Automaten en Berekenbaarheid: opgave 2

Prof. B. Demoen (Bart.Demoen@cs.kuleuven.be)
W. Van Onsem (Willem.VanOnsem@cs.kuleuven.be)

12 december 2013

Richtlijnen Je schrijft enkel op de gekleurde bladen die je gekregen hebt: op je werkplek liggen geen andere bladen dan die gekleurde bladen en de bladen van de cursustekst. De bladen van je cursustekst waarop oplossingen van oefeningen of aanwijzingen tot oplossingen van oefeningen staan, steek je ook weg. Je jas, je boekentas, je gsm ... leg je vooraan in het auditorium, zoals gebruikelijk bij een examen. Bij het afgeven zorg je dat je naam op je bladen staat, en dat je alles aan elkaar niet. Er zijn 3 vragen.

Vraag 1 (Herkenbaar, beslisbaar, regulier, context-vrij,...?)**.** *Waar of niet waar? Beargumenteer/bewijs.*

- $EQ_{CFG} = \{\langle G_1, G_2\rangle \,|\, \textit{Twee context-vrije grammatica's } G_1 \textit{ en } G_2 \textit{ met } L(G_1) = L(G_2)\}$ *is co-herkenbaar.*
- *Bepalen of er voor een gegeven Java-programma een invoer bestaat zodat de Java-emulator een gemarkeerde lijn code zal uitvoeren is beslisbaar.*
- $A_{CSG} = \{\langle G, x\rangle \,|\, \textit{Een context-sensitieve grammatica } G \textit{ en een string } s \in \Sigma^* \textit{ met } s \in L(G)\}$ *is herkenbaar.*
- $\{\langle M\rangle \,|\, M$ *is een Turingmachine waarbij de taal* $L(M)$ *een eindig aantal strings bevat*$\}$ *is beslisbaar, herkenbaar, en/of co-herkenbaar.*
- $\{\langle M\rangle \,|\, M$ *is een Turingmachine waarbij de taal* $L(M)$ *minder dan* k *strings bevat*$\}$ *met* k *een vaste parameter is beslisbaar, herkenbaar, en/of co-herkenbaar.*

Antwoord 1.1.

- *Waar. Een PDA kan beslissen (in eindige tijd dus) of een gegeven string door een gegeven grammatica bepaald wordt. Een Turing machine* M *die* EQ_{CFG} *co-herkent werkt dus als volgt: bij de twee gegeven grammatica's wordt de overeenkomstige PDA geconstrueerd; die PDA's worden vervolgens gesimuleerd op een opeenvolging van strings (in lexicografische orde bijvoorbeeld), één string per keer, en op het moment dat een string gevonden wordt waarop de PDA's een verschillend antwoord geven, accepteert* M*.*
- *Niet waar. Je kan de taal* $\langle$*Javaprogramma, gemarkeerde lijn*$\rangle$ Turing-reduceren *naar een taal* $\langle TM$*, set toestanden*$\rangle$ *met de vraag of er een invoer voor de* TM *bestaat die de uitvoering in één van de toestanden brengt die overeen komt met de gemarkeerde lijn. Daarvan hebben we in de oefeningen gezien dat het niet beslisbaar is.*
- *Waar. Dit komt neer op het parsen van een string voor een context sensitieve grammatica. Het acceptance probleem voor context sensitieve grammatica's is beslisbaar als volgt: zet de CSG om in een LBA*[1]*, en voeg (zoals in de cursus) lus-detectie toe aan die LBA. Bepaald worden door een CSG is dus equivalent met* A_{LBA} *hetgeen beslisbaar is.*

[1] Dit deden we niet in de cursus, maar is wel vermeld als eigenschap van CSG

- *Noch herkenbaar noch co-herkenbaar. De eigenschap L_M* bevat een eindig aantal strings *is een niet-triviale, taal-invariante eigenschap van Turing machines, dus volgens Rice is die eigenschap niet beslisbaar. Bijgevolg is de verzameling van TMs met een eindige taal niet herkenbaar of niet co-herkenbaar (misschien beide).*

  Je kan A_{TM} (Turing-)reduceren naar $FINITE_{TM}$ en naar $INFINITE_{TM}$ als volgt:

  1. *$\langle M, w\rangle \to M_{1w}$: op input s laat M $|s|$ stappen lopen op w, indien M ondertussen accept, dan accepteert M_{1w} s anders reject M_{1w} s*
  2. *$\langle M, w\rangle \to M_{2w}$: op input s laat die M $|s|$ stappen lopen op w, indien M ondertussen accept, dan reject M_{2w} s anders accept M_{2w} s*

  M accepteert w als en slechts als M_{1w} heeft oneindige taal M accepteert w als en slechts als M_{2w} heeft eindige taal

  we hebben dus een reductie van een niet herkenbare taal $\overline{A_{TM}}$ naar $FINITE_{TM}$ en $INFINITE_{TM}$ dus $FINITE_{TM}$ is niet herkenbaar, en niet co-herkenbaar.

- *Co-herkenbaar. We gebruiken Rice om te laten zien dat de taal niet beslisbaar is: de taalinvariante, niet-triviale eigenschap is $|L_M| \leq k$*

  Hier is een TM die gegeven een herkent dat $|L_M| > k$: maak van M een enumerator die elke herkende string slechts 1 keer op de outputband zet; vanaf dat er meer dan k strings op de tape staan accept. Indien $|L_M| > k$ stopt dit, anders niet altijd.

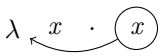
Vraag 2 (Lambda calculus)**.** $(f\ x)$

- *Ontwerp een zuivere λ-expressie hier il genoemd zodat voor elke $i \geq 0$ geldt: $head\left(tail^i\left(il\right)\right) = f^i\left(x\right)$. Mocht men dus de expressie il evalueren zou dit resulteren in volgende oneindige expressie:*

$$CONS(x, CONS(f\left(x\right), CONS(f\left(f\left(x\right)\right), CONS(f\left(f\left(f\left(x\right)\right)\right), \ldots)))) \tag{1}$$

  Evalueer ook de expressie $head\left(tail\left(il\right)\right)$ volgens reductie in normaalorde.

- *In lambda-calculus zijn variabelen niet altijd gebonden. Omcirkel bij onderstaande expressie de gebonden variabelen en wijs met pijl de overeenkomstige lambda aan. Bijvoorbeeld:*



  Ongebonden variabelen duid je aan met een hoedje ($\hat{x}$).

$$+\ y\ \ (\lambda\ x\ .\ +\ y\ \ (\lambda\ y\ .\ (\lambda\ x\ .\ +\ \ (+\ y\ y)\ \ x)\ \ y))(\lambda\ y\ .\ +\ \ y\ x) \tag{2}$$

Antwoord 2.1.

- *We definiëren een helperfunctie il2 die we tijdens de uitwerking vaak zullen gebruiken:*

$$il2 = Y\ \ (\lambda\ g\ \ f\ \ x\ .\ CONS\ x\,(g\ \ f\ \ \ (f\ \ x))) \tag{3}$$

  . De expressie voor il is dan de volgende:

$$il = il2\ f\ x = Y\ \ (\lambda\ g\ \ f\ \ x\ .\ CONS\ x\,(g\ \ f\ \ \ (f\ \ x)))\ \ f\ \ x \tag{4}$$

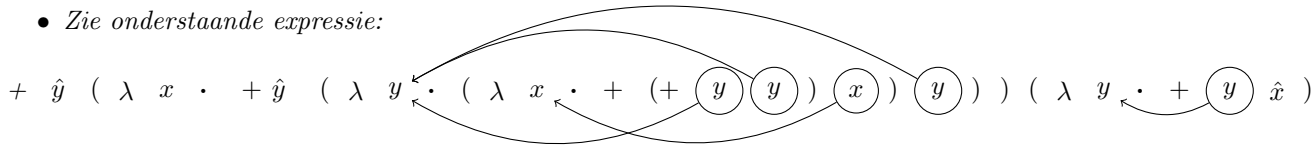
  ofwel voluit

$$(\lambda\ g\ .\ (\lambda\ x\ .\ g\ x\ x)(\lambda\ x\ .\ g\ x\ x))\ \ (\lambda\ g\ \ f\ \ x\ .\ (\lambda\ a\ \ b\ \ f\ .\ f\ \ a\ \ b)\ \ x\,(g\ \ f\ \ \ (f\ \ x)))\ \ f\ \ x \tag{5}$$

De werking kan geïllustreerd worden met behulp van de FAC functie: de Y operator wordt vervangen door $Y\ ily \rightarrow g\ (Y\ ily)$ *met ily de il2 functie zonder Y operator. Dit betekent dus dat g in het gedeelte rechts van Y zal binden met il2. De functie vereist verder twee parameters f en x, als resultaat geven wen een CONS-structuur terug, wat dus een head-tail element is van een lijst. De head wordt bepaald door de parameter x, de tail rekenen we uit door il2 ofwel g verder te propageren. We roepen deze il2 op met dezelfde functie f maar met f x in plaats van x. Doordat deze oproep il2 op zijn beurt weer verder zal propageren, ontstaat de gewenste sequentie.*
We zullen de evaluatie met behulp van de compacte notatie uitvoeren en enkel operatoren ontbinden wanneer dit nodig is:

$$head\ (tail\ (il2\ f\ x)) \tag{6}$$
$$(\lambda\ c\ .\ c\ (\lambda\ a\ b\ .\ a))\ (tail\ (il2\ f\ x)) \tag{7}$$
$$(tail\ (il2\ f\ x))\ (\lambda\ a\ b\ .\ a) \tag{8}$$
$$((\lambda\ c\ .\ c\ (\lambda\ a\ b\ .\ b))\ (il2\ f\ x))\ (\lambda\ a\ b\ .\ a) \tag{9}$$
$$((il2\ f\ x)\,(\lambda\ a\ b\ .\ b))\ (\lambda\ a\ b\ .\ a) \tag{10}$$
$$(((Y\ (\lambda\ g\ f\ x\ .\ CONS\,x\,(g\ f\ (f\ x))))\ f\ x)\,(\lambda\ a\ b\ .\ b))\ (\lambda\ a\ b\ .\ a) \tag{11}$$
$$(((\lambda\ f\ x\ .\ CONS\,x\,(il2\ f\ (f\ x)))\ f\ x)\,(\lambda\ a\ b\ .\ b))\ (\lambda\ a\ b\ .\ a) \tag{12}$$
$$(CONS\,x\,(il2\ f\ (f\ x))\,(\lambda\ a\ b\ .\ b))\ (\lambda\ a\ b\ .\ a) \tag{13}$$
$$((\lambda\ h\ .\ h\ x\,(il2\ f\ (f\ x)))\,(\lambda\ a\ b\ .\ b))\ (\lambda\ a\ b\ .\ a) \tag{14}$$
$$((\lambda\ a\ b\ .\ b)\ x\,(il2\ f\ (f\ x)))\ (\lambda\ a\ b\ .\ a) \tag{15}$$
$$((Y\ (\lambda\ g\ f\ x\ .\ CONS\,x\,(g\ f\ (f\ x))))\ f\ (f\ x))\,(\lambda\ a\ b\ .\ a) \tag{16}$$
$$((\lambda\ f\ x\ .\ CONS\,x\,(il2\ f\ (f\ x)))\ f\ (f\ x))\,(\lambda\ a\ b\ .\ a) \tag{17}$$
$$CONS\ (f\ x)\,(il2\ f\ (f\ (f\ x)))\,(\lambda\ a\ b\ .\ a) \tag{18}$$
$$(\lambda\ h\ .\ h\ (f\ x)\,(il2\ f\ (f\ (f\ x))))\,(\lambda\ a\ b\ .\ a) \tag{19}$$
$$(\lambda\ a\ b\ .\ a)\ (f\ x)\,(il2\ f\ (f\ (f\ x))) \tag{20}$$
$$f\ x \tag{21}$$

- *Zie onderstaande expressie:*



Vraag 3 (Primitief recursief en FOR-programma's)**.** *Primitief recursieve programma's zijn even sterk als* FOR*-programma's: programma's die als enig controle-mechanisme een* `for`*-lus kunnen definiëren. De grenzen (bijvoorbeeld* `m` *en* `n`*) moet voor het binnengaan van de for-lus gekend zijn en zowel de teller als de grens mogen niet aangepast worden tijdens uitvoer van de lus. Recursie is uiteraard ook verboden (anders zou men via recursie for-lussen met aanpasbare teller/grenzen kunnen emuleren). Een geldig* FOR*-programma is bijvoorbeeld:*

```
function f(m,n)
    for i = m:n
        m = m*n;
    endfor
    for i = 0:m
        n = n+i;
    endfor
    return n;
endfunction
```

Een ongeldig FOR*-programma is bijvoorbeeld:*

```
function f(m,n)
    for i = m:n
        i = i*m;
    endfor
    for i = 0:m
        m = m-i;
        i = 2*i;
        n = f(m+1,n-1);
    endfor
    return n;
endfunction
```

Beschrijf aan de hand van een reeks transformatieregels hoe men een gegeven primitief-recursieve functie kan omzetten naar een FOR*-programma.*

Antwoord 3.1. *De basisfuncties zoals nul, suc en* p_i^n *hebben een straight-forward implementatie:*

```
function nul(n)
    return 0;
endfunction

function succ(n)
    return n+1;
endfunction
```

De i*-de projectie uit* n *elementen:*

```
function pin(x1,x2,...,xn)
    return xi;
endfunction
```

De compositie $Cn[f, g_1, g_2, \ldots, g_m]$ *met* $f : \mathbb{N}^m \to \mathbb{N}$ *en voor elke* g_i, $g_i : \mathbb{N}^k \to \mathbb{N}$ *wordt dan als volgt gedefinieerd:*

```
function cn(x1,x2,...,xk)
    y1 = g1(x1,x2,...,xk);
    y2 = g2(x1,x2,...,xk);
    //...
    ym = gm(x1,x2,...,xk);
    return f(y1,y2,...,ym);
endfunction
```

Tot slot de primitieve recursie functie $Pr[f, g]$ *met* $f : \mathbb{N}^k \to \mathbb{N}$ *en* $g : \mathbb{N}^{k+2} \to \mathbb{N}$*:*

```
function pr(x1,x2,...,xk,l)
    y = f(x1,x2,...,xk);
    for i=1:l
        y = g(x1,x2,...,xk,i,y);
    endfor
    return y;
endfunction
```

Indien er in een programma meerdere primitief recursieve en/of compositie-functies voorkomen, moet men natuurlijk verschillende namen definiëren.

Voorbeeld Bij wijze van voorbeeld: de som-functie is gedefinieerd als: $\Pr\left[p_1^1, \mathrm{Cn}\left[\mathrm{succ}, p_3^3\right]\right]$. We bekomen dan volgende code:

```
function succ(n)
    return n+1;
endfunction

function p11(x1)
    return x1;
endfunction

function p33(x1,x2,x3)
    return x3;
endfunction

function cn(x1,x2,x3)
    y1 = p33(x1,x2,x3);
    return succ(y1);
endfunction

function pr(x1,x2)
    y = p11(x1);
    for i = 1:x2
      y = cn(x1,i,y);
    endfor
    return y;
endfunction
```