Tweede deeltentamen Grammatica's en Ontleden, 2 februari 2006, 9–12 uur

1. (a) In een reguliere taal L geldt:

er bestaat een	n	$: n \in \mathbb{N}$	:
voor alle	u, z, y	$: uzy \in L$ en $\lvert z\rvert \geq n$	:
er bestaan	v, w, x	$: z = vwx$ en $\lvert w\rvert > 0$	:
voor alle	$i \in \mathbb{N}$	$: uvw^i xy \in L$	

   Leg uit waarom dit zo is. Geef daarbij onder andere aan hoe n gekozen kan worden, en waarom $|w| > 0$ is.

   (b) Deze stelling wordt meestal in zijn contrapositieve vorm gebruikt ('met omgekeerde implicatie'). Hoe luidt de stelling dan?

   (c) *In deze opgave is het van belang dat je je bewering ook bewijst. Alleen ja/nee is niet voldoende!*

   - Is $\{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}\}$ een reguliere taal? Bewijs het antwoord.
   - Is $\{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}\}$ een contextvrije taal? Bewijs het antwoord.

   (d) De volgende stellingen mag je in deze opgave gebruiken; je hoeft ze dus niet zelf te bewijzen:

   i. $\{\mathsf{a}^n \mid n \in \mathbb{N}\}$ is regulier en contextvrij.
   ii. $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \in \mathbb{N}\}$ is niet regulier en ook niet contextvrij.
   iii. De vereniging van twee reguliere talen is ook regulier.
   iv. De concatenatie van twee reguliere talen is ook regulier.
   v. De vereniging van twee contextvrije talen is ook contextvrij.
   vi. De concatenatie van twee contextvrije talen is ook contextvrij.

   Beantwoord nu de volgende vragen:

   - Is de doorsnede van twee reguliere talen ook regulier? Bewijs het antwoord.
   - Is de doorsnede van twee contextvrije talen ook contextvrij? Bewijs het antwoord.

2. Om te bepalen of een grammatica de LL(1)-eigenschap heeft, kan de grammatica worden geanalyseerd met behulp van de functies *lookaheadset*, *firsts*, *follow*, en *empty*.

   (a) Wat is het verschil tussen de begrippen *lookaheadset* en *firsts* ?

   (b) Beschrijf in termen van deze functies wanneer een grammatica de LL(1)-eigenschap heeft. *(Dat kan met een formule, maar het mag ook in woorden. Je kunt het in één zin formuleren, maar let extra goed op de precieze formulering!)*

   (c) Wat is het voordeel bij het schrijven van een parser dat een grammatica de LL(1)-eigenschap heeft?

   (d) Beschrijf het algoritme om de *lookaheadset* te bepalen als je de *firsts*, *follow*, en *empty* al weet. (Mag in woorden).

   (e) Zowel het LL- als het LR-ontleedalgoritme maken gebruik van een *stack*. Beschrijf het verschil tussen deze twee algoritmes voor wat betreft de *initialisatie* en het *stopcriterium*.

   (f) Zowel het LL- als het LR-algoritme selecteren af een toe een regel van de grammatica. Beschrijf het verschil tussen deze twee algoritmes in wat ze met de geselecteerde regel doen. (*Hoe* die selectie gebeurt is ingewikkeld, maar dat is in deze opgave niet van belang).

3. We bekijken een expressie-georiënteerde taal waarin integer- en boolean-expressies gebruikt worden. Het startsymbool van de grammatica is integer-expressie. De vorm van ontleedboom van een expressie wordt gegeven door:

```
data IntExp  = Con Int
             | Var String
             | Add IntExp  IntExp
             | If  BoolExp IntExp IntExp
             | Let String  IntExp IntExp   -- bijv. let x=2+3 in x+x
data BoolExp = Equal IntExp   IntExp
             | And   BoolExp  BoolExp
```

Gegeven is verder het type `Env`, en een bijbehorende opzoek-operator:

```
type Env = [(String,Int)]
((x,n):ts) ? y | x==y = n
               | otherwise = ts ? y
```

(a) Definieer een type `ExpAlgebra` die het type geeft van een semantiek voor dit soort expressies.

(b) Definieer het type van de functie `foldExp` die aan de hand van zo'n algebra de semantiek van een ontleedboom bepaalt. *Je hoeft alleen het type te definiëren, dus de regel met* `::`*, niet de implementatie van de functie!*

In de rest van deze opgave mag je de functie `foldExp` aanroepen. In elke deel-opgave moet je een algebra schrijven, en een functie die daar gebruik van maakt. Het is niet toegestaan om de algebra te vermijden door de recursieve tree-walks helemaal uit te schrijven.

Geef van de functies en algebra's die je zelf moet schrijven steeds het type (met `::`) én de definitie (met `=`).

(c) Definieer een functie `vars` die de lijst oplevert van alle variabele-namen die in een integer-expressie daadwerkelijk gebruikt (dus niet alleen maar gedefinieerd) worden.

(d) Definieer een functie `eval` die de waarde van een integer-expressie uitrekent.

(e) Definieer een functie `code` die code genereert voor de Simple Stack Machine (SSM, zie bijlage voor de details), zodat wanneer deze code gerund wordt de waarde van de expressie op de stack achterblijft. Je mag aannemen dat alle variabelen in de expressie door een `Let`-constructie worden gedefinieerd, met één uitzondering: de waarde van de variabele `Var "x"` is run-time te vinden in het geheugen, op de plaats waar de Markpointer naar wijst. Je mag verder aannemen dat zowel de Markpointer als de Stackpointer runtime naar een voldoende groot stuk geheugen wijzen.

(f) Wat moet je in de algebra van de vorige opgave aanpassen, om ervoor te zorgen dat de `And`-operator *lazy* wordt uitgerekend, dat wil zeggen dat zijn rechter operand niet wordt uitgerekend als dat niet nodig is? (Of, als je code al lazy was, om hem juist *eager* uit te rekenen).

Bijlage: de instructieset van de SSM

```
data Instr
 = STR Reg | STL Int  | STS Int  | STA Int  -- Store from stack
 | LDR Reg | LDL Int  | LDS Int  | LDA Int  -- Load on stack
 | LDC Int | LDLA Int | LDSA Int | LDAA Int -- Load on stack
```

```
 | BRA Int | BSR Int  | BRF Int  | BRT Int  -- Branch always / to subroutine / on fals
 | Bra String | Bsr String  | Brf String  | Brt String -- Idem, to label
 | ADD | SUB | MUL | DIV | MOD                    -- Arithmetical operations (2 stack
 | EQ  | NE  | LT  | LE  | GT  | GE              -- Relational   operations (2 stack
 | AND | OR  | XOR | NEG | NOT                  -- Other operations (2 or 1 stack op
 | RET | UNLINK | LINK Int | AJS Int            -- Procedure utilities
 | SWP | SWPR Reg | SWPRR Reg Reg | LDRR Reg Reg  -- Various swaps
 | JSR | TRAP Int | NOP | HALT    | LABEL String  -- Other instructions
data Reg = PC | SP | MP | R3 | R4 | R5 | R6 | R7
type Code = [Instr]
codeSize :: Code -> Int   -- gegeven functie
```