Eindtoets

Eindtoets

INTRODUCTIE

Deze eindtoets is bedoeld als voorbereiding op het schriftelijk tentamen van de cursus Concepten van programmeertalen en is te beschouwen als proeftentamen. Het is belangrijk dat u de eindtoets pas probeert te maken op het moment dat u denkt klaar te zijn met de tentamenvoorbereiding. Hebt u over dat laatste nog twijfels, bekijk dan nog eens de leerdoelen en bestudeer de samenvattingen om te ontdekken welke onderdelen u nog onvoldoende beheerst.

Toegestane hulpmiddelen

Tijdens het tentamen mag het cursusmateriaal *niet* geraadpleegd worden.

Tentamenduur

Een tentamen duurt drie uur. We adviseren u dan ook de eindtoets binnen een aaneengesloten periode van drie uur te maken.

Samenstelling

Het aantal opgaven, de moeilijkheidsgraad en de verdeling over de leerstof komen overeen met het tentamen. Het tentamen bestaat uit acht vragen, ongeveer twee vragen per blok van de cursus. Er kunnen echter vragen zijn die betrekking hebben op meerdere blokken.
Bij het tentamen wordt actieve kennis van Java en Haskell verondersteld. Er kunnen vragen zijn waarbij u programmacode moet geven. In dat geval is de syntaxis van de taal niet relevant, zolang de bedoeling wel duidelijk is. Van de andere talen die in de cursus aan de orde komen, wordt passieve kennis verondersteld. Er kunnen vragen komen waarbij u code moet kunnen interpreteren, maar er zullen geen vragen zijn waar u code moet geven.

Terugkoppeling

De antwoorden op de opgaven staan in de terugkoppeling. We willen echter benadrukken dat u het meeste leert als u eerst de opgaven maakt en pas daarna de antwoorden controleert.

Beoordeling

Het aantal punten dat u per opgave kunt behalen, staat bij die opgave vermeld. U kunt in totaal maximaal 100 punten halen. Voor een voldoende voor het tentamen moet u ten minste 55 punten behalen.

Studeeraanwijzingen
De studielast van deze eindtoets bedraagt circa 4 uur, inclusief het nakijken van de opgaven aan de hand van de terugkoppeling.

Opgaven

OPGAVE 1 (15 punten)

6 punten a Wat is dynamische typering? Wat is een relatief voordeel ten opzichte van statische typering? Hoe kan in Java dynamische typering min of meer worden afgedwongen?

3 punten b Wat is het verschil tussen een variabele in een objectgeoriënteerde taal als Java en in een functionele taal als Haskell?

Beschouw twee Haskell-functies f en g met parameters van het type *A* en *B* en een resultaat van het type *C*. De waardeverzamelingen van *A, B* en *C* zijn eindig. Het type van de functies is:

```
f :: (A, B) -> C
g :: A -> (B -> C) ofwel g :: A -> B -> C
```

NB De functie f is niet gecurryed en de functie g is wel gecurryed.

6 punten c Toon aan dat de waardeverzamelingen van de functies f en g evenveel elementen bevatten. Maak hiervoor gebruik van de formules voor de cardinaliteit van respectievelijk een cartesisch product en een functieruimte.

OPGAVE 2 (10 punten)

Een functie in Haskell behoort tot een van de volgende vier categorieën:
– de functie is monomorf
– de functie is overloaded
– de functie is parametrisch polymorf
– de functie is zowel overloaded als parametrisch polymorf.

Deel elk van de volgende Haskell-functies in bij een van deze categorieën. Motiveer uw antwoord.

3 punten a De functie (+) voor de optelling van numerieke getallen.

3 punten b De functie map voor het toepassen van een functie op een lijst.

4 punten c De functie (<=) voor het vergelijken van elementen die ordenbaar zijn.

OPGAVE 3 (10 punten)

6 punten a In de cursus worden vier verschillende vormen van polymorfie beschreven overeenkomstig de indeling van Luca Cardelli en Peter Wegner. Geef aan wat de vier vormen zijn en beschrijf kort elke vorm.

4 punten b In de cursus worden vier manieren genoemd om de loop van een programma te beïnvloeden. Noem twee manieren en beschrijf ze kort.

OPGAVE 4 (10 punten)

Gegeven zijn de volgende klassen Queue en Box<T> in Java:

```
public class Queue {
  private char[] elems;
  private int length, front, rear;

  public Queue(int capacity) {
    elems = new char[capacity];
  }
```

```
  public void add(char newelem) {
    // add element to the rear of the queue
    ...
  }

  public int remove() {
    // remove and return the front element of this queue
    ...
  }

  {
    length = 0;
    front = 0;
    rear = 0;
  }
}

public class Box<T> {
  private T object;

  public void save(T object) {
    this.object = object;
  }

  public T getObject() {
    return object;
  }

  public String toString() {
    return "Box: " + object;
  }
}
```

5 punten a In de twee klassen wordt parametrisatie toegepast. Leg uit wat het verschil in parametrisatie is tussen de twee klassen.

5 punten b Is er in de klasse Box sprake van een vorm van polymorfisme? Verklaar uw antwoord.

OPGAVE 5 (15 punten)

Gegeven is de volgende monitor in (pseudo-)Java voor een begrensde buffer die gebruikmaakt van de conditievariabelen nonfull en nonempty en van synchronized-clausules.

```
public class QueueMonitor {
  private class MessageBuffer {
    int size, front, rear;
    Message[] items;
  }

  public int capacity;

  private MessageBuffer buffer = new MessageBuffer();
  private Signal nonfull;
  private Signal nonempty;

  public QueueMonitor(int capacity)   {
    // we don't use items[0]
    buffer.items = new Message[capacity + 1];
    this.capacity = capacity;
    nonfull = new Signal();
    nonempty = new Signal();
  }
```

```
  public synchronized void sendmessage(Message newitem) {
    while (buffer.size == capacity)    // buffer full
      nonfull.sig_wait();
    buffer.size = buffer.size + 1;
    buffer.rear = buffer.rear % capacity + 1;
    buffer.items[buffer.rear] = newitem;
    nonempty.sig_signal();
  }

  public synchronized Message receivemessage() {
    while (buffer.size == 0)   // buffer empty
      nonempty.sig_wait();
    buffer.size = buffer.size - 1;
    Message olditem = buffer.items[buffer.front];
    buffer.front = buffer.front % capacity + 1;
    nonfull.sig_signal();
    return olditem;
  }

  {
    // initialisation of buffer
    buffer.size = 0;
    buffer.front = 1;
    buffer.rear = 0;
  }
}
```

2 punten a Waarmee is communicatie geïmplementeerd in QueueMonitor?

2 punten b Waarmee is wederzijdse uitsluiting geïmplementeerd in QueueMonitor?

In het vervolg van deze opgave vervangt u in deze klasse de synchronized-clausules en de conditievariabelen door semaforen. Maak hierbij alleen gebruik van de klasse Semaphore zoals deze is gedefinieerd in paragraaf 5.5 van leereenheid 9:

```
public class Semaphore {
  private int s;

  public Semaphore(int n) {
    if (n >= 0)
      s = n;
    else
      throw new IllegalArgumentException();
  }

  public synchronized void sema_wait() {    // operation P
    while (s <= 0) {
      try {this.wait();}
      catch (InterruptedException e) { }
    }
    s--;
  }

  public synchronized void sema_signal() {  // operation V
    s++;
    this.notify();
  }
}
```

5 punten c Geef de declaratie, creatie en initialisatie van de benodigde semaforen. Motiveer kort uw antwoord.

6 punten d Wijzig de methoden sendmessage en receivemessage zó, dat gebruik wordt gemaakt van semaforen in plaats van synchronized methoden en conditievariabelen.

OPGAVE 6 (10 punten)

5 punten a Er zijn twee belangrijke verschillen tussen 'echte' monitors en Java-monitors. Geef een beschrijving van een van de twee verschillen.

5 punten b Wat houdt Software Transactional Memory in?

OPGAVE 7 (15 punten)

5 punten a Wat zijn de essentiële kenmerken van een imperatieve programmeertaal?

5 punten b Geef een beschrijving van het typevolledigheidsprincipe.

5 punten c Leg uit waarom de taal Haskell aan het typevolledigheidsprincipe voldoet.

OPGAVE 8 (15 punten)

Gegeven is het volgende programmafragment in Scala, dat voorzien is van regelnummers:

```
1   val belowFirst = (xs : List[Int]) => {
2     val first = xs(0)
3     val isBelow = (y : Int) => y < first
4     for (x <- xs; if (isBelow(x)) yield x
5   }
6   ...
7   belowFirst(List(5, 1, 7, 4, 9, 11, 3))
8   // => List(1, 4, 3)
```

Het sleutelwoord yield geeft aan dat het resultaat van een berekening de waarde van x oplevert.

3 punten a Van de lokale variabele first op regel 2 is het type niet gedeclareerd. Welke techniek maakt dit mogelijk?

3 punten b Wat is de betekenis van het sleutelwoord val op regel 2?

3 punten c Wat is het type van de variabelen first op regel 2 en isBelow op regel 3?

3 punten d Welke techniek wordt op regel 4 in de for-lus gebruikt om iteratie tot stand te brengen?

3 punten e Leg uit wat een vrije variabele is en geef aan welke variabele vrij is op regel 3.

TERUGKOPPELING

Uitwerking van de eindtoets

1 a Dynamische typering is een mechanisme waarbij alleen de waarden (en dus niet de expressies en variabelen) vaste typen bezitten en waarbij de typecontrole plaatsvindt tijdens de verwerking van het programma. Een relatief voordeel ten opzichte van statische typering is de flexibiliteit bij het programmeren.
In Java kan dynamische typering worden afgedwongen door een cast in de programmacode op te nemen.

b Een variabele in een objectgeoriënteerde taal als Java is een veranderbare variabele. Het is een referentie naar een geheugenlocatie waarin een waarde is opgeslagen. Tijdens de uitvoering van het programma kan de waarde veranderen.
Een variabele in een functionele taal als Haskell is een onveranderbare variabele. Het is een vorm van wiskundige variabele die een willekeurige waarde kan aannemen.

c We noteren het aantal elementen van type A met $\#A$. De regels voor het aantal elementen van een cartesisch product en een functietype luiden respectievelijk: $\#(A, B) = \#A \times \#B$ en $\#(A \text{ -> } B) = \#B^{\#A}$.
Met behulp van deze regels berekenen we het aantal elementen van de niet-gecurryde functie f met twee parameters op de volgende manier:
$\#((A, B) \text{ -> } C) = \#C^{\#(A,B)} = \#C^{\#A \times \#B}$.
Het aantal elementen van de gecurryde functie g met twee parameters is:
$\#(A \text{ -> } (B \text{ -> } C)) = \#(B \text{ -> } C)^{\#A} = (\#C^{\#B})^{\#A}$.
Uit de rekenregels voor herhaald machtsverheffen $(x^y)^z = x^{y \times z}$ volgt dat de berekende aantallen gelijk zijn.

2 a De functie (+) is overloaded, omdat hij kan werken op de typen Int, Integer, Float, Double en Ratio, waarbij voor elk type een verschillende definitie geldt.
NB De typen zijn instanties van typeklasse Num waarin (+) is gespecificeerd. De typespecificatie luidt:

```
(+)  ::  Num a => a -> a -> a
```

b De functie map is parametrisch polymorf, omdat de functie op verschillende (functie)typen kan werken, terwijl er toch maar één definitie nodig is.
NB De typespecificatie luidt:

```
map  ::  (a -> b) -> [a] -> [b]
```

c De functie (<=) is zowel overloaded als parametrisch polymorf.
De functie is overloaded omdat er verschillende definities gelden voor het ordenen van bijvoorbeeld getallen, karakters en lijsten.
Alle standaardtypen in Haskell, uitgezonderd type IO en functietypen, zijn instanties van klasse Ord waarin (<=) is gespecificeerd.

De functie is tevens parametrisch polymorf omdat de definitie van ordening voor lijsten geldig is voor lijsten van een willekeurig elementtype. Het ordenen van lijsten gebeurt op een uniforme manier, ongeacht het type van de elementen: er wordt geteld hoeveel elementen de lijsten bevatten en deze twee getallen worden vergeleken.
NB De typespecificatie luidt:

```
(<=)  ::  Ord a => a -> a -> Bool
```

3 a Cardelli en Wegner onderscheiden ad-hoc-polymorfie, met coërcie en overloading, en universele polymorfie, met parametrische polymorfie en inclusiepolymorfie.
Coërcie is een impliciete, door de context afgedwongen, conversie van een waarde van een bepaald type naar een corresponderende waarde van een ander type.
Er is sprake van overloading als twee of meer *verschillende* operaties die op parameters van verschillend type werken, dezelfde functienaam of hetzelfde operatorsymbool hebben.
We spreken van parametrische polymorfie als een abstractie (functie) toepasbaar is – en op een uniforme wijze werkt – op een verzameling gerelateerde typen met een bepaalde structuur.
Inclusiepolymorfie is een vorm van polymorfie zoals die ontstaat bij overerving in objectgeoriënteerde talen. Een type kan subtypen hebben die operaties erven van het supertype.

b In de cursus worden vier manieren genoemd om de loop van een programma te kunnen beïnvloeden: sequencers, jumps, escape-opdrachten en exceptions met exception-handlers. We beschrijven hier de vier manieren.
Een sequencer is een constructie die de normale verwerkingsvolgorde van een programma onderbreekt door een lus (while-opdracht) of een vertakking (keuzeopdracht) in te brengen.
Een jump is een radicale sprong (goto-opdracht) naar een specifiek punt in een programma.
Een escape-opdracht wordt gebruikt om de executie van een lus of een procedure af te breken.
Een exception is een abnormale conditie tijdens de uitvoering van een programma. Een exception-handler is een stukje code dat uitgevoerd moet worden als een exception optreedt.

4 a De klasse Queue is geparametriseerd met betrekking tot de waarde van de variabele capacity. Er is sprake van één klasse, ongeacht de waarde van capacity.
De klasse Box is een generieke klasse die geparametriseerd is met betrekking tot het type object dat in een box wordt opgeslagen.
Instanties van klasse Box kunnen voor verschillende waarden van de klasseparameter T worden geïnstantieerd.

b Nee, er is geen sprake van polymorfisme. Er kunnen, zoals in a al gezegd, verschillende instanties van type Box worden gemaakt, maar elke keer betreft het een instantie van een monomorf type, bijvoorbeeld Box<Cirkel>, Box<Integer>, enzovoort.

5 a Communicatie is geïmplementeerd met de conditievariabelen nonfull en nonempty.

b Wederzijdse uitsluiting is geïmplementeerd met de synchronized-clausules.

c Voor het programmeren van de wederzijdse uitsluiting gebruiken we de semafoor lock – die we toepassen als binaire semafoor – en voor het programmeren van de communicatie de semaforen nonfull en nonempty. De semafoor lock wordt geïnitialiseerd met de waarde 1, opdat het eerste proces vrije toegang tot de kritieke sectie krijgt. De semaforen nonempty en nonfull krijgen respectievelijk de waarden 0 en de capaciteit van de buffer om de hele buffergrootte te kunnen gebruiken. De creatie en initialisatie doen we in de constructor van de klasse.

```
private Semaphore nonempty, nonfull, lock;

public QueueMonitor(int capacity) {
  buffer.items = new Message[capacity + 1];
  this.capacity = capacity;
  nonempty = new Semaphore(0);
  nonfull = new Semaphore(capacity);
  lock = new Semaphore(1);
}
```

d We verwijderen de synchronized-clausules in de methoden sendmessage en receivemessage en vervangen deze door aanroepen van sema_wait en sema_signal op de semafoor lock. De communicatie programmeren we met de semaforen nonfull en nonempty. Als semafoor nonfull > 0 is, is er plaats in de buffer en kan de zender een boodschap in de buffer plaatsen; als nonfull gelijk is aan 0, is de buffer vol en moet de zender wachten totdat de ontvanger een boodschap uit de buffer heeft gehaald en nonfull weer > 0 heeft gemaakt. Als semafoor nonempty > 0 is, kan de ontvanger een boodschap uit de buffer halen; als nonempty gelijk is aan 0, is de buffer leeg en moet de ontvanger wachten totdat de zender een boodschap in de buffer heeft geplaatst en nonempty weer > 0 heeft gemaakt. De methoden sendmessage en receivemessage gaan nu luiden:

```
public void sendmessage(Message newitem) {
  nonfull.sema_wait();     // nonfull = 0: buffer full
  lock.sema_wait();
  buffer.size = buffer.size + 1;
  buffer.rear = buffer.rear % capacity + 1;
  buffer.items[buffer.rear] = newitem;
  lock.sema_signal();
  nonempty.sema_signal();
}

public Message receivemessage() {
  nonempty.sema_wait();    // nonempty = 0: buffer empty
  lock.sema_wait();
  buffer.size = buffer.size - 1;
  Message olditem = buffer.items[buffer.front];
  buffer.front = buffer.front % capacity + 1;
  lock.sema_signal();
  nonfull.sema_signal();
  return olditem;
}
```

De aanroepen van sema_wait op lock mogen *niet* voor de aanroepen van sema_wait op respectievelijk nonfull en nonempty staan. In dat geval is het volgende scenario mogelijk. Thread-1 roept receivemessage aan, maakt semafoor lock gelijk aan 0 en gaat dan wachten op semafoor nonempty (die initieel 0 is). Even later roept thread-2 sendmessage aan en gaat wachten op semafoor lock (die dan immers 0 is). Beide threads hebben nu de status blocked.

6 a We geven hier een beschrijving van de twee verschillen:
– In een monitor kunnen verschillende conditievariabelen worden gedeclareerd met elk een eigen wait-queue. Op iedere conditievariabele kunnen de operaties *wait* en *signal* worden uitgevoerd, die dus werken met de wait-queue van die betreffende variabele. Een Java-monitor heeft slechts één enkele anonieme conditievariabele en slechts één wait-set. De operaties *wait* en *notify* worden aangeroepen op de monitor zelf en werken met de wait-set van de monitor.
– Een monitor kent wait-queues (wachtrijen); als gevolg van een signal-operatie wordt het proces *op kop van* een wait-queue hervat. Een Java-monitor heeft één wait-set (wachtverzameling); als gevolg van een notify-operatie wordt een *willekeurige* thread in de wait-set hervat.

b Software Transactional Memory (STM) is een nieuwe benadering voor het programmeren van parallelle processen met gemeenschappelijk geheugen. Deze benadering is geïnspireerd op database-transacties (met commit en rolback). STM maakt het mogelijk om de waarde van variabelen te veranderen via transacties. Het beheren van de concurrente toegang tot gemeenschappelijke variabelen wordt gerealiseerd door gebruik te maken van een transactional log.

7 a Als essentiële kenmerken van een imperatieve taal kunnen we noemen:
– Een imperatieve programmeertaal is gebaseerd op de architectuur van de hardware.
– Imperatieve talen kunnen hierdoor in principe efficiënt worden geïmplementeerd.
– Een programma in een imperatieve taal bestaat uit een rij opdrachten die in een voorgeschreven volgorde moeten worden verwerkt.
– Voor de opslag van waarden worden (veranderbare) variabelen gebruikt die geheugenplaatsen in de computer representeren.
– Er bestaat een toekenningopdracht voor het veranderen van de waarden van deze variabelen.
NB Als antwoord verwachten we dat u in ieder geval kenmerk 3 en 4 noemt om een voldoende voor deze vraag te kunnen scoren.

b Het typevolledigheidsprincipe stelt dat geen enkele operatie zonder speciale redenen beperkt zou moeten worden in de typen van de betrokken waarden. Anders gezegd: wat met waarden van één type kan, moet – voor zover zinvol – ook met waarden van een ander type kunnen.

c In Haskell zijn functies eersteklas waarden. Ze kunnen, net als alle andere waarden, gebruikt worden als argument van een functie of het resultaat zijn van een functie. Daarom voldoet Haskell aan het typevolledigheidsprincipe: er is geen beperking op het type van de parameters en van het resultaat.

8 a Door type-inferentie is de compiler in staat om het type van een variabele uit de context te extraheren.

b Het sleutelwoord val betekent dat de gedeclareerde variabele onveranderbaar (immutable) is.

c De variabele first heeft hetzelfde type als de elementen van de lijst xs (zie regel 1), dat wil zeggen het type Int.
De variabele isBelow is een functie van Int naar een boolean-waarde.

d Om iteratie tot stand te brengen, wordt in regel 4 een lijstcomprehensie gebruikt.

e Een vrije variabele is een variabele die in een expressie gebruikt wordt, maar buiten de expressie is gedefinieerd. Op regel 3 is dat de variabele first.