Tentamen vraag 1

- Elke reguliere zin is ook contextvrij

nonsens
"regulier" en "contextvrij" zegt men niet van "zin"

- Elke reguliere taal is ook contextvrij

waar

- Elke contextvrije grammatica is ook regulier

onwaar
Een reguliere grammatica moet aan strengere regels voldoen

Tentamen vraag 2

Een links-recursieve grammatica is vaak een probleem, want

- Zo'n grammatica is altijd ambigu
- Zo'n grammatica termineert niet
- Het is lastig om een parser voor de taal ervan te schrijven

onwaar
S ->Sa | a
is linksrecursief,
maar niet ambigu

nonsens
Een grammatica is geen programma, dus "termineren" is geen issue

Waar
de standaard-aanpak levert een niet-terminerende parser, dus je moet de grammatica eerst transformeren

Tentamen vraag 3

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

W is een zin van L(G) als er een string u is met

- $S \rightarrow u$ en $u \Rightarrow w$

nonsens
dit is een *herschrijfregel*,
niet iets dat waar kan zijn

- $S \rightarrow w \in R$ en $u \Rightarrow w$

onwaar, i.h.a.
het herschrijven mag
meer dan één stap kosten

- $S \Rightarrow u$ en $u \Rightarrow^* w$

waar
dit is zowat
de definitie van L(G)

Tentamen vraag 4

Als G_1 en G_2 CF-grammatica's zijn, dan is

- $L(G_1) \cup L(G_2)$ altijd CF

waar
eenvoudige constructie:
$R = R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$

- $G_1 \cup G_2$ soms CF

nonsens
een grammatica is een 4tupel, niet een verzameling

- $L(G_1) \cap L(G_2)$ nooit CF

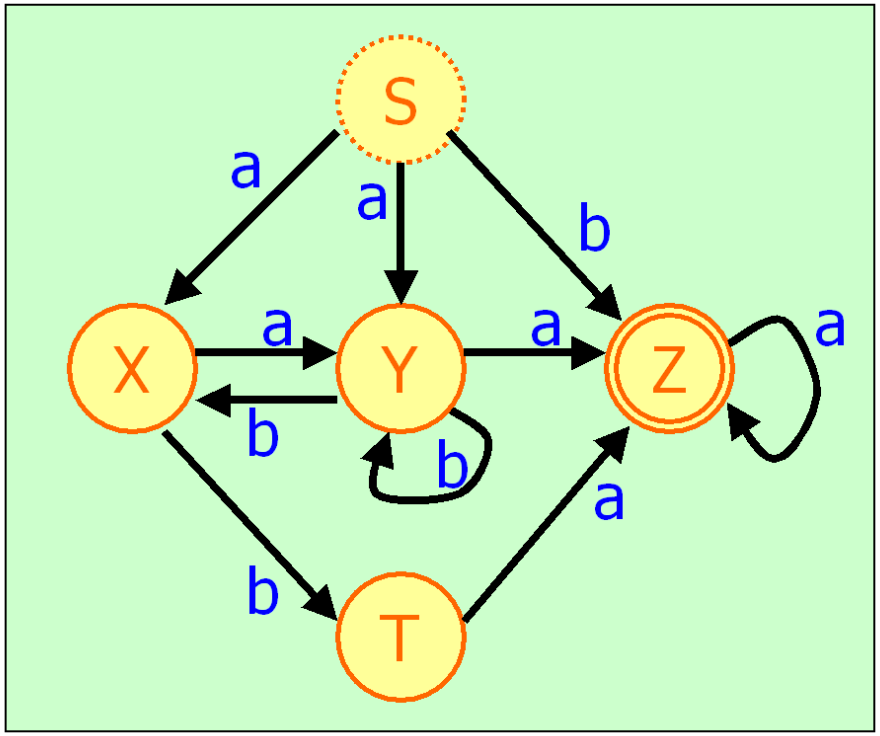
onwaar
de doorsnede is niet altijd CF, maar soms wel: bijv. als $G_1 = G_2$

Tentamen vraag 6a

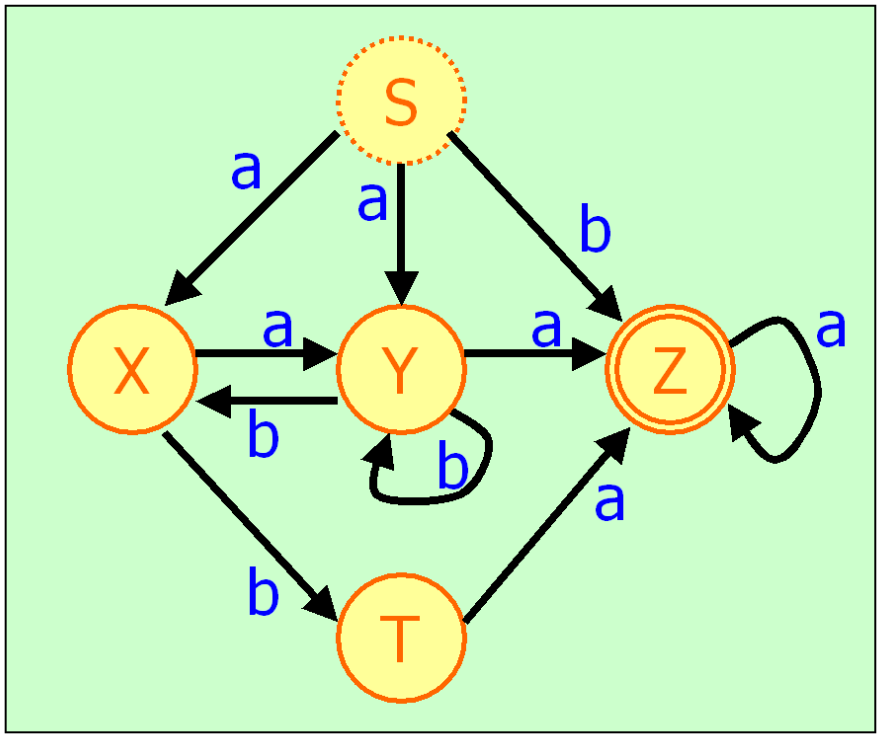
G = ({a,b}
, {S,T,X,Y,Z}
,{

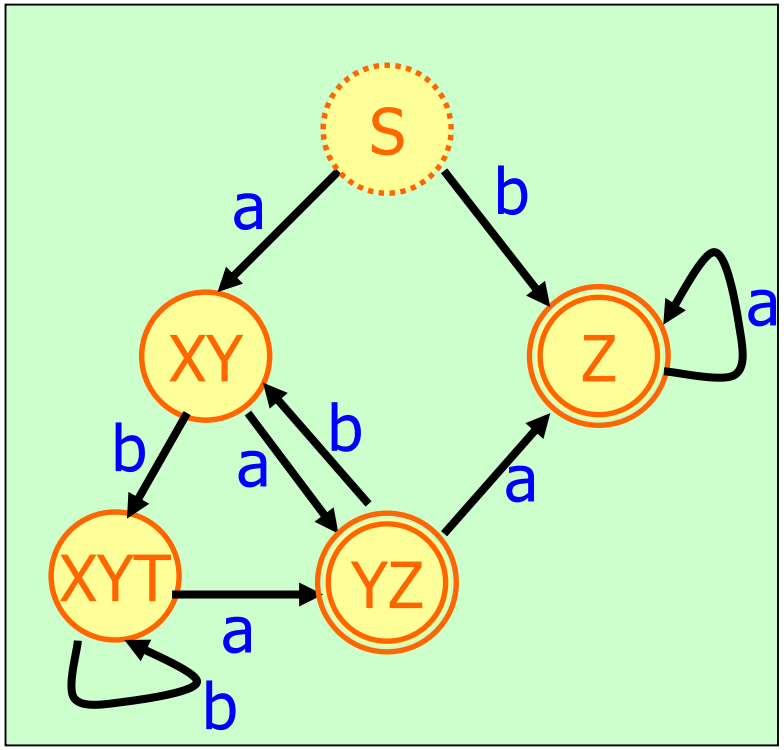
$S \rightarrow a\ X$	$Y \rightarrow a\ Z$
$S \rightarrow a\ Y$	$Y \rightarrow b\ X$
$S \rightarrow b\ Z$	$Y \rightarrow b\ Y$
$X \rightarrow a\ Y$	$T \rightarrow a\ Z$
$X \rightarrow b\ T$	$Z \rightarrow a\ Z$
	$Z \rightarrow \varepsilon$

}, S }



Tentamen vraag 6b





Tentamen vraag 6c

$R = R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$

- Beschrijft de taal van concatenaties van zinnen uit resp. G_1 en G_2
  $L(G) = L(G_1)\, L(G_2)$

- Niet regulier want $S \rightarrow S_1 S_2$ voldoet niet aan de regulariteits-eis

- $R = R_1 \cup R_2'$
  R_2' is R_2 met bij de regels die *niet* op een terminal eindigen een S2 aan het eind
  $S = S_1$

Tentamen vraag 5a

- Definieer

```
stdTypes = [“int”, “long”, “double”, … ]

lexStdType :: Parser Char Token

lexStdType =  StdType ®
              choice  (map token stdTypes)
```

Tentamen vraag 5b

- Definieer

```
type Klasse = (String, [Elem])
data Elem   =  D Decl
            |  M  Meth
            |  S  Stat
type Meth   =  (Type, String, [Decl], Stat)
type Decl   =  (Type, String)
data Type   =  …
data Stat   =  …
```

Tentamen vraag 5b vervolg

```
data Type   = Void
            | Prim      String
            | Obj       String
            | Gen       Type Type
            | Array     Type
data Stat   =  Exec      Expr
            | If        Expr Stat Stat
            | While     Expr Stat
            | Return    Expr
            | Try       Stat    [(Decl,Stat)]
            | Block     [Elem]
```

Tentamen vraag 5c (Decl)

```
pDecl :: Parser Token Decl
pDecl  =   (\x y → (x,y))   ®
           pType ⊗ pLowerId

pDeclPK = (\x _ → x)  ®
          pDecl ⊗ symbol Puntkomma
```

Tentamen vraag 5c (Klasse)

```
pKlasse :: Parser Token Klasse
pKlasse  = (\_ x y → (x,y))   ®
             symbol KeyClass
           ⊗ pUpperId
           ⊗ braced (many (  D ®  pDeclPK
                           ⊕ M ® pMeth
                          )      )

braced p = pack (symbol AcOpen) p (symbol AcSluit)
```

Tentamen vraag 5c (Meth)

```
pMeth :: Parser Token Meth
pMeth  =  (\x y z u→ (x,y,z,u))   ®
             ( pType ⊕  pVoid )
          ⊗ pLowerId
          ⊗ parenth (option (listOf pDecl
                                  symbol Komma)
                            [ ] )
          ⊗ pBlock

 pVoid   =   const Void ® symbol KeyVoid
```

Tentamen vraag 5c (Stat)

```
pStat :: Parser Token Stat
pStat  =    If    ® symbol KeyIf  ⊗ ...  ⊗ ...
          ⊕ While  ® symbol KeyWhile ⊗ ...
          ⊕ ...
```