

# Grammatica's en Ontleden

## Deeltentamen 1 (van 2)

Bastiaan Heeren

Center for Software Technology, Universiteit Utrecht  
<http://www.cs.uu.nl/groups/ST/>

Donderdag 21 december 2006



# 1) Doorsnede van twee talen

Gegeven zijn de talen  $L$  en  $M$  met hetzelfde alfabet. Hieronder staan twee beweringen over deze talen:

**Bewering 1:** “Als  $L$  en  $M$  allebei context-vrij zijn, dan is  $L \cap M$  ook een context-vrije taal.”

**Bewering 2:** “Als  $L$  en  $M$  allebei regulier zijn, dan is  $L \cap M$  ook een reguliere taal.”

- A) Beide beweringen zijn onjuist
- B) Alleen bewering 1 is waar
- C) Alleen bewering 2 is waar
- D) Beide beweringen zijn waar

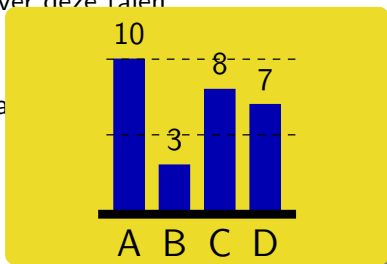


# 1) Doorsnede van twee talen

Gegeven zijn de talen  $L$  en  $M$  met hetzelfde alfabet. Hieronder staan twee beweringen over deze talen:

**Bewering 1:** “Als een context-vrije taal, dan is  $L \cap M$  ook een context-vrije taal.”

**Bewering 2:** “Als een reguliere taal, is  $L \cap M$  ook een reguliere taal.”



dan is  $L \cap M$  ook

is  $L \cap M$  ook een

- A) Beide beweringen zijn onjuist
- B) Alleen bewering 1 is waar
- C) Alleen bewering 2 is waar
- D) Beide beweringen zijn waar



# 1) Doorsnede van twee talen

Gegeven  
twee beweringen

**Bewering 1**  
een context-vrije taal

**Bewering 2**  
reguliere taal

Opgave 2.32 laat zien dat Bewering 1 niet klopt:

- $L_1 = \{a^n b^n c^m \mid n, m \in \mathbb{N}\}$  is context-vrij
- $L_2 = \{a^n b^m c^m \mid n, m \in \mathbb{N}\}$  is context-vrij
- $L_1 \cap L_2$  is dus  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$  en deze taal is niet context-vrij

(Het zal niet lukken om voor deze taal een context-vrije grammatica te geven.)

- A) Beide beweringen zijn onjuist
- B) Alleen bewering 1 is waar
- C) Alleen bewering 2 is waar
- D) Beide beweringen zijn waar



# 1) Doorsnede van twee talen

Gegeven  
twee bew

Opgave 2.32 laat zien dat Bewering 1 niet klopt:

- $L_1 = \{a^n b^n c^m \mid n, m \in \mathbb{N}\}$  is context-vrij
- $L_2 = \{a^n b^m c^m \mid n, m \in \mathbb{N}\}$  is context-vrij
- $L_1 \cap L_2$  is dus  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$  en deze taal is niet context-vrij

**Bewering**  
een conte

**Bewering**  
reguliere

(Het zal niet lukken om voor deze taal een context-vrije grammatica te geven.)

Bewering 2 klopt wel: lees de tekst onder Stelling 10 van H5:

- A)
- B)
- C)
- D)

*"Regular languages are closed under intersection and complement."*

Opgave 5.6 beschrijft hoe je dit kunt bewijzen met behulp van twee NFA's.



# 1) Doorsnede van twee talen

Gegeven zijn de talen  $L$  en  $M$  met hetzelfde alfabet. Hieronder staan twee beweringen over deze talen:

**Bewering 1:** “Als  $L$  en  $M$  allebei context-vrij zijn, dan is  $L \cap M$  ook een context-vrije taal.”

**Bewering 2:** “Als  $L$  en  $M$  allebei regulier zijn, dan is  $L \cap M$  ook een reguliere taal.”

- A) Beide beweringen zijn onjuist
- B) Alleen bewering 1 is waar
- C) Alleen bewering 2 is waar (JUIST)
- D) Beide beweringen zijn waar



## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument) dat gescheiden is door iets anders (tweede argument). Ter herinnering: het type van deze combinator is:

$$\text{listOf} :: \text{Parser } s \ a \rightarrow \text{Parser } s \ b \rightarrow \text{Parser } s \ [a]$$

Welke van de onderstaande definities is een correcte implementatie van *listOf*?

- A)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ s \ \langle * \rangle \ \text{listOf } p \ s \ \langle | \rangle \ \text{succeed } []$
- B)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{many1 } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ p)$
- C)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{option } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ \text{listOf } p \ s) \ []$
- D)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{option } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ p) \ []$



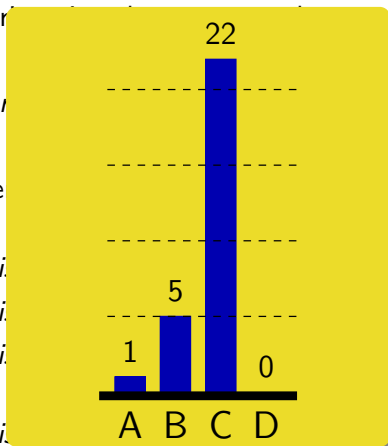
## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument) dat gescheiden is door iets anders (tweede argument). Ter herinnering: de type signature van de parser combinator is:

*listOf* :: Parser a -> Parser b -> Parser [a]

Welke van de onderstaande is de implementatie van *listOf*?

- A) *listOf* p s = li
- B) *listOf* p s = li
- C) *listOf* p s = li
- D) *listOf* p s = li



De implementatie van

> *succeed* []

) <\$> s <\*> p)

)<\$>

) <\$> s <\*> p) []





## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument) dat gescheiden is door iets anders (tweede argument). Ter herinnering: het type van deze combinator is:

*listOf* ::  $F \rightarrow F \rightarrow F$

Eerst laten we de semantische functies buiten beschouwing

Welke van de onderstaande definities is een correcte implementatie van *listOf*?

- A)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ s\ \langle * \rangle\ listOf\ p\ s\ \langle | \rangle\ succeed\ []$
- B)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ many1\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ p)$
- C)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ option\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ listOf\ p\ s)\ []$
- D)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ option\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ p)\ []$



## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument) dat gescheiden is door iets anders (tweede argument). Ter herinnering: het type van deze combinator is:

$$\text{listOf} :: \text{Parser } s \ a \rightarrow \text{Parser } s \ b \rightarrow \text{Parser } s \ [a]$$

Welke van de onderstaande definities is een correcte implementatie van *listOf*?

- A)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ s \ \langle * \rangle \ \text{listOf } p \ s \ \langle | \rangle \ \text{succeed } \square$
- B)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{many1 } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ p)$
- C)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{option } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ \text{listOf } p \ s) \square$
- D)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{option } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ p) \square$



## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument is een string, tweede argument is een parser).

- Definitie A kan  $\epsilon$  afleiden
- Definitie B herkent minstens twee  $p$ 's
- Definitie D kan hooguit twee  $p$ 's afleiden

Welke van de onderstaande definities is een correcte implementatie van *listOf*?

- A)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ s\ \langle * \rangle\ listOf\ p\ s\ \langle | \rangle\ succeed\ []$
- B)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ many1\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ p)$
- C)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ option\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ listOf\ p\ s)\ []$
- D)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ option\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ p)\ []$



## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument is een string, tweede argument is een parser).

- Definitie A kan  $\epsilon$  afleiden
- Definitie B herkent minstens twee  $p$ 's
- Definitie D kan hooguit twee  $p$ 's afleiden

Gelukkig is Definitie C precies wat we zoeken: ook de semantische functies zijn correct!

Welke definitie is de juiste semantische definitie van *listOf*?

- A)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ s\ \langle * \rangle\ listOf\ p\ s\ \langle | \rangle\ succeed\ \square$
- B)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ many1\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ p)\ \square$
- C)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ option\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ listOf\ p\ s)\ \square$
- D)  $listOf\ p\ s = list\ \langle \$ \rangle\ p\ \langle * \rangle\ option\ ((\lambda x\ y \rightarrow y)\ \langle \$ \rangle\ s\ \langle * \rangle\ p)\ \square$



## 2) Parser combinators

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument) dat gescheiden is door iets anders (tweede argument). Ter herinnering: het type van deze combinator is:

$$\text{listOf} :: \text{Parser } s \ a \rightarrow \text{Parser } s \ b \rightarrow \text{Parser } s \ [a]$$

Welke van de onderstaande definities is een correcte implementatie van *listOf*?

- A)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ s \ \langle * \rangle \ \text{listOf } p \ s \ \langle | \rangle \ \text{succeed } \square$
- B)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{many1 } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ p)$
- C)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{option } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ \text{listOf } p \ s) \square$  (JUIST)
- D)  $\text{listOf } p \ s = \text{list } \langle \$ \rangle \ p \ \langle * \rangle \ \text{option } ((\lambda x \ y \rightarrow y) \ \langle \$ \rangle \ s \ \langle * \rangle \ p) \square$



### 3) Grammatica transformatie

Laat  $G = (T, N, R, S)$  een reguliere grammatica zijn voor de reguliere taal  $L$  en  $S' \notin N$ . Welke van de onderstaande grammatica's genereert de taal  $L^*$  en is nog steeds regulier?

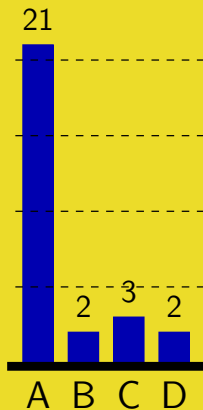
- A)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  alle regels uit  $R$  bevat waarbij achter iedere productieregel die niet eindigt op een hulpsymbool  $S'$  wordt geplakt.
- B)  $(T, N \cup \{S'\}, R \cup \{S' \rightarrow S S', S' \rightarrow \epsilon\}, S')$
- C)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  precies alle regels uit  $R$  bevat plus de productieregels  $X \rightarrow S'$  voor alle hulpsymbolen  $X$  uit de verzameling  $N$ .
- D) De grammatica  $G$  zelf.



### 3) Grammatica transformatie

Laat  $G = (T, N, R, S')$  een reguliere grammatika zijn voor de reguliere taal  $L$  en  $S' \notin N$ .  $V$  is een reguliere uitdrukking voor de taal  $L^*$  en is nog

- A)  $(T, N \cup \{S'\}, R')$  alle regels uit  $R$  behalve die die eindigt op een hulpsymbool
- B)  $(T, N \cup \{S'\}, R \cup \{X \rightarrow S'\})$
- C)  $(T, N \cup \{S'\}, R')$  alle regels uit  $R$  behalve die die eindigt op een hulpsymbool
- D) De grammatica  $(T, N \cup \{S'\}, R \cup \{X \rightarrow S'\})$



n voor de reguliere grammatika's genereert

anig dat  $R'$  alle regels uit  $R$  behalve die die eindigt op een hulpsymbool

anig dat  $R'$  precies alle regels uit  $R$  behalve die die eindigt op een hulpsymbool



### 3) Grammatica transformatie

Opgave 5.1 geeft het juiste antwoord:

- De grammatica van B is niet langer regulier
- Antwoord D is nonsens:  $L(G)$  kan anders zijn dan  $L(G)^*$

iere  
aert

- A)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  alle regels uit  $R$  bevat waarbij achter iedere productieregel die niet eindigt op een hulpsymbool  $S'$  wordt geplakt.
- B)  $(T, N \cup \{S'\}, R \cup \{S' \rightarrow S S', S' \rightarrow \epsilon\}, S')$
- C)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  precies alle regels uit  $R$  bevat plus de productieregels  $X \rightarrow S'$  voor alle hulpsymbolen  $X$  uit de verzameling  $N$ .
- D) De grammatica  $G$  zelf.





### 3) Grammatica transformatie

Opgave 5.1 geeft het juiste antwoord:

- De grammatica van B is niet langer regulier
- Antwoord D is nonsens:  $L(G)$  kan anders zijn dan  $L(G)^*$
- Ook C is fout (hele andere taal)

regels uit  $R$  bevat waarbij achter iedere productieregel die niet eindigt op een hulpsymbool  $S'$  wordt geplakt.

B)  $(T, N \cup \{S'\}, R \cup \{S' \rightarrow S S', S' \rightarrow \epsilon\}, S')$

C)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  precies alle regels uit  $R$  bevat plus de productieregels  $X \rightarrow S'$  voor alle hulpsymbolen  $X$  uit de verzameling  $N$ .

D) De grammatica  $G$  zelf.



### 3) Grammatica transformatie

Laat  $G = (T, N, R, S)$  een reguliere grammatica zijn voor de reguliere taal  $L$  en  $S' \notin N$ . Welke van de onderstaande grammatica's genereert de taal  $L^*$  en is nog steeds regulier?

- A)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  alle regels uit  $R$  bevat waarbij achter iedere productieregel die niet eindigt op een hulpsymbool  $S'$  wordt geplakt. (JUIST)
- B)  $(T, N \cup \{S'\}, R \cup \{S' \rightarrow S S', S' \rightarrow \epsilon\}, S')$
- C)  $(T, N \cup \{S'\}, R' \cup \{S' \rightarrow S, S' \rightarrow \epsilon\}, S')$ , zodanig dat  $R'$  precies alle regels uit  $R$  bevat plus de productieregels  $X \rightarrow S'$  voor alle hulpsymbolen  $X$  uit de verzameling  $N$ .
- D) De grammatica  $G$  zelf.



## 4) Links-recursie verwijderen

Met de volgende grammatica kan een treinreis worden beschreven:

$$TS \rightarrow TS \text{ Tijd Tijd } TS \mid \text{Station}$$

$$\text{Station} \rightarrow \text{Identifier}$$

$$\text{Tijd} \rightarrow \text{Nat} : \text{Nat}$$

Welke van de onderstaande grammatica's is niet meer links-recursief en nog wel equivalent?

A)  $TS \rightarrow TS (Tijd \text{ Tijd } Station)^*$

B)  $TS \rightarrow Station \mid Station Z$

$$Z \rightarrow Tijd \text{ Tijd } TS \mid Tijd \text{ Tijd } TS Z$$

C)  $TS \rightarrow Z \text{ Tijd Tijd } TS \mid Station$

$$Z \rightarrow TS \mid \epsilon$$

D)  $TS \rightarrow Station \text{ Tijd Tijd } Z$

$$Z \rightarrow Station \mid TS$$


## 4) Links-recursie verwijderen

Met de volgende grammatica kan een treinreis worden beschreven:

$$TS \rightarrow TS \text{ Tijd } TS \mid \text{Station}$$

$$\text{Station} \rightarrow \text{Identifier}$$

$$\text{Tijd} \rightarrow$$

Welke van de onderstaande grammaticas is niet links-recursief en nog wel equivalent?

er links-recursief en

A)  $TS \rightarrow TS (T$

B)  $TS \rightarrow \text{Station}$

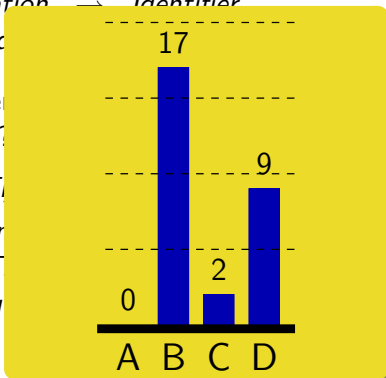
$Z \rightarrow \text{Tijd } T$

C)  $TS \rightarrow Z \text{ Tijd}$

$Z \rightarrow TS \mid \epsilon$

D)  $TS \rightarrow \text{Station } \text{tijd } \text{tijd } Z$

$Z \rightarrow \text{Station} \mid TS$



## 4) Links-recursie verwijderen

Met de volgende grammatica kan een treinreis worden beschreven:

$$TS \rightarrow TS \text{ Tijd Tijd } TS \mid \text{Station}$$

- A is nog steeds links-recursief
- C is indirect links-recursief (en ook verschillend):

$$TS \Rightarrow Z \text{ Tijd Tijd } TS \Rightarrow TS \text{ Tijd Tijd } TS$$

ef en

nog wel equivalent?

- A)  $TS \rightarrow TS (Tijd \text{ Tijd } Station)^*$
- B)  $TS \rightarrow Station \mid Station Z$   
 $Z \rightarrow Tijd \text{ Tijd } TS \mid Tijd \text{ Tijd } TS Z$
- C)  $TS \rightarrow Z \text{ Tijd Tijd } TS \mid Station$   
 $Z \rightarrow TS \mid \epsilon$
- D)  $TS \rightarrow Station \text{ Tijd Tijd } Z$   
 $Z \rightarrow Station \mid TS$



## 4) Links-recursie verwijderen

Met de volgende grammatica kan een treinreis worden beschreven:

$$TS \rightarrow TS \text{ Tijd Tijd } TS \mid \text{Station}$$

- A is nog steeds links-recursief
- C is indirect links-recursief (en ook verschillend):

$$TS \Rightarrow Z \text{ Tijd Tijd } TS \Rightarrow TS \text{ Tijd Tijd } TS$$

- Voor grammatica D geldt dat  $TS \not\Rightarrow \text{Station}$

Het goede antwoord (B) is verkregen na de grammatica transformatie op pagina 24 van het diktaat (“removing left recursion”).

C)  $TS \rightarrow Z \text{ Tijd Tijd } TS \mid \text{Station}$

$$Z \rightarrow TS \mid \epsilon$$

D)  $TS \rightarrow \text{Station Tijd Tijd } Z$

$$Z \rightarrow \text{Station} \mid TS$$

ef en



## 4) Links-recursie verwijderen

Met de volgende grammatica kan een treinreis worden beschreven:

$$TS \rightarrow TS \text{ Tijd Tijd } TS \mid \text{Station}$$

$$\text{Station} \rightarrow \text{Identifier}$$

$$\text{Tijd} \rightarrow \text{Nat} : \text{Nat}$$

Welke van de onderstaande grammatica's is niet meer links-recursief en nog wel equivalent?

A)  $TS \rightarrow TS (Tijd \text{ Tijd } Station)^*$

B)  $TS \rightarrow Station \mid Station Z$

$Z \rightarrow Tijd \text{ Tijd } TS \mid Tijd \text{ Tijd } TS Z$  (JUIST)

C)  $TS \rightarrow Z \text{ Tijd Tijd } TS \mid Station$

$Z \rightarrow TS \mid \epsilon$

D)  $TS \rightarrow Station \text{ Tijd Tijd } Z$

$Z \rightarrow Station \mid TS$



## 5) Lexen en parsen

We gaan een lexer en een parser schrijven voor een taal die bestaat uit proposities met variabelen en universele kwantoren. Eerst worden een aantal voorbeeldzinnen gegeven uit de taal, daarna volgt nog een toelichting.

```

    voor alle x,y : x of niet y
  xs en (waar of onwaar of (voor alle y : niet y))
    p en (q) of niet (p en q)
    waar of niet niet waar
  
```

In proposities mogen variabelen worden gebruikt: zo'n variabele bestaat uit één of meer kleine letters. Sommige woorden (bijvoorbeeld *niet*) hebben een speciale betekenis en mogen niet als variabele worden gebruikt. Iedere geldige propositie mag tussen haakjes geschreven worden. Universele kwantificatie wordt genoteerd door *voor alle* (twee losse woorden), gevolgd door één of meer variabelen gescheiden door komma's, gevolgd door een dubbele punt en een propositie. Tenslotte hebben we nog de binaire operatoren *en* en *of* (deze worden infix geschreven), de unaire operator *niet*, en de twee constanten *waar* en *onwaar*.





## 5) Lexen en parsen (deel twee)

Als eerste gaan we een lexer schrijven. Net als bij de eerste twee praktikumopgaven zijn we niet geïnteresseerd in whitespace: spaties dienen alleen om de tokens van elkaar te scheiden en mogen door de lexer worden weggegooid. Hieronder staat een datatype voor de tokens en de definitie van de tabel *terminals*. Deze definities mag je gebruiken om de lexer te definiëren.

```
data Token = Variabele String | Waar | Onwaar | Voor | Alle
           | Niet | En | Of | Komma | DPunt | Open | Sluit
deriving (Show, Eq)
```

*terminals* =

```
[(Waar, "waar"), (Onwaar, "onwaar"), (Voor, "voor")
, (Alle, "alle"), (Niet, "niet" ), (En, "en" )
, (Of, "of" ), (Komma, "," ), (DPunt, ":" )
, (Open, "(" ), (Sluit, ")" )]
```



## 5a) Lexicale analyse

Schrijf een lexer voor de taal van proposities met behulp van de parser combinators. Het resultaat moet van het type `[Token]` zijn. Geef ook het type van de lexer.



## 5a) Lexicale analyse

Schrijf een lexer voor de taal van proposities met behulp van de parser combinators. Het resultaat moet van het type `[Token]` zijn. Geef ook het type van de lexer.

```

lexProp :: Parser Char [Token]
lexProp = pack whiteSpace (listOf parseToken whiteSpace) whiteSpace

whiteSpace :: Parser Char String
whiteSpace = greedy (satisfy isSpace)

parseToken :: Parser Char Token
parseToken = choice (map f terminals)
              <|> Variabele <$> greedy1 (satisfy isLower)
where f (t, s) = const t <$> token s
  
```



## 5a) Lexicale analyse

### Opmerkingen:

Schrijf een  
combinatie  
het type

- Variabelen komen na de terminals
- Gebruik *determ* in *parseToken* om een keyword gebruikt als variabele te verbieden.

```
lexProp :: Parser Char [Token]
```

```
lexProp = pack whiteSpace (listOf parseToken whiteSpace) whiteSpace
```

```
whiteSpace :: Parser Char String
```

```
whiteSpace = greedy (satisfy isSpace)
```

```
parseToken :: Parser Char Token
```

```
parseToken = choice (map f terminals)
```

```
    <|> Variabele <$> greedy1 (satisfy isLower)
```

```
  where f (t, s) = const t <$> token s
```



## 5) Lexen en parsen (vervolg)

Voordat we een grammatica en een parser kunnen schrijven voor de taal moeten we zeer precies specificeren hoe een zin uit de taal moet worden ontleed. Om een universeel gekwantificeerde propositie als onderdeel van een grotere propositie te gebruiken moeten er haakjes omheen geschreven worden. De unaire operator `niet` heeft de hoogste prioriteit. De zin `niet niet p` is een geldige propositie en betekent `niet (niet p)`. Van de twee binaire operatoren krijgt `en` de hoogste prioriteit. Beide operatoren zijn rechts associatief.



## 5b) Productieregels

Geef productieregels die de taal van proposities beschrijven en die niet ambigu zijn. Het is toegestaan (maar niet verplicht) om EBNF notatie te gebruiken.



## 5b) Productieregels

Geef productieregels die de taal van proposities beschrijven en die niet ambigu zijn. Het is toegestaan (maar niet verplicht) om EBNF notatie te gebruiken.

$$\begin{aligned}
 Prop_1 &\rightarrow Q^? Prop_2 \\
 Prop_2 &\rightarrow Prop_3 \text{ (of } Prop_3)^* \\
 Prop_3 &\rightarrow Prop_4 \text{ (en } Prop_4)^* \\
 Prop_4 &\rightarrow \text{niet } Prop_4 \mid Prop_5 \\
 Prop_5 &\rightarrow V \mid \text{waar} \mid \text{onwaar} \mid ( Prop_1 ) \\
 Q &\rightarrow \text{voor alle } V (, V)^* : \\
 V &\rightarrow ULetter^+
 \end{aligned}$$


## Opmerkingen:

- Let op de verschillende prioriteits-niveaus in de grammatica
- Een alternatieve formulatie (in BNF):  

$$Prop_2 \rightarrow Prop_3 \text{ of } Prop_2 \mid Prop_3$$
- $Prop_4$  en  $Prop_5$  mogen worden samengevoegd
- In  $Prop_5$  zijn de haakjes terminal symbolen

$Prop_1 \rightarrow Q^? P_1 P_2$

$Prop_2 \rightarrow Prop_3 \text{ (of } Prop_3)^*$

$Prop_3 \rightarrow Prop_4 \text{ (en } Prop_4)^*$

$Prop_4 \rightarrow \text{niet } Prop_4 \mid Prop_5$

$Prop_5 \rightarrow V \mid \text{waar} \mid \text{onwaar} \mid ( Prop_1 )$

$Q \rightarrow \text{voor alle } V (, V)^*$  :

$V \rightarrow U\text{Letter}^+$





## 5c) Abstracte syntax

Definieer een Haskell datatype *Prop* om ontleedbomen van proposities te representeren.



## 5c) Abstracte syntax

Definieer een Haskell datatype *Prop* om ontleedbomen van proposities te representeren.

```
data Prop = Const Bool
          | Var String
          | Not Prop
          | Prop :||: Prop
          | Prop :&&: Prop
          | Forall [String] Prop
```



## 5c) Abstracte syntax

Definieer een Haskell datatype *Prop* om ontleedbomen van proposities te representeren.

```
data Prop = Const Bool
          | Var String
          | Not Prop
          | Prop :||: Prop
          | Prop :&&: Prop
          | Forall [String] Prop
```

### Opmerkingen:

- Haakjes horen niet thuis in de abstracte syntax
- De kwantor krijgt een lijst van strings (niet [*Prop*])



## 5d) Ontleider

Schrijf een parser voor proposities: ga er van uit dat de invoer al verwerkt is tot een lijst van tokens. De grammatica van onderdeel **b** is een goede basis voor de parser. Het resultaat van de parser moet het datatype zijn dat je voor onderdeel **c** hebt gekozen. Geef ook het type van de parser.

*Hint: mocht je niet uit de semantische functies komen, schrijf dan wel de parser op waarin je de semantische functies open laat.*



*parseProp* :: Parser Token Prop

*parseProp* = *prop1* **where**

*prop1* = (\$) <\$> option *pQuan id* <\*> *prop2*

*pQuan* = ( $\lambda \_ \_ \_ \_ \_ \rightarrow$  *Forall xs*) <\$> symbol *Voor* <\*> symbol *Alle*  
 <\*> *listOf pVar* (symbol *Komma*) <\*> symbol *DPunt*

*prop2* = *chainr prop3* (*const (:||:)*) <\$> symbol *Of*)

*prop3* = *chainr prop4* (*const (:&&:)*) <\$> symbol *En*)

*prop4* = *const Not* <\$> symbol *Niet* <\*> *prop4* <|> *prop5*

*prop5* = *Var* <\$> *pVar*

<|> *const (Const True)* <\$> symbol *Waar*

<|> *const (Const False)* <\$> symbol *Onwaar*

<|> *pack* (symbol *Open*) *prop1* (symbol *Sluit*)

*pVar* :: Parser Token String

*pVar* = ( $\lambda$ (*Variabele x*)  $\rightarrow$  *x*) <\$> *satisfy isVar*

**where** *isVar* (*Variabele* \_) = *True*

*isVar* \_ = *False*



*parseProp :: Parser Token Prop*

*parseProp = prop1 where*

*prop1 = (\$) <\$> option pQuan id <\*> prop2*

*pQuan = (λ\_ \_ xs \_ → Forall xs) <\$> symbol Voor <\*> symbol Alle  
<\*> listOf pVar (symbol Komma) <\*> symbol DPunt*

*prop2 = chainr prop3 (const (:||:)) <\$> symbol Of*

*prop3 = chainr prop4 (const (:&&:)) <\$> symbol En*

*prop4 = const Not <\$> symbol Niet <\*> prop4 <|> prop5*

*prop5 = Var <\$> pVar*

*<|> const (Const True) <\$> symbol Waar*

*<|> const (Const False) <\$> symbol Onwaar*

*<|> pack (symbol Open) prop1 (symbol Sluit)*

*pVar :: Parser Token String*

*pVar = (λ(Variabele x) → x) <\$> satisfy isVar*

**where** *isVar (Variabele \_) = True*

*isVar \_ = False*



## Opmerking:

- Deze parser is een directe vertaling van de eerder getoonde grammatica.

*parseProp* :: Parser Token

*parseProp* = *prop1* **where**

*prop1* = (\$) <\$> c

*pQuan* = ( $\lambda\_ \_ xs \_ \rightarrow$  *forall xs*) <\$> *symbol Voor* <\*> *symbol Alle*  
 <\*> *listOf pVar (symbol Komma)* <\*> *symbol DPunt*

*prop2* = *chainr prop3 (const (:||:))* <\$> *symbol Of*

*prop3* = *chainr prop4 (const (:&&:))* <\$> *symbol En*

*prop4* = *const Not* <\$> *symbol Niet* <\*> *prop4* <|> *prop5*

*prop5* = *Var* <\$> *pVar*

<|> *const (Const True)* <\$> *symbol Waar*

<|> *const (Const False)* <\$> *symbol Onwaar*

<|> *pack (symbol Open) prop1 (symbol Sluit)*

*pVar* :: Parser Token String

*pVar* = ( $\lambda(\text{Variabele } x) \rightarrow x$ ) <\$> *satisfy isVar*

**where** *isVar* (*Variabele* \_) = *True*

*isVar* \_ = *False*



## 5e) Lexen en parsen

Schrijf een functie

$$test :: String \rightarrow Prop$$

die een string ontleedt en een propositie teruggeeft. Je mag er van uit gaan dat de string altijd een geldige propositie voorstelt. De functie *start* om een parser mee op te starten mag *niet* bekend worden verondersteld. Als je deze functie wilt gebruiken dan moet je deze eerst nog definiëren.





## 5e) Lexen en parsen

Schrijf een functie

$$test :: String \rightarrow Prop$$

die een string ontleedt en een propositie teruggeeft. Je mag er van uit gaan dat de string altijd een geldige propositie voorstelt. De functie *start* om een parser mee op te starten mag *niet* bekend worden verondersteld. Als je deze functie wilt gebruiken dan moet je deze eerst nog definiëren.

$$test :: String \rightarrow Prop$$

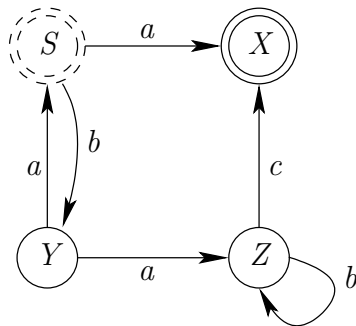
$$test = start \text{ parseProp} . start \text{ lexProp}$$

$$start :: Parser \ s \ a \rightarrow [s] \rightarrow a$$

$$start \ p = fst . head . filter (null . snd) . p$$

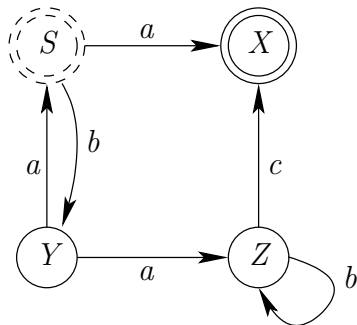

## 6) Reguliere talen

Gegeven is de volgende niet-deterministische eindige toestandsautomaat (NFA), waarin  $S$  de enige start-toestand is en  $\{S, X\}$  de verzameling is van eind-toestanden.



## 6a) Wel in de taal

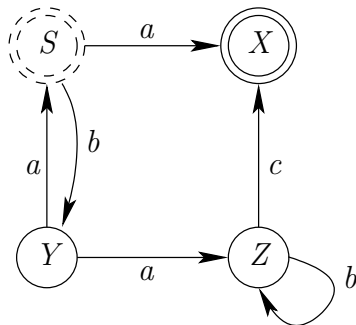
Geef drie verschillende strings uit  $\{a, b, c\}^*$  die behoren tot de taal van de NFA.



## 6a) Wel in de taal

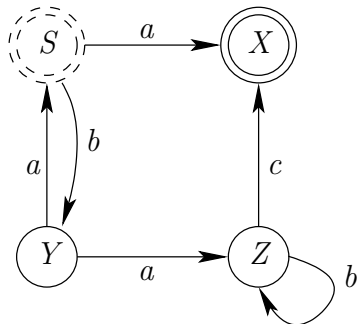
Geef drie verschillende strings uit  $\{a, b, c\}^*$  die behoren tot de taal van de NFA.

- $\epsilon$
- $a$
- $ba$
- $baa$
- $bac$
- ... etcetera ...



## 6b) Niet in de taal

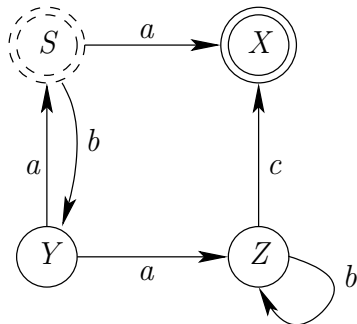
Geef drie verschillende strings uit  $\{a, b, c\}^*$  die *niet* behoren tot de taal van de NFA.



## 6b) Niet in de taal

Geef drie verschillende strings uit  $\{a, b, c\}^*$  die *niet* behoren tot de taal van de NFA.

- $c$
- $ca$
- $cb$
- $cc$
- $aa$
- $bb$
- ... etcetera ...

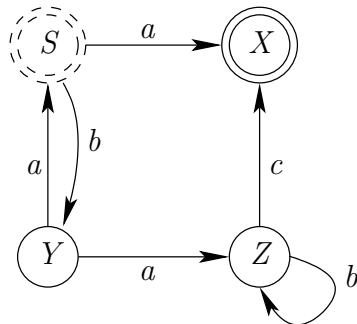


## 6b) Niet in de taal

Geef drie verschillende strings uit  $\{a, b, c\}^*$  die *niet* behoren tot de taal van de NFA.

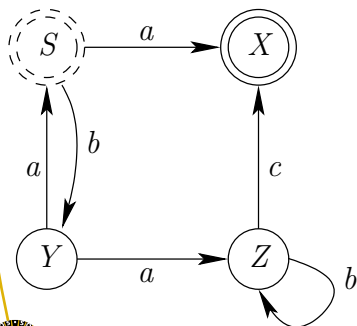
- $c$
- $ca$
- $cb$
- $cc$
- $aa$
- $bb$

Uitdaging: construeer eerst een automaat voor het complement van de NFA.



## 6c) Deterministische automaat

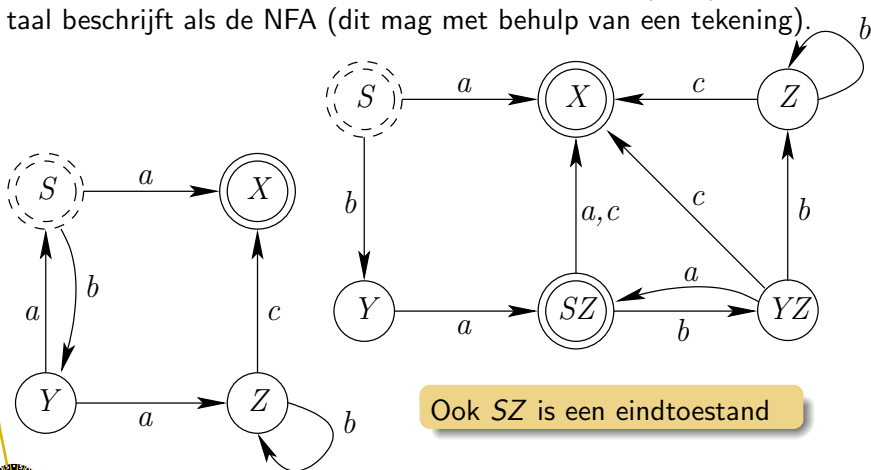
Construeer een deterministische toestandsautomaat (DFA) die dezelfde taal beschrijft als de NFA (dit mag met behulp van een tekening).





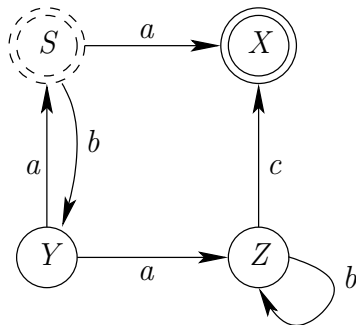
## 6c) Deterministische automaat

Construeer een deterministische toestandsautomaat (DFA) die dezelfde taal beschrijft als de NFA (dit mag met behulp van een tekening).



## 6d) Reguliere expressie

Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.



## 6d) Reguliere expressie

Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.

**Oplossing 1:** goed kijken

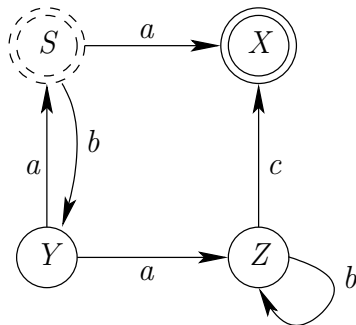
Eindigen in  $S$ :

- $(ba)^*$

Eindigen in  $X$ :

- $(ba)^* a$
- $(ba)^* bab^* c$

Oftewel:  $(ba)^* (\epsilon + a + bab^* c)$

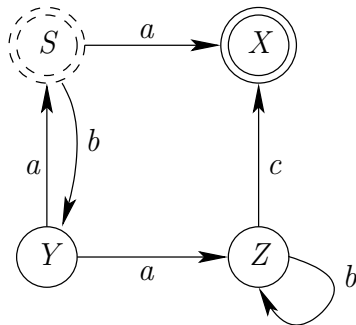


## 6d) Reguliere expressie

Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.

**Oplossing 2:** vergelijkingen opstellen

$$\begin{cases} S &= aX + bY + \epsilon \\ X &= \epsilon \\ Y &= aS + aZ \\ Z &= bZ + cX \end{cases}$$



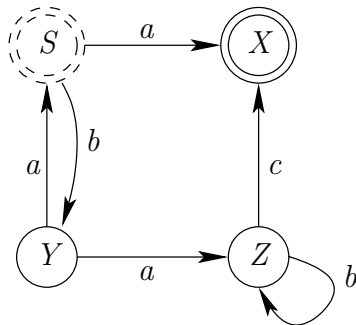
## 6d) Reguliere expressie

Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.

**Oplossing 2:** vergelijkingen opstellen

$$\begin{cases} S &= aX + bY + \epsilon \\ X &= \epsilon \\ Y &= aS + aZ \\ Z &= bZ + cX \end{cases}$$

$$\begin{cases} S &= a + b(aS + aZ) + \epsilon \\ Z &= bZ + c \end{cases}$$



## 6d) Reguliere expressie

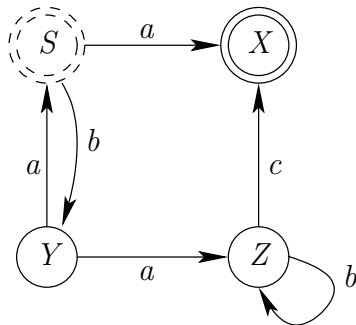
Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.

**Oplossing 2:** vergelijkingen opstellen

$$\begin{cases} S &= aX + bY + \epsilon \\ X &= \epsilon \\ Y &= aS + aZ \\ Z &= bZ + cX \end{cases}$$

$$\begin{cases} S &= a + b(aS + aZ) + \epsilon \\ Z &= bZ + c \end{cases}$$

$$\begin{cases} S &= (ba)^* (\epsilon + a + baZ) \\ Z &= b^* c \end{cases}$$



## 6d) Reguliere expressie

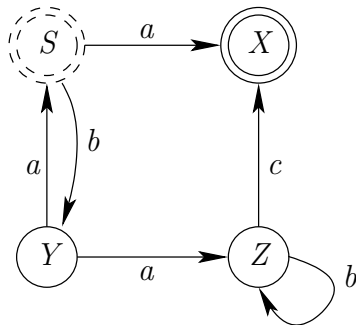
Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.

**Oplossing 2:** vergelijkingen opstellen

$$\begin{cases} S &= aX + bY + \epsilon \\ X &= \epsilon \\ Y &= aS + aZ \\ Z &= bZ + cX \end{cases}$$

$$\begin{cases} S &= a + b(aS + aZ) + \epsilon \\ Z &= bZ + c \end{cases}$$

$$\begin{cases} S &= (ba)^* (\epsilon + a + baZ) \\ Z &= b^* c \end{cases}$$



Oplossing:

$$(ba)^* (\epsilon + a + bab^* c)$$

