

Info 2 Zusammenfassung für mündliche Prüfungen

Legende: von mir hinzugefügt, wahrscheinlich falsch

Prädikatenlogik

Allgemein

Mit Prädikatenlogik lassen sich Aussagen über Objekte machen. Mit Aussagenlogik nicht.

Bestandteile der Prädikatenlogik

Besteht aus Termen: Variablen (x, y, z) , Funktionssymbole $(f(x))$, Konstanten (Fs der Stelligkeit 0: a)

Zusätzlich: Prädikatensymbole $(P(x), Q(x))$, Quantoren (\forall, \exists) , logische Symbole (\neg, \wedge, \vee)

Variablen

Variablen können gebunden oder frei vorkommen. Gebunden werden sie durch Quantoren.

Struktur

Eine Struktur setzt sich zusammen aus Interpretation und Universum.

- Universum
 - Enthält die Grundmenge (zum Beispiel „Menschen“)
- Interpretation
 - Weist Prädikatssymbolen ein Prädikat zu
 - Weist Funktionssymbolen eine Funktion zu
 - Weist freien Variablen einen Wert aus dem Universum zu

Atomare Formeln und Formeln

- Atomare Formeln
 - Sind alleinstehende Prädikatssymbole und Funktionssymbole
- Formeln
 - Durch Junktoren verbundene atomare Formeln

Passend, gültig, erfüllbar etc.

- Passend
 - Eine Struktur A heisst zu einer Formel F passend, wenn in der

Interpretation jedem Prädikatensymbol ein Prädikat, jedem Funktionssymbol ein Symbol, jeder freien Variablen einen Wert aus dem Universum zugeordnet wird

- Wahr
 - Eine passende Struktur A ist wahr, wenn die Formel unter ihr erfüllbar ist: $A(F)=1$
 - Man spricht dann auch von einem Modell für F
- Gültig
 - Wenn für eine Formel F alle **passenden** Strukturen wahr sind, dann ist F gültig
- Erfüllbar
 - Ist eine Formel F für mindestens **eine** Struktur wahr, so ist die Formel erfüllbar
- Erfüllbarkeitsäquivalent
 - Gibt es für F eine Struktur, die wahr ist und für G eine Struktur, die wahr ist, dann sind F und G erfüllbarkeitsäquivalent. (Beachte den Unterschied zur Äquivalenz: Hier spreche ich nicht nur von **einer** Struktur, die für **beide** Formeln passt)
 - Zwei Formeln sind auch dann erfüllbarkeitsäquivalent, wenn beide kein Modell haben.
- Kongruent
 - Als kongruent bzw. semantisch äquivalent bezeichnet man zwei Formeln, wenn sie für jede passende Belegung den gleichen Wahrheitswert haben.
 - Haben zwei Formeln F und G keine Modelle, so sind sie kongruent
- Modell
 - $F \models G: \Leftrightarrow$ jede zu F und G passende Struktur, die ein Modell für F ist, ist auch ein Modell für G
 - Nicht ganz so formal: Für jede Belegung, für die F wahr ist, ist auch G wahr
- Tautologie
 - $\models F: \Leftrightarrow 1 \models F \Leftrightarrow F$ ist Tautologie

Unifikation

Bei Unifikation versucht man durch Substitution (sprich: Ersetzen von Variablen durch Terme (da Variablen auch Terme sind, ist es also erlaubt, Variablen durch Variablen zu ersetzen)) aus einer n -elementigen Literalmenge eine ein-elementige Literalmenge zu machen.

Beispiel:

$$\{P(x), P(f(y))\}_{[x/f(y)]}$$

$$\rightarrow \{P(f(y))\}$$

Allgemeinster Unifikator

- Das Beispiel ist der allgemeinste Unifikator. Man könnte in dem Beispiel auch wie folgt substituieren:

$$\begin{aligned} & \{P(x), P(f(y))\}_{[x/f(a)]} \\ & \rightarrow \{P(f(a)), P(f(y))\}_{[y/a]} \\ & \rightarrow \{P(f(a))\} \end{aligned}$$

- Das ist nicht mehr der allgemeinste Unifikator, da mehr als nötig unifiziert wurde und man somit nicht mehr zurück kann.

Normalformen

- Bereinigte Pränexform
 - Bereinigen: Keine Variable darf sowohl gebunden als auch ungebunden vorkommen
 - Pränex: Alle Quantoren stehen ganz vorne
 - Gebundene Umbenennung
 - Benennt man beim Bereinigen die gebundenen Variablen um, so ist die BPF äquivalent zur Ausgangsformel
- Skolemform
 - BPF liegt vor, sonst erst in BPF überführen, dann Existenzquantoren eliminieren.

Resolution

Mit Resolution weist man die Unerfüllbarkeit einer Formel nach. Schafft man dies nicht, so sagt das **nichts** über die Formel aus!

Um Resolution anwenden zu können, muss die Formel in Konjunktiver Normalform (KNF) vorliegen. Die UNDs trennen die Klauseln, die ODERs trennen die Prädikate in der Klausel durch „,“

Beispiel:

$$F = (P(x) \vee \neg Q(y)) \wedge \neg P(x) \wedge Q(y)$$

Dies ergibt folgende Klauseln:

1. $\{P(x), \neg Q(y)\}$
2. $\{\neg P(x)\}$
3. $\{Q(y)\}$

4. $\{\neg Q(y)\}$ 1ste und 2te Klausel verbunden
5. $\{\}$ 3te und 4te Klausel verbunden ergibt die leere Klausel

Somit ist gezeigt, dass F unerfüllbar ist und $\neg F$ ist eine Tautologie („immer war“)

O-Kalkül

Keine Zusammenfassung vorhanden. Am besten im Cormen nachlesen.

Dies wurde bisher gefragt:

- obere untere Schranke aufzeichnen & erklären
- Warum $f \in O(g)$? Grafik aufgezeichnet $(c \cdot g(n), f(n), n_0)$
- Warum ist $3n$ in $O(n)$? Weil man ja c z.B. 4 wählen kann
- O-Kalkül (Obere Schranke erklären..., n_0, c und Bild gezeichnet)
- Teta-Kalkül, Beispiel

Datentypen

Abstrakte Datentypen

- Keller (Stack)
 - First In, Last Out
 - Methoden:
 - push einfügen eines Elementes
 - pop entfernen des obersten Elementes
 - top liefert oberstes Element
 - length liefert die Länge des Stacks
- Schlange (Queue)
 - First In, First Out
 - Methoden:
 - insert einfügen eines Elementes
 - tail entfernen des letzten Elementes
 - head liefert letztes Element
 - length liefert Länge der Schlange

Hashing

- Prinzip
 - Um schneller auf die Daten zugreifen zu können wird ihnen jeweils ein Index zugeordnet. Damit man den Speicher gering halten kann, begrenzt man den Speicherplatz durch eine Hashfunktion:
Beispiel:
 $h(x) = x \bmod 7$ Nun habe ich 7 Plätze (0-6)
 - Was tun bei Kollision?
 - Verketteten

- An jedem Index ist eine List, so können dort mehrere Elemente abgelegt werden
- Sondieren
 - Element wird verschoben
 - Linear: $+j$
 - Quadratisch: $+j^2$
- Kollision 2. Ordnung
 - Soll ein Element an eine Stelle, wo bereits ein **kollidiertes** Element ist, so handelt es sich um eine Kollision zweiter Ordnung.

Graphen

- Allgemein
 - Graphen können gerichtet oder ungerichtet sein.
 - Sie lassen sich durch eine Adjazenzliste oder Adjazenzmatrix darstellen (und im Rechner speichern)
- Halde (Heap)
 - Allgemein
 - Heap ist ein Baum, in dem das größte (maxHeap) oder das kleinste (minHeap) Element in der Wurzel steht.
 - Die Knoten sind von oben, von links nach rechts durchnummeriert.
 - Daher lässt sich ein Heap gut als Liste darstellen.
 - Man benutzt Heaps, wenn man schnell auf das Element höchster Priorität zugreifen möchte.
 - Unterschied zum Binärbaum
 - In einem Binärbaum stehen links vom Knoten nur kleinere, rechts vom Knoten nur größere Elemente. Dies kann im Heap nicht so sein, da das kleinste oder das größte Element in der Wurzel steht.
- Suchen im Graphen
 - Tiefensuche
 - Suche zunächst im Pfad des linken Kindes des Startknotens. Lässt sich mit STACK realisieren.
 - Breitensuche
 - Suche in beiden Kindern des Startknotens. Lässt sich mit QUEUE realisieren.
 - Kruskal
 - Sucht den minimal spannenden Baum in einem ungerichteten Graphen, dazu fügt er immer die günstigste Kante hinzu, sofern diese keinen Zyklus erzeugt.
 - Prim

- Wie Kruskal, nur dass Prim beim Knoten ganz links anfängt und immer die günstigste Kante zu einem Knoten hinzufügt, der noch nicht im Baum ist.
- Dijkstra
 - Sucht den günstigsten Weg vom Startknoten zum gesuchten Knoten und bildet dabei den minimal spannenden Baum in einem **gerichteten** Graphen.

Algorithmen

- Mögliche Eigenschaften
 - Terminierend
 - Der Algorithmus endet nach einer endlichen Anzahl an Schritten
 - Deterministisch
 - Der nachfolgende Schritt ist immer eindeutig festgelegt
 - Determinierend
 - Das Ergebnis des Algorithmus ist immer eindeutig
- Teile und Herrsche
 - Aufteilen von Problem in Teilprobleme. Dann Teilprobleme lösen
 - Beispiel: MergeSort
- Greedy
 - In jedem Schritt die beste Möglichkeit wählen
 - Beispiele: Kruskal, Prim, Dijkstra
- Dynamische Programmierung
 - Wie Teile und Herrsche werden Teilprobleme gelöst, diese dann aber in einer Tabelle gespeichert und bei Wiederauftreten des gleichen Teilproblems wird dieses nicht noch einmal gelöst, sondern die Lösung aus der Tabelle genommen.
 - Beispiele: Trellis, Viterbi
- Probabilistische Algorithmen
 - Sie sind zufallsgesteuert und lassen sich in folgende Arten unterteilen:
 - Monte Carlo Algorithmen
 - Sie terminieren immer, liefern allerdings nicht zwingend ein richtiges Ergebnis
 - Beispiel: Miller-Rabin (Primzahltest, der eine Zahl n immer wieder durch eine Zahl a zu teilen versucht, bis er Erfolg hat und dann sagt: „Ist keine Primzahl“. Wenn ihm dies nach endlich vielen Schritten nicht gelingt, dann sagt er, dass n eine Primzahl ist, was aber nicht bewiesen ist.
 - Las Vegas Algorithmen

- Sie terminieren nicht zwingend, aber wenn sie terminieren, dann liefern sie auf jeden Fall ein korrektes Ergebnis.
- Macao Algorithmen
 - Sie terminieren immer und liefern ein richtiges Ergebnis. Dieses kann aber mitunter sehr, sehr lange dauern. Daher kann man aus einem Macao schnell einen Las Vegas Algorithmus machen, indem man ihn einfach nach gewisser Zeit abbricht. Dann hat er entweder ein richtiges Ergebnis oder er hat kein Ergebnis.
 - Beispiel: RandomQuickSort (Pivotelement wird zufällig bestimmt)
- Sortieralgorithmen
 - SelectionSort
 - Sortieren durch Auswählen.
 - Liste wird nach dem kleinsten Element durchsucht und dieses wird dann mit dem ersten Element vertauscht. Danach wird das kleinste Element der Restliste gesucht und mit dem Element an zweiter Stelle vertauscht, und so weiter.
 - Der Aufwand ist dabei logischerweise recht hoch. Da jedes Element mit jedem verglichen wird, ist es $n*n$ also n^2 . Sowohl im Best als auch im Worst Case
 - InsertionSort
 - Sortieren durch Einfügen
 - Jedes Element wird mit seinem Vorgänger verglichen. Ist es kleiner, wird es mit ihm vertauscht. So „wandert“ jedes Element durch die Liste zu seinem Platz.
 - Im Best Case hat man hier eine sortierte Liste vorliegen, dann wird nichts verschoben und jedes Element wird einmal verglichen. Also $O(n)$
 - Im WorstCase ist die Liste falschrum sortiert, dann muss jedes Element verschoben werden.
 - Der Aufwand ist dann n^2
 - MergeSort
 - Sortieren durch Mischen
 - Die Liste wird geteilt und die beiden entstandenen Listen werden ebenfalls wieder geteilt und so weiter, so lange bis nur noch ein-elementige Teillisten vorliegen. Diese werden nun zusammengefügt und dabei sortiert, indem immer die ersten Elemente (also die kleinsten) der Teillisten miteinander verglichen werden.
 - Da die Listen immer weiter aufgesplittet werden, ergibt sich quasi ein Baum. Somit hat der MergeSort im Best und im WorstCase $O(n \log n)$

- QuickSort
 - Zuerst wird ein Pivotelement gewählt (beim probabilistischen RandomQuickSort geschieht dies zufällig)
 - Idealerweise ist das Pivotelement wertmäßig in der Mitte der Liste.
 - Nun werden alle Elemente, die kleiner sind als das Pivotelement in einer Liste und alle Elemente, die größer sind in einer anderen Liste gespeichert. Danach kommt das Pivotelement zwischen beide Listen und der Algorithmus macht rekursiv mit den beiden neuen Teillisten weiter.
 - Im WorstCase ist das Pivotelement immer das kleinste oder das größte Element der Liste.
 - So wird immer nur ein Element von der Liste abgespalten. Jedesmal müssen n Vergleiche gemacht werden für n Elemente. Also $O(n^2)$
 - Der BestCase ist, wie oben beschrieben, wenn man immer den Median wählt. So erstellt sich durch das Aufsplitten der Listen ein Baum $\rightarrow O(n \log n)$
- HeapSort
 - Man hat einen Heap. Dieser lässt sich gut als Liste darstellen.
 - Man vergleicht nun das Element an der Stelle i mit seinen beiden Nachfolgern, die an den Stellen $2i$ und $2i+1$ sitzen. Das kleinste dieser drei Elemente kommt an Stelle i .
 - Dies macht man mit jedem Element. Das ganze nennt sich MinHeapify.
 - Danach ist die Heapeigenschaft hergestellt: Jeder Knoten ist größer als seine beiden Nachfolger.
 - Auf diese Art weiß man, dass das kleinste Element in der Wurzel ist.
 - Dieses wird nun entfernt und in einer Liste gespeichert.
 - Danach wendet man wieder MinHeapify an um die Heapeigenschaften wieder herzustellen.
 - Und so weiter.
 - Im BestCase hat man bereits eine sortierte Heapliste, so dass man nur nach und nach die Wurzeln in die Liste speichern muss und fertig. Aufwand: $O(n)$
 - Im WorstCase hat man eine falschsortierte Heapliste. Somit hat man dann den Aufwand $O(n \log n)$
- Suchalgorithmen
 - Knuth-Morris-Pratt
 - Sucht in einem Text/String nach einem Wort/Teilwort.
 - Boyer-Moore
 - Wie KMP, allerdings wird hier geprüft, ob der Buchstabe

überhaupt im Wort vorkommt und wenn nicht, wird dieses direkt um die gesamte Wortlänge verschoben.

- Diverse Algorithmen
 - Euklidischer Algorithmus
 - Algorithmus zur Berechnung des ggT
 - Interessant ist dabei der rekursive Aufruf:
 - $ggT(a, b) = ggT(b, a \bmod b)$
 - Brute Force
 - zu Deutsch: Brutaler Zwang
 - Algorithmus geht jede Möglichkeit durch (zum Beispiel bei Passwörtern)
 - Dies ist logischerweise sehr ineffektiv

Objektorientierte Programmierung

- Allgemeines
 - Bei der OOP wird in Klassen programmiert, auf die aus dem Hauptprogramm/Hauptklasse (main) zugegriffen wird.
- Sinn der OOP
 - Dies hat den Vorteil, dass der Code einfach verändert werden kann.
 - Zudem kann er wieder verwendet werden.
- Die „5 Schlüsselbegriffe“
 - Klasse
 - Die Klasse ist quasi eine Schablone für die Objekte. In ihnen werden die Methoden und Variablen deklariert
 - Objekt
 - Sind die Instanzen der Klasse
 - Polymorphie
 - Zu Deutsch: Vielgestaltigkeit.
 - Sie besagt, dass eine Variable/eine Methode in verschiedenen Objekten für verschiedene Zwecke verwendet werden kann.
 - Wenn zum Beispiel die Unterklasse die Methode „hallo(int x)“ erbt, dann kann man sie in der Unterklasse so überschreiben, dass man sie danach mit einem Char füttern kann. (also „hallo(char x)“)
 - Kapselung
 - oder auch Geheimnisprinzip
 - Die Variablen in den Objekten werden als Private deklariert. Von außen kann man nicht auf sie zugreifen. Nur Methoden aus derselben Klasse können darauf zugreifen.

- Von außen greift man über get und set Methoden auf die privaten Variablen zu
- Vererbung
 - Eine Oberklasse kann ihre Eigenschaften (Methoden und Variablen) an die Unterklasse vererben.
 - Beispiel:
 - Man hat Anfang der 90er eine Adresskartei programmiert. Nun will man diese erweitern, damit man auch eMail-Adressen darin speichern kann.
 - Man hat also eine Klasse „Adressen“
 - Nun erstellt man eine Klasse „Adressen2005“. Diese lässt man alles aus „Adressen“ erben und ergänzt das Ganze um „eMail“
- Abstrakte Klasse
 - Eine abstrakte Klasse kann keine Objekte erzeugen. Sie kann lediglich Methoden und Variablen in sich haben, die sie an andere Klassen vererbt.
 - Beispiel:
 - Man hat eine Klasse „Fahrzeuge“. In ihr ist definiert, was ein Fahrzeug alles haben muss. Also Räder, Motor, etc.
 - Nun hat man eine Klasse „Auto“. Sie erbt die Fahrzeugeigenschaften und füllt sie mit Werten. Also 4 Räder, ein Motor, etc. Daraus lassen sich dann Objekte instanziiieren wie zum Beispiel „Auto1“
- UML
 - Steht für Unified Modelling Language ist von der OMG (Object Management Group)
 - Mit UML lassen sich Abläufe eines Programms gut darstellen

Fragen

Folgende Fragen sollte jeder beantworten können, der das oben beschriebene verstanden hat. Die Fragen wurden so oder so ähnlich in den mündlichen Informatik II Prüfungen im Jahr 2005 gefragt.

- Prädikatenlogik
 - Was ist Prädikatenlogik?
 - Erklären Sie die Syntax der Prädikatenlogik
 - Was ist Unifikation?
 - Was ist der allgemeinste Unifikator?
 - Was ist ein Prädikat?
 - Was ist ein Term?
 - Was ist eine Substitution?
 - Was bedeutet $P(x)$ in der Prädikatenlogik?

- Was ist „passend“ in der Prädikatenlogik?
- Was bedeutet $f(x)$?
- Schreiben Sie eine prädikatenlogische Formel auf
- Wie können Variablen auftauchen?
- Was ist die Disjunktive Normalform?
- Was ist die Skolemnormalform?
- Algorithmen
 - Erklären:
 - MergeSort (Teile und Herrsche)
 - Kruskal (Graphenalgorithmus)
 - Brute-Force
 - Primzahltest (Miller-Rabin)
 - Algorithmus für ggT -> Euklid
 - Knuth-Morris-Pratt (Zeichensuche)
 - Heapsort
 - Beispiel für Macao, Monte-Carlo, Las Vegas
 - Machen Sie eine Aufwandsanalyse bei InsertionSort
 - Erklären Sie das O-Kalkül und machen Sie ein Beispiel
- Wissensfragen
 - Was sind die Unterschiede von Baum und Graphen?
 - Was ist ein Zyklus?
 - Was sind Graphenalgorithmus?
 - Welche Arten von Probabilistischen Algorithmen gibt es?
 - Was bedeutet deterministisch?
 - Was ist ein Greedy-Algorithmus?
 - Was ist eine Datenstruktur?
 - Was ist ein Heap?
 - Wie speichert man einen Heap?
 - Wie ist die Vater-Sohn-Anordnung in einem Heap?
 - Überführen Sie eine Heap-Liste in einen Heap-Baum
 - Was ist der Unterschied zwischen einem Heap-Baum und einem Binärbaum?
 - Welchen Graphenalgorithmus gibt es?
 - Wie kann man einen Graphen speichern?
 - Wie funktioniert Tiefensuche?
 - Erklären Sie einen Suchbaum
 - Wie fügt man eine Zahl hinzu?

- Was sind ADT (abstrakte Datentypen)? Machen Sie ein Beispiel
- Wie geht Hashen? Machen Sie ein Beispiel
- Welche Kollisionen kann es beim Hashen geben?
- Was ist Teile-und-Herrsche? Geben Sie ein Beispiel
- Was ist dynamische Programmieren?
- Erklären Sie die Rekurrenz für Fibonaccizahlen
- Was ist ein Keller? Welche Operationen gibt es und was machen sie? Beispiel für „pop“
- OOP
 - Was ist OOP?
 - Was bedeutet Polymorphie in der OOP?
 - Was ist ein Attribut?
 - Was bedeutet Vererbung?
 - Gehört ein Objekt immer einer Klasse an?
 - Kennt ein Objekt seine Klasse und auch umgekehrt?
 - Nennen sie die 5 Schlüsselwörter in der OOP
 - Was ist Kapselung? Machen Sie ein Beispiel
 - Für was steht UML und was ist die Bedeutung? Machen Sie ein Beispiel

Aktueller Stoff

Entwurfsmuster (design patterns)

- Werden in der objektorientierten Software Entwicklung entgesetzt um immer wiederkehrende Probleme zu erkennen und zu lösen
- Vier Elemente eines Entwurfsmusters
 - Name (pattern name)
 - Problem (problem)
 - Beschreibung der Anwendung des Musters
 - spezifisches Entwurfs-Problem oder
 - Beschreibung von Klassen und Strukturen oder
 - zu erfüllende Bedingungen
 - Lösung (solution)
 - beschreibt abstrakt wie und mit welchen Mitteln das Problem zu lösen ist
 - Konsequenzen (consequences)
 - Ergebnisse und Kosten für die Anwendung des Musters

- Konsequenzen können sein
 - veränderter Speicherbedarf
 - verändertes Laufzeitverhalten
 - Veränderungen in der Implementierung
 - verbesserte Flexibilität, Erweiterbarkeit oder Portabilität
- Drei Arten von Entwurfsmustern
 - Erzeugende Muster (creational pattern): erzeugen Objekte
 - Beispiel: Singleton
 - Ein Objekt, das es nur einmal in einem System geben darf
 - Konstruktor checkt bei der Instanzierung, ob es es schon eine Instanz gibt, wenn ja, wird diese zurückgeliefert, ansonsten eine neu erzeugt
 - Strukturelle Muster (structural patterns): Einfügen von Objekten in größere Strukturen
 - Beispiel: Adapter
 - Erzeugt zu einer Klasse eine neue Klasse mit den gleichen Schnittstellen und adaptiert die Methoden der alten auf die neue Schnittstelle
 - Verhaltensmuster (behavioral patterns): definieren die Kommunikation zwischen Objekten
 - Beispiel: Iterator
 - Sammlung von Daten kann traversiert werden
 - Jedes Mitglied wird genau einmal besucht
 - Kapselt die genaue Struktur der Datensammlung → Geheimnisprinzip
- Antimuster (antipatterns)
 - Zeigen eine schlechte Lösung
 - Beschreiben den Weg von einer schlechten zu einer guten Lösung
 - Wertvoller als normale Entwurfsmuster, da sie
 - aus einer echten Erfahrung entstanden sind
 - als warnendes Beispiel dienen können
 - zeigen wie man einen Missstand von der Ursache her beheben kann