Hertentamen Geavanceerd Programmeren – 2007

18 April 2008 – 15:30u - 17:30u

- Dit tentamen is *gesloten boek*. Je mag dus geen dictaat, aantekeningen, *notebook* of andere hulpmiddelen gebruiken.
- Per opgave staat aangegeven hoeveel punten je kunt halen.
- Lees de opgaven nauwkeurig. Schroom niet vragen te stellen over eventuele onduidelijkheden in de opgaven.
- Geef precieze en compacte antwoorden. Meer tekst is zeker niet automatisch beter.
- Zet op elk vel je *naam* en *studentnummer*.

Succes!

1 Algemeen

8 pt. **1.a**) Geef de *kind* van de volgende type definities:

```
:: T1       = C11 Bool | C12 Int
:: T2 a     = C21 a (T2 a) | C21
:: T3 a b c = C31 (a → b → c)
:: T4 a b   = C41 a b b a
:: T5 a b c = C51 (a b) c
:: T6 a b c = C61 a (b c)
:: T7 a b c = C71 (a (b c))
:: T8 a b c = C81 a c | C82 (b c)
```

4 pt. **1.b**) Geef in maximaal 5 regels aan wat de rol van kinds is in een functionele taal als Clean.

4 pt. **1.c**) Geef in maximaal 5 regels aan wat Epigram toevoegd aan een functionele taal.

De balie

De rest van de opgaven in dit tentamen spelen zich af rond de balie van een congresbureau. Aan die balie kunnen twee acties plaatsvinden. **1)** Een persoon kan zich inschrijven voor een lijstje conferenties. **2)** Een persoon kan zich uitschrijven voor één conferentie. Indien gewenst mag je conferentie in je gedachten ook vervangen door tentamen, vak, of iets van dien aard.

In Clean types is dit:

```
:: Registration = In Person [Conf] | Out Person Conf
:: Conf         = AFP | TFP
:: Person       = Person String
```

derive `gEq Registration, Conf, Person`

Een mogelijke lijstje registraties is

```
[In (Person "Pieter") [AFP,TFP], Out (Person "Rinus") TFP]
```

Voor het versturen of opslaan van registraties is het handig deze om te kunnen zetten in een lijst van strings. In deze representatie worden alle constructoren weggegooid behalve die van de constructoren in het enumeratie type `Conf`. De constructoren uit `Conf` worden vervangen door een string met hun naam. Voor lijsten gebruiken we ook een speciale conversie; de lijst elementen worden gescheiden door een `","` en de lijst (ook een lege lijst) wordt afgesloten met een `"."`. De representatie van bovenstaande lijst registraties is dus:

```
["Pieter", "AFP", ",", "TFP", ".", "," ,"Rinus", "TFP", "."]
```

2 Generiek Programmeren

4 pt. **2.a**) Geef van de types `Conf` en `Registration` de generieke representatie in termen van:

```
:: UNIT       = UNIT
:: PAIR   a b = PAIR a b
:: EITHER a b = LEFT a | RIGHT b
:: CONS   a   = CONS a
```

Als er meerdere representaties mogelijk zouden zijn, mag je zelf wat kiezen.

4 pt. **2.b**) Geef generieke de representatie van de termen

```
In (Person "Pieter") []
Out (Person "Rinus") AFP
```

Als er keuze zou zijn, zorg dan dat die in overeenstemming is met het vorige onderdeel.

5 pt. **2.c**) We willen nu een functie `registration2strings :: [Registration] → [String]` die een lijst van registraties omzet in een lijst van Strings. We maken deze functie met behulp van de generische functie

generic `tt a :: a [String] → [String]`

Het tweede argument van `tt` is een continuatie zoals we die ook op het practicum hebben gebruikt. Deze continuatie bevat de lijst van Strings die moeten komen na de Strings die het huidige object representeren. Definieer de instanties van de generieke functie `tt` die je nodig hebt om de functie `registration2strings` te maken. Gebruik de ingebouwde generics van Clean en derive het algorithme voor types als dat mogelijk is.

2 pt. **2.d**) Definieer met behulp van de boven besschreven generieke definities de conversie van een lijst van registraties naar een lijst van strings: de functie `registration2strings :: [Registration] → [String]`.

3 Parsers

Definieer met behulp van parser combinators, zie appendix A, een functie die een lijst strings omzet in een lijst registrations: `strings2registrations :: [String] → [Registration]`. Gebruik hiervoor de volgende stappen. **Hint**: de meeste van deze onderdelen kunnen als one-liner, als je meer dan enkele regels code aan het schrijven bent is er zeker een handigere manier.

2 pt. **3.a**) Definieer een parser `pConf :: Parser String Conf` die uit de boven gegeven representatie als lijst van strings een `Conf` probeert te herkennen. Ter herinnering: de constructoren `AFP` en `TFP` worden gerepresenteerd als de strings `"AFP"` en `"TFP"`.

2 pt. **3.b**) Definieer een parser `pPerson :: Parser String Person` die een `Person` probeert te herkennen in de invoer.

4 pt. **3.c**) Maak een parser combinator `pList :: (Parser String x) → Parser String [x]` die een lijst van `x`-en probeert te herkennen in de gegeven lijst van strings. Deze `x`-en worden geparseerd door de parser die als argument aan de `pList` combinator gegeven wordt. Zoals boven uitgelegd worden de `x`-en gescheiden door het symbool `","` en wordt de lijst representatie wordt afgesloten met het symbool `"."`.

3 pt. **3.d**) Maak nu een parser voor het type `Registration`.

2 pt. **3.e**) Definieer hiermee een parser `pRegList :: Parser String [Registration]`. De registraties zijn hierin weer gescheiden door een komma-token (`","`) en de lijst wordt afgesloten met een punt-token (`"."`).

2 pt. **3.f**) Maak hiermee de functie `strings2registrations :: [String] → [Registration]`. Deze functie moet de inverse functie van `registration2strings` zijn. Gebruik de parser `eof` (zie appendix A) om te zorgen dat je alleen resultaten krijgt die de hele invoer consumeren.

4 iTasks

Met iTasks maken we een web-based interface om registraties te kunnen invoeren. Je mag in deze opgave aannemen dat alle benodigde instanties van de generische functies bestaan, die hoef je dus niet zelf op te schrijven. Zie appendix B voor een overzicht van de belangrijkste iTask combinatoren.

5 pt. **4.a**) Definieer een iTask `regT :: Task Registration` waarmee één registratie ingevoerd kan worden. Gebruik zoveel mogelijk de generieke componenten.

10 pt. **4.b**) Maak een iTask `regT :: Task Registration`, waarmee de gebruiker ook een registratie kan invoeren, maar op een andere manier. Eerst kiest de gebruiker via buttons voor de deeltaken `In` of `Out`. Binnen ieder van die taken wordt eerst naar de naam gevraagd zolang die leeg is. De `Out`-Task vraagt daarna om één conferentie. De `In`-Task vraagt om een lijst van conferenties. Zolang deze lijst leeg is of duplicaten bevat wordt er opnieuw om zo'n lijst gevraagd.

5 Model-based testen

Om de omzetting van Registrations van en naar strings te testen, kunnen we gebruik maken van het feit dat deze functies elkaars inverse zijn.

7 pt. **5.a**) Definieer een functie `propParser :: [Registration] → Bool` die Gastin staat stelt te testen of de omzetting van een lijst van registraties naar strings en terug inderdaad de identiteit is.

3 pt. **5.b**) Laat zien hoe je deze eigenschap kunt gebruiken om daadwerkelijk tests uit te voeren (geef dus een `Start` regel). Je mag hierbij aannemen dat de generatie van de datatypes die je nodig hebt gedefinieerd zijn, evenals alle andere generieke functies die er nodig zijn.

Appendix

A Parser Combinatoren

Je kunt in dit tentamen gebruik maken van de volgende parser combinatoren. Natuurlijk mag je zelf extra combinatoren definiëren als dat handig is.

```
:: Parser s r :== [s] → [(r,[s])]

fail    :: Parser s r
yield   :: r → Parser s r
satisfy :: (s → Bool) → Parser s s
symbol  :: s → Parser s s | == s

// Choice of parsers
(∨) infixr 4 :: (Parser s r) (Parser s r) → Parser s r        // non-deterministic
(\/) infixr 4 :: (Parser s r) (Parser s r) → Parser s r       // p2 only if p1 fails

// Sequencing of parsers
(/\) infixl 6 :: (Parser s (r → t)) (Parser s r) → Parser s t
(∧) infixr 6 :: (Parser s r) (r → Parser s t) → Parser s t
(/:\) infixr 6 :: (Parser s r) (Parser s [r]) → Parser s [r] // Build list

(@>) infixr 7 :: (r → t) (Parser s r) → Parser s t            // Insert function in parser
(<@) infixr 7 :: (Parser s r) (r → t) → Parser s t            // Apply function to result

(/>\) infixl 6 :: (Parser s r) (Parser s t) → Parser s t      // Retain right result
(/<\) infixl 6 :: (Parser s r) (Parser s t) → Parser s r      // Retain left result

|*| :: (Parser s r) → Parser s [r]                             // Kleene star
|+| :: (Parser s r) → Parser s [r]                             // one ore more times

eof :: Parser s Int                                           // succeeds if input is empty
eof = λl.if (isEmpty l) [(1,[])] []
```

B iTask Combinatoren

Deze appendix bevat een overzicht van de belangrijkste operatoren uit de iTasks bibliotheek.

```
editTask    :: String a              → Task a   | iData a           // create an editor

(=>>) infix 1 :: (Task a) (a→Task b) → Task b | iCreateAndPrint b // bind
return_V    :: a                     → Task a   | iCreateAndPrint a // lift value to a task

seqTasks    :: [(String,Task a)]     → Task [a] | iCreateAndPrint a // sequence of tasks

buttonTask  :: String (Task a)       → Task a   | iCreateAndPrint a // do task after button
chooseTask  :: [(String,Task a)]     → Task a   | iCreateAndPrint a // one task from list

// done as soon as SOME Task runs to completion
(−||−) infixr 3 :: (Task a) (Task a)→ Task a   | iCreateAndPrint a // both in any order
orTask2 :: (Task a,Task b) → Task (EITHER a b) | iCreateAndPrint a & iCreateAndPrint b

// both task in any order, done when all tasks run to completion
(−&&−) infixr 4 :: (Task a) (Task b)→ Task (a,b)| iCreateAndPrint a & iCreateAndPrint b

(<|) infix 6 :: (Task a) (a→Bool, a→[BodyTag]) → Task a | iCreate a // check condition
(!>>) infix 5 :: [BodyTag] (Task a)→ Task a   | iCreate a          // prompt

// the iData elements needed: pulldown (number visible,width) (item chosen,menulist)
:: PullDownMenu = PullDown (Int,Int) (Int,[String])
```

Met behulp van `editTask` kun je ook voor iData elementen als een pull down menus eenvoudig een editor maken. Bijvoorbeeld:

```
aTask = editTask "ok" (richting,snelheid)
where
    richting = PullDown (1,100) (0,["Noord","Oost","Zuid","West"])
    snelheid = PullDown (1,100) (1,["stop","langzaam","snel"])
```

De pull down menus voor richting en snelheid komen nu naast elkaar in de browser. De initiële waarden zijn `Noord` en `langzaam`. Met de instantie van `toInt` kun je uit zo'n pull down menu het nummer van het geselecteerde item halen, met `toString` de naam van dat item. Bijvoorbeeld:

```
bTask :: Task (Int,String)
bTask = aTask =>> λ(r,s).return_V (toInt r, toString s)
```