : \{a = 5\}$
$?_2 : \{b = 5\}$

Hoare-Kalkül: Aufgabe

$S = x \leftarrow (x - y) \cdot (x + y)$,
$Q = \{x + y^2 \neq 0\}$.

Bestimme die Vorbedingung P.

Hoare-Kalkül: Aufgabe

$S = x \leftarrow (x - y) \cdot (x + y)$,
$Q = \{x + y^2 \neq 0\}$.

Einsetzen (HT_3) in Q:
$\{(x - y) \cdot (x + y) + y^2 \neq 0\} \Leftrightarrow$
$\{x^2 - y^2 + y^2 \neq 0\} \Leftrightarrow$
$\{x^2 \neq 0\} = P$

Hoare-Kalkül: Verzweigungen

In Algorithmen gibt es auch Verzweigungen der Form `if B then S else T`. Diese können mit Hoare-Tripeln ausgewertet werden.

$\{P\}$ Verzweigung $\{Q\}$ wahr $\Leftrightarrow$ $\{P \wedge B\}S\{Q\}$ u. $\{P \wedge \neg B\}T\{Q\}$ wahr
d.h. für eine Verzweigung sind beide Fälle wahr wenn man die Bedigung im `if`-Teil umdreht.

Hoare-Kalkül: Schleifen

Neben Verzweigungen gibt es auch Schleifen. Wir reden hier nur von Schleifen der Form `while B do S`, wo das Tripel $\{I \wedge B\} S \{I\}$ gelten soll. Dabei ist I die Schleifeninvariante. Eine Schleifeninvariante ist eine Bedigung, die vor und nch jedem Durchlauf der Schleife gültig ist.

Gilt die Invariante I so gilt auch $\{I\}$ Loop $\{I \wedge \neg B\}$

Hoare-Kalkül: Schreibweise

if-then-else

$\{P\}$
if B then
    $\{P \wedge B\}$
    S
    $\{Q\}$
else
    $\{P \wedge \neg B\}$
    T
    $\{Q\}$
$\{Q\}$

while-do

$\{I\}$
while B do
    $\{I \wedge B\}$
    S
    $\{I\}$
$\{I \wedge \neg B\}$

Hoare-Kalkül: Beispiele

if-then-else

if $(x > y)$ then
    $z \leftarrow y$
else
    $z \leftarrow x$
Mit Eingabe a, b für x, y.

while-do

$s \leftarrow a$
$y \leftarrow b$
for $(i \leftarrow 0 \text{ to } b - 1)$ do
    $s \leftarrow s + 1$
    $y \leftarrow y - 1$

Graphen: Grundlagen

Ein Graph ist eine Struktur, die eine Menge von Elementen und eine Verbindung dieser Elemente darstellt.
Dabei ist V die Menge aller Elemente auf dem Graph (die Knoten, engl. Vertices) und E die Verbindung zwischen den Elementen (die Kanten, engl. Edges)

Formal:
Ein Graph G ist ein Tupel (V, E) mit V Menge der Knoten und E Menge der Verbindungen/Kanten.

Graphen: Gerichtet vs. Ungerichtet

Karlsruher Institut für Technologie

Gerichtet

- $G = (V, E)$
- $E \subseteq V \times V$
- E ist also Teilmenge des karth. Produktes und besteht aus Tupeln der Form (A, B)
- (A, B) heißt damit "von A nach B"
- Die Relation wird durch einen Pfeil $A \to B$ ausgedrückt

Ungerichtet

- $G = (V, E)$
- $E \subseteq \{\{x, y\} | x, y \in V\}$
- E Teilmenge einer Menge aus Mengen der Form $\{A, B\}$
- $\{A, B\}$ heißt damit "A und B sind verbunden"
- Die Relation wird durch einen Strich $A - B$ ausgedrückt

Beispielgraph

Figure: Ausweichfahrplan S5/S52 März 2017

Graphen: Teilgraphen

Ein Teilgraph ist ähnlich einer Teilmenge nur ein bestimmter Abschnitt vom Graph. Dabei ist egal ob der Teilgraph zusammenhängt (Alle Knoten verbunden sind) oder nicht.

Formal:
Ist $G = (V, E)$ ein gerichteter Graph so ist $T = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E \cap (V' \times V')$ ein Teilgraph von G.
Ist G ungerichtet so ist $E' \subseteq E \cap \{\{x, y\} | x, y \in V'\}$

Graphen: Wege und Pfade

Sind zwei Knoten mit einer Kante verbunden, so sind sie *adjazent*.

In einem gerichteten Graph ist ein Pfad von A nach B eine Liste aus zueinander adjazenten Knoten. Die Länge des Pfades ist dabei die Anzahl der Kanten zwischen den Knoten.

In einem ungerichteten Graph ist ein Weg von A nach B ein ungerichteter Pfad von A nach B.

- Gerichteter Graph $G = (V, E)$ *streng zusammenhängend* $\Leftrightarrow \forall A, B \in V \exists$ Pfad $A \to B$
- Ungerichteter Graph $G = (V, E)$ *zusammenhängend* $\Leftrightarrow$ G ist aus einem Stück

Graphen: Grade

Der Grad eines Knoten in einem gerichteten Graph ist die Anzahl der Pfeile die von ihm weggehen und zu ihm führen.

- Der Eingangsgrad $d^-(D)$ ist die Anzahl der Kanten die in D hineingehen
- Der Ausgangsgrad $d^+(D)$ ist die Anzahl an Kanten die aus D weggehen
- der Grad $d(D)$ ist $d^+(D) + d^-(D)$, also alle Kanten kombiniert

Für ungerichtete Graphen gibt es keinen Eingangs-/Ausgangsgrad, also ist der Grad $d(D)$ die Anzahl der Kanten in denen D vorkommt.

Adjazenzmatrizen

Man kann eine Graph nicht nur als Zeichnung darstellen, sondern auch als Matrix.
Diese Matrizen heißen Adjazenzmatrizen, sie sind immer quadratisch und gehören eindeutig zu einem Graphen.

An einer Adjazenzmatrix kann man die Kanten eines Graphen ablesen.

Adjazenzmatrizen: Aufbau

- Jede Zeile i steht für einen Knoten $k_i \in V$
- Jede Spalte j steht für die von k_i direkt erreichbaren Knoten
  - $1 \Leftrightarrow \exists \{k_i, k_k\} \in E$, es existiert eine Kante zwischen den beiden Knoten
  - $0 \Leftrightarrow \neg\exists \{k_i, k_k\} \in E$, es existiert keine Kante

Adjazenzmatrizen: Beispiel

Die folgende Beispielmatrix beschreibt einen Graphen mit 4 Knoten:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Ungerichtete Graphen

Bei ungerichteten Graphen ändert sich die Adjazenzmatrix nicht, die einzige Besonderheit ist dass sie symmetrisch ist.

Die folgende Beispielmatrix beschreibt den selben Graphen wie eben als ungerichteten Graph:

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Aufgaben

Bestimme den Graph anhand der Adjazenzmatrix:

$$X = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Lösung

$$A_C = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$$A_D = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$