Informatica Instituut, Faculteit Wiskunde en Informatica, UU.
In elektronische vorm beschikbaar gemaakt door de TBC van A−Eskwadraat.
Het college INFOGONT werd in 2004/2005 gegeven door Jeroen Fokker.
De uitwerking is samengesteld/gemaakt door Roland Vaandrager.

# Uitwerking[1] Grammatica's en Ontleden (INFOGONT) 16 december 2004

N.B. Deze uitwerkingen zijn gebaseerd op de PowerPoint slides van Jeroen Fokker.

## Opgave 1

a) Dit antwoord is nonsens, het complement neem je van talen, niet van grammatica's.

b) Dit antwoord is waar, maak de DFA compleet met een extra sink-state, en complementeer eind-states.

c) Dit antwoord is onwaar, de contextvrijheid is wel gesloten onder vereniging en concatenatie, maar niet onder doorsnede en complement.

## Opgave 2

a) Dit antwoord is onwaar, een andere volgorde van herschrijven geeft een andere derivation, maar dat maakt de grammatica nog niet ambigu.

b) Dit antwoord is waar, het is de definitie van ambiguïteit.

c) Dit antwoord is nonsens, *leftrecursive* zeg je van regels, niet van derivations.

## Opgave 3

Zij $G$ een CF-grammatica $(T, N, R, S)$.

a) $L(G) = \{w \mid T^* \stackrel{*}{\Rightarrow} w, w \in S\}$

Dit antwoord is nonsens. Links van $\stackrel{*}{\Rightarrow}$ moet een string staan, geen sequence; en $S$ is geen verzameling, dus $w \in S$ kan niet.

b) $L(G) = \{w \mid S \stackrel{*}{\Rightarrow} w, w \in T^*\}$

Dit antwoord is waar, dit is de definitie van $L(G)$.

c) $L(G) = \{w \mid S \rightarrow w \in R, w \in T\}$

Dit antwoord is onwaar, in het algemeen geldt dat het herschrijven niet meer dan één stap mag kosten.

## Opgave 4

```
succeed n =
```

a) `const n <$> epsilon`

Dit antwoord is waar, `const n` is de functie die zijn volgende parameter negeert.

b) `n <$> epsilon`

Dit antwoord is nonsens, het geeft een typeringsfout. `n` is een waarde, geen nabewerkingsfunctie.

[1] Deze uitwerkingen zijn met de grootste zorg gemaakt. In geval van fouten kan de TBC niet verantwoordelijk worden gesteld, maar wordt zij wel graag op de hoogte gesteld: `tbc@A-Eskwadraat.nl`

c) `(\x -> n) <$> failp`

Dit antwoord is onwaar, `succeed` levert altijd een lijst met 1 element, `failp` levert altijd de lege lijst.

## Opgave 5

a) Geef het type van:

- `<|> :: Parser a b → Parser a b → Parser a b`
- `<*> :: Parser a (b → c) → Parser a b → Parser a c`
- `<$> :: (b → c) → Parser a b → Parser a c`

b) Kan `f` een functie zijn met:

- 1 parameter:
  `f :: a → c`
  `f <$> p :: Parser Char c`
  Dit kan niet, dit past niet als linkerparameter van `<*>`.
- 2 parameters:
  `f :: a → b → c`
  `r :: Parser Char c`
  Dit kan, dit is het normale gebruik van `<$>`.
- 3 parameters:
  `f :: a → b → c → d`
  `r :: Parser Char (c → d)`
  Dit kan, het is een partiële parametrisatie van `f`.

c) Schrijf `many1 :: Parser a b → Parser a [b]`

Er zijn 3 mogelijkheden:

1. `many1 p = (:) <$> p <*> many p`
   `many p = succeed [] <|> (:) <$> p <*> many p`
2. `many1 p = (:[]) <$> p <|> (:) <$> p <*> many1 p`
3. `many1 p = (:) <$> p <*> (succeed [] <|> many1 p)`

d) Java Boolean *expressies*

```
data Expr = Waar | Onwaar
          | Var String
          | Fun String [Int]
          | :&: Char Expr Expr
          | Not Expr
```

Opmerkingen:

- Bij `:&:` zijn de prioriteiten niet van belang
- Overal moet één constructor komen
- Het geval *expressie tussen haakjes* hoeft niet apart in de abstacte syntax.

e) *Parser* voor Java booleans:

```
expr   = term <|>
         (\x _ y -> Or x y) <$> term <*> token "||" <*> expr

term   = factor <|>
         (\x _ y -> And x y) <$> factor <*> token "&&" <*> term

factor = (\ _ y -> Not y) <$> symbol '!' <*> factor            <|>
         (\ y   -> Var y) <$> ident                            <|>
         Fun              <$> ident <*> parth (listOf integer) <|>
         const Waar       <$> token "False"                    <|>
         const (Con True) <$> token "True"                     <|>
                              parenthesized expr
```

## Opgave 6

a) Maak een reguliere grammatica voor $L(C)$ $L(D)$:

| $L(C)$ | $L(D)$ | $L(C)$ $L(D)$ | |
|---|---|---|---|
| $S \to aP$ | $T \to bT$ | $S \to aP$ | $T \to bT$ |
| $S \to bQ$ | $T \to aW$ | $s \to bQ$ | $T \to aW$ |
| $P \to \varepsilon$ | $T \to W$ | $P \to T$ | $T \to W$ |
| $Q \to a$ | $W \to aX$ | $Q \to aT$ | $W \to aX$ |
| | $W \to bY$ | | $W \to bY$ |
| | $X \to aU$ | | $X \to aU$ |
| | $U \to \varepsilon$ | | $U \to \varepsilon$ |
| | $U \to aU$ | | $U \to aU$ |
| | $Y \to ab$ | | $Y \to ab$ |

b) Maak hiervan een DFA:

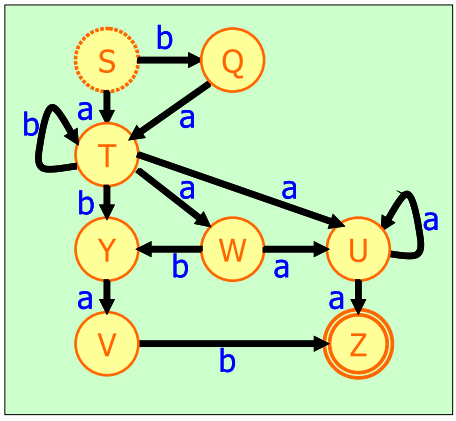
| | |
|---|---|
| $S \to aP$ | $T \to bT$ |
| $S \to bQ$ | $T \to aW$ |
| $P \to T$ | $T \to W$ |
| $Q \to aT$ | $W \to aX$ |
| | $W \to bY$ |
| | $X \to aU$ |
| | $U \to \varepsilon$ |
| | $U \to aU$ |
| | $Y \to ab$ |

$\to$

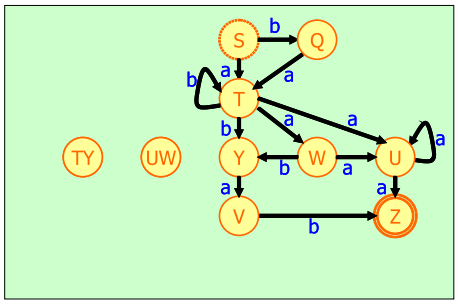
| | | |
|---|---|---|
| $S \to aT$ | $T \to bT$ | |
| $S \to bQ$ | $T \to aW$ | |
| $Q \to aT$ | $W \to aX$ | $T \to aX$ |
| | $W \to bY$ | $T \to bY$ |
| | $X \to aU$ | $X \to aZ$ |
| | $U \to aU$ | $U \to aZ$ |
| | | $Y \to aV$ |
| | | $V \to bZ$ |

Nu is de grammatica "zeer regulier". Rechts is de bijbehorende NFA.

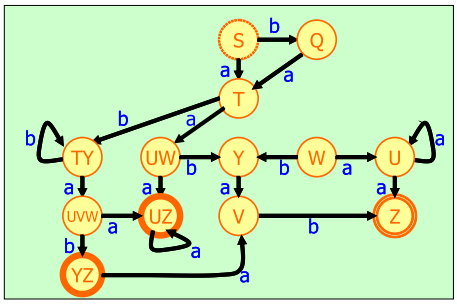
| | | |
|---|---|---|
| $S \to aT$ | $T \to bT$ | |
| $S \to bQ$ | $T \to aW$ | |
| $Q \to aT$ | $W \to aX$ | $T \to aX$ |
| | $W \to bY$ | $T \to bY$ |
| | $X \to aU$ | $X \to aZ$ |
| | $U \to aU$ | $U \to aZ$ |
| | | $Y \to aV$ |
| | | $V \to bZ$ |



c) Maak de automaat deterministisch:




→




→