

# Compilers - Voorbeeldexamen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 4 juni 2019

# Inhoudsopgave

<b>1</b>	<b>Vragen</b>	<b>2</b>
1.1	Compilerfasen - I . . . . .	2
1.2	Lexicale analyse . . . . .	3
1.3	Compilerfasen - II . . . . .	5
1.4	NFA . . . . .	7

# Hoofdstuk 1

## Vragen

### 1.1 Compilerfasen - I

Gegeven de opeenvolgende fasen van een compiler, geef aan de programmaproducten, grafen of datastructuren die aan het *eind* van elke fase geproduceerd worden. Geef uw antwoord in de vorm: a-7, b-3, ...

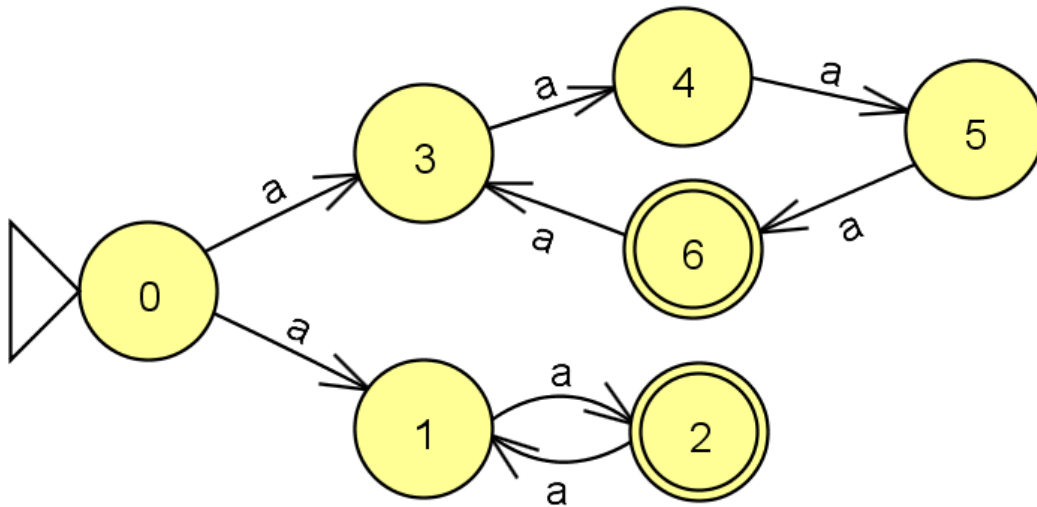
Compilerfase	Programmaproduct
(a) scanning (lexicale analyse)	1. intermediaire boomtaal
(b) parsing (syntactische analyse)	2. control flow graaf (CFG)
(c) type checking (semantische analyse)	3. frame layout, activation records
(d) vertaling (translate)	4. symbooltabellen
(e) canonicalisering	5. interferentiegraaf
(f) instructieselectie	6. gekleurde interference graph
(g) controlestroom analyse	7. tokens
(h) dataflow analyse (liveness)	8. assembler code
(i) register allocatie	9. assembler instructies
(j) code emission	10. abstracte syntax tree (AST)

### Antwoord

a-7, b-10, c-?, d-?, e-?, f-9, g-?, h-6, i-?, j-8

## 1.2 Lexicale analyse

Volgende nondeterministische eindige automaat herkent geldige tokens  $(aa)^+$  en  $(aaa)^+$ .



1. Beschrijf de manier om in het algemeen verschillende NFA's samen te voegen tot 1 DFA. Bespreek daarbij de definitie van *sluiting*, *DFAedge* en het *algoritme* dat de NFA-toestanden samenvoegt tot toestanden in een DFA.
2. Pas dit toe op dit voorbeeld: de herkenning van twee- en drievouden van  $a$  en teken de bekomen DFA.
3. Zijn er in de gevonden DFA nog equivalente toestanden die men kan vereenvoudigen?

### Antwoord

1. Een deterministische eindige automaat (DFA) is een eindige automaat waarbij elke transitie uniek is. Het algoritme om een NFA om te vormen naar een DFA maakt enerzijds gebruik van sluitingen. De sluiting  $T$  van een verzameling van toestanden  $S$ , of  $\text{closure}(S)$ , bevat alle toestanden die kunnen bereikt worden voor de lege transitie  $\epsilon$  voor elke staat in  $S$ . Dit kan wiskundig gedefinieerd worden als:

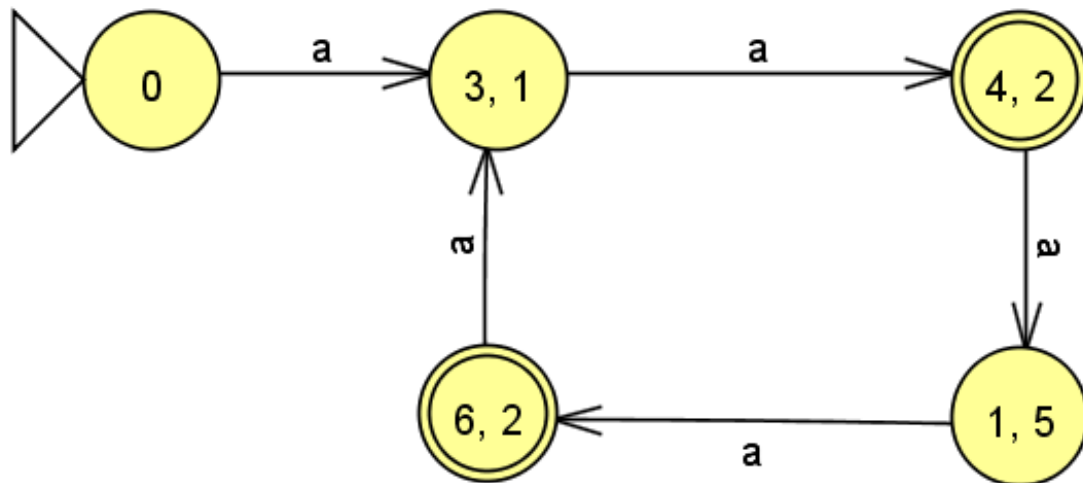
$$T = S \cup \left( \bigcup_{s \in T} \text{edge}(s, \epsilon) \right)$$

waarbij  $\text{edge}(s, c)$  de staten geeft die vanuit  $s$  via symbool  $c$  kan bereikt worden.

Anderzijds maakt het algoritme ook gebruik van de functie  $\text{DFAedge}(d, c)$ , die de staten teruggeeft die vanuit  $d$  kunnen bereikt worden bij symbool  $c$ .

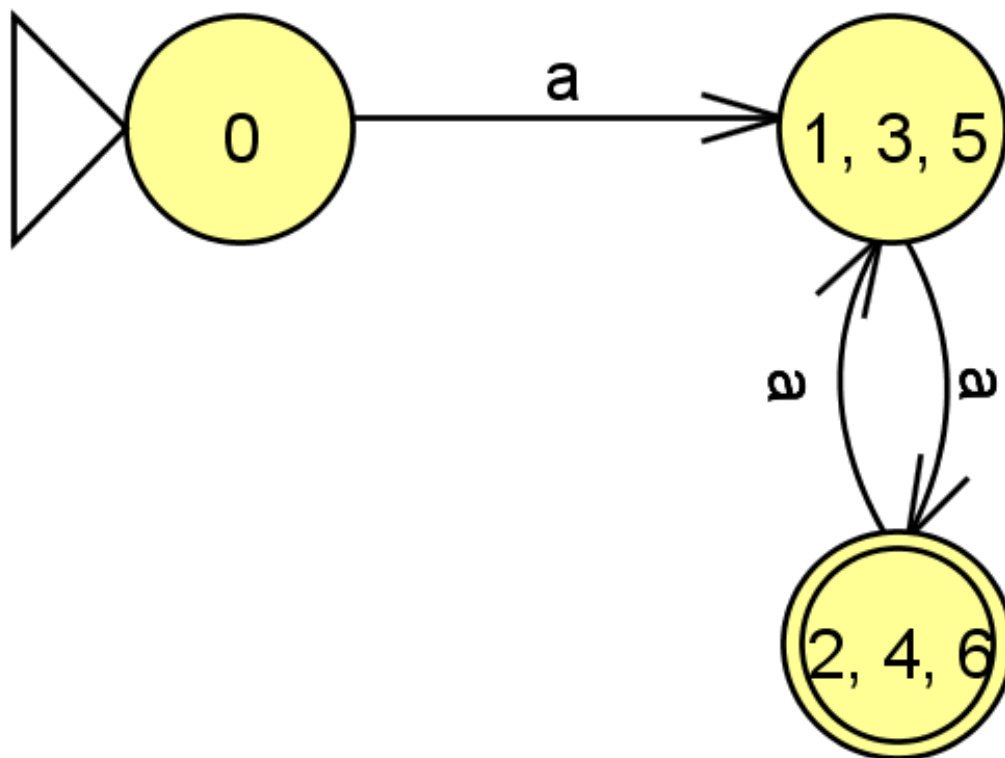
*ToDo: verder uitleggen*

2. De automaat is enkel in staat om enkelvoudige  $a$  symbolen te verwerken en aangezien er geen lege strings zijn, draagt de  $\text{closure}(S)$  functie hier niets bij. De eerste verwerking zorgt ervoor dat staten 1 en 3 bereikt worden. Vanuit 1 kan er naar 2 gegaan worden en vanuit 3 kan er naar 4 gegaan worden. In dit geval moet dit herhaald worden tot dat er een combinatie van staten gevonden is die al eerder gecombineerd zijn. Dit is het geval als uiteindelijk de staten



6 en 2 bereikt worden. Vanuit 6 kan naar 3 gegaan worden en vanuit 2 kan naar 1 gegaan worden. Die combinatie bestaat al, en de DFA is voltooid.

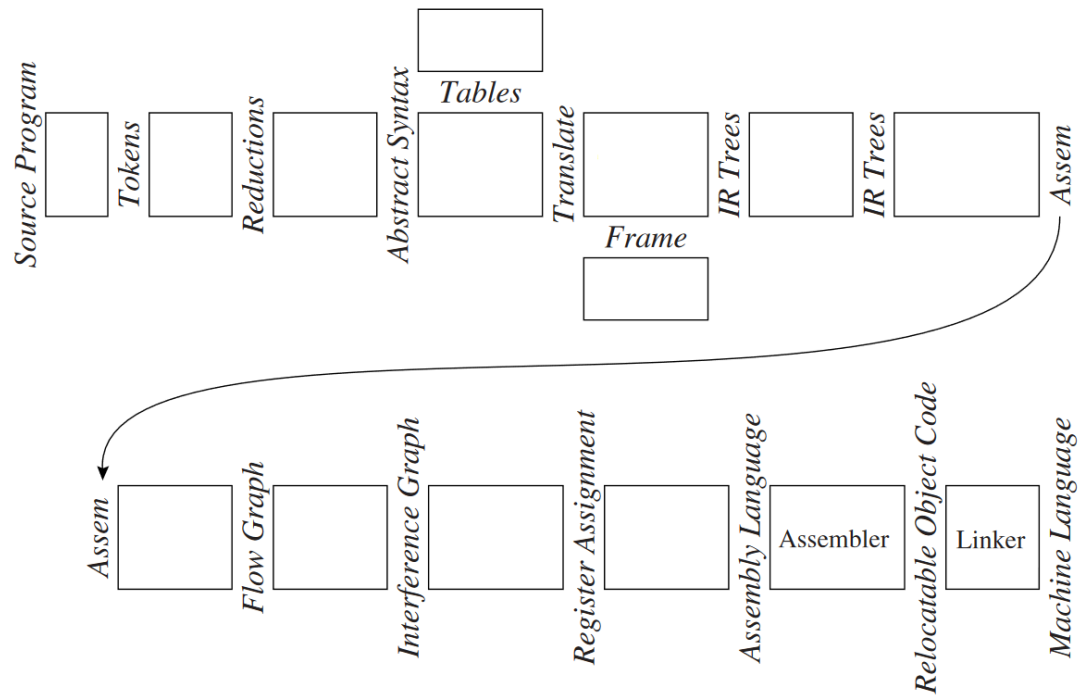
3. Het algoritme garandeert niet dat de opgeleverde DFA optimaal is, maar er kunnen nadien wel nog optimalisaties doorgevoerd worden. Staten die equivalent zijn kunnen samengenomen worden. Een staat  $s_1$  is equivalent met staat  $s_2$  als ze beiden finaal of niet finaal zijn voor dezelfde symbolen en als voor elk symbool  $c$ ,  $\text{trans}[s_1, c] = \text{trans}[s_2, c]$ . In dit geval is dit waar voor staat  $\{3, 1\}$  met  $\{1, 5\}$  en voor staat  $\{4, 2\}$  met  $\{6, 2\}$ . Deze kunnen gecombineerd worden.



### 1.3 Compilerfasen - II

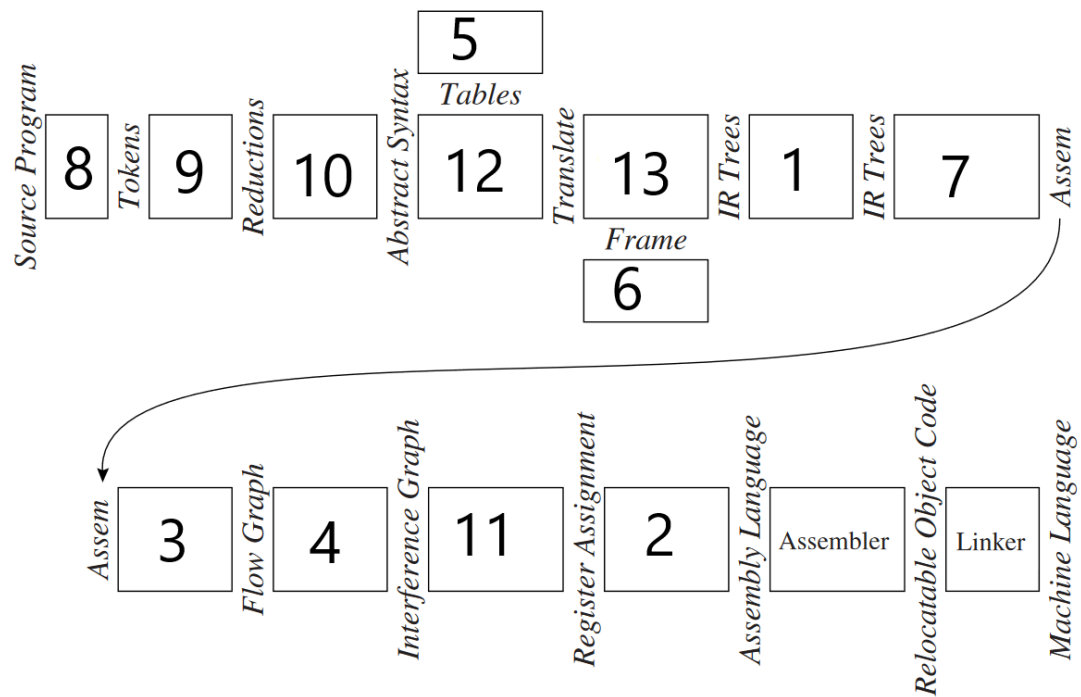
De volgende figuur toont de intermediaire voorstellingen die gebruikt worden als interface tussen de verschillende fasen van een compiler. Vul de nummers in van de corresponderende fasen in een compiler.

1. Canonicalize
2. Code Emission
3. Control Flow Analysis
4. Data Flow Analysis
5. Environments
6. Frame Layout
7. Instruction Selection
8. Lex
9. Parse
10. Parsing Actions

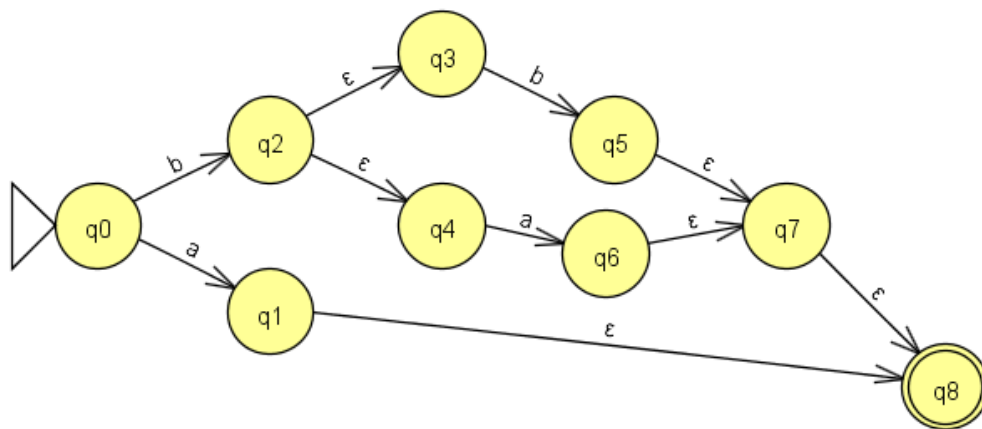


11. Register Allocation
12. Semantic Analysis
13. Translate

**Antwoord**



## 1.4 NFA

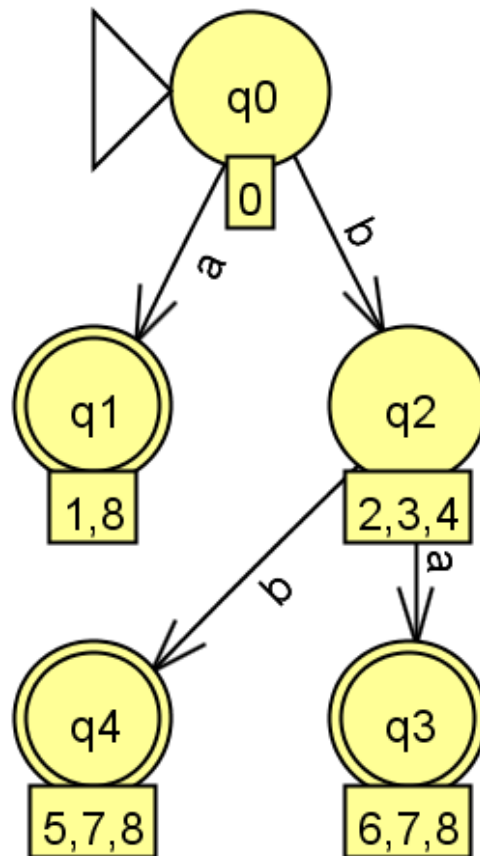


1. Converteer deze NFA naar een DFA.
2. Welke reguliere expressie wordt hiermee voorgesteld?

### Antwoord

1. Gebruik dezelfde methodologie als in vraag 1.2.





2. De reguliere expressie die voorgesteld wordt is

$a+ba+bb$

ToDo: hoe bekomen