Eindige Automaten

12

zie dictaat

ch.12 Languages, Automata, Grammars
12.5 finite state automata

toestands-actie-diagrammen

Eindige automaten zijn zgn. toestands-actie- diagrammen. Vanuit een toestand raak je via een actie in een andere toestand. “eindig” omdat er slechts eindig veel mogelijke toestanden en acties zijn.
De diagrammen zijn handig om systemen te analyseren. We formaliseren het “gedrag” van een systeem als de bijbehorende taal van actie-reeksen.

Eerst voorbeelden.

voorbeelden

- de state-transition diagrams van **DiTe** zijn technisch anders maar hebben een gelijke onderliggende filosofie.
- een plaatje en procedure uit een **netwerken** boek, om te kunnen analyseren of een algoritme dat tussen twee computers nooit stopt, bijvoorbeeld doordat ze op elkaar staan te wachten.
- bij **Algoritmiek** kun je problemen modelleren door de mogelijke toestanden te inventariseren.
- het **testen** van een stukje software door uit de beschrijving de toestanden te abstraheren, en vervolgens vreemd gedrag te constateren.
- **syntax-diagrammen** voor stukjes programmeertaal hebben ook toestanden en overgangen.

Example: Moore Machine (cont.)

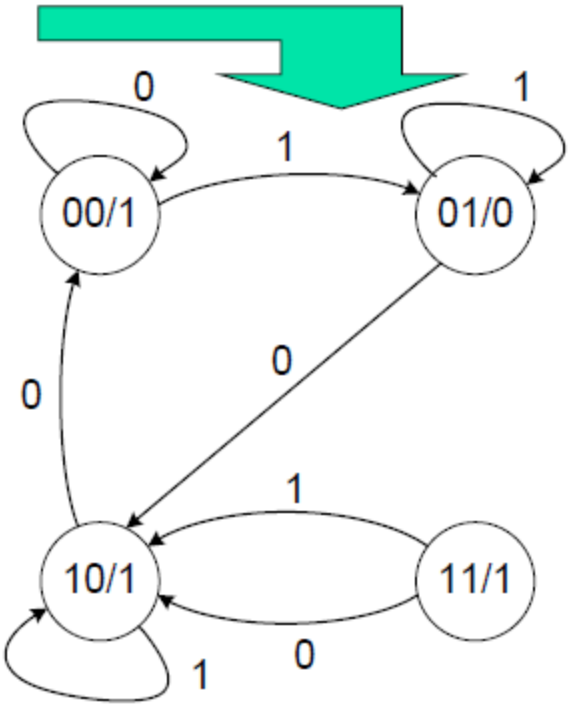
Inputs	Present State		Next State		Outputs
x(t)	$q_1(t)$	$q_2(t)$	$q_1(t+1)$	$q_2(t+1)$	z(t)
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	1	1

state / transition
toestand / actie

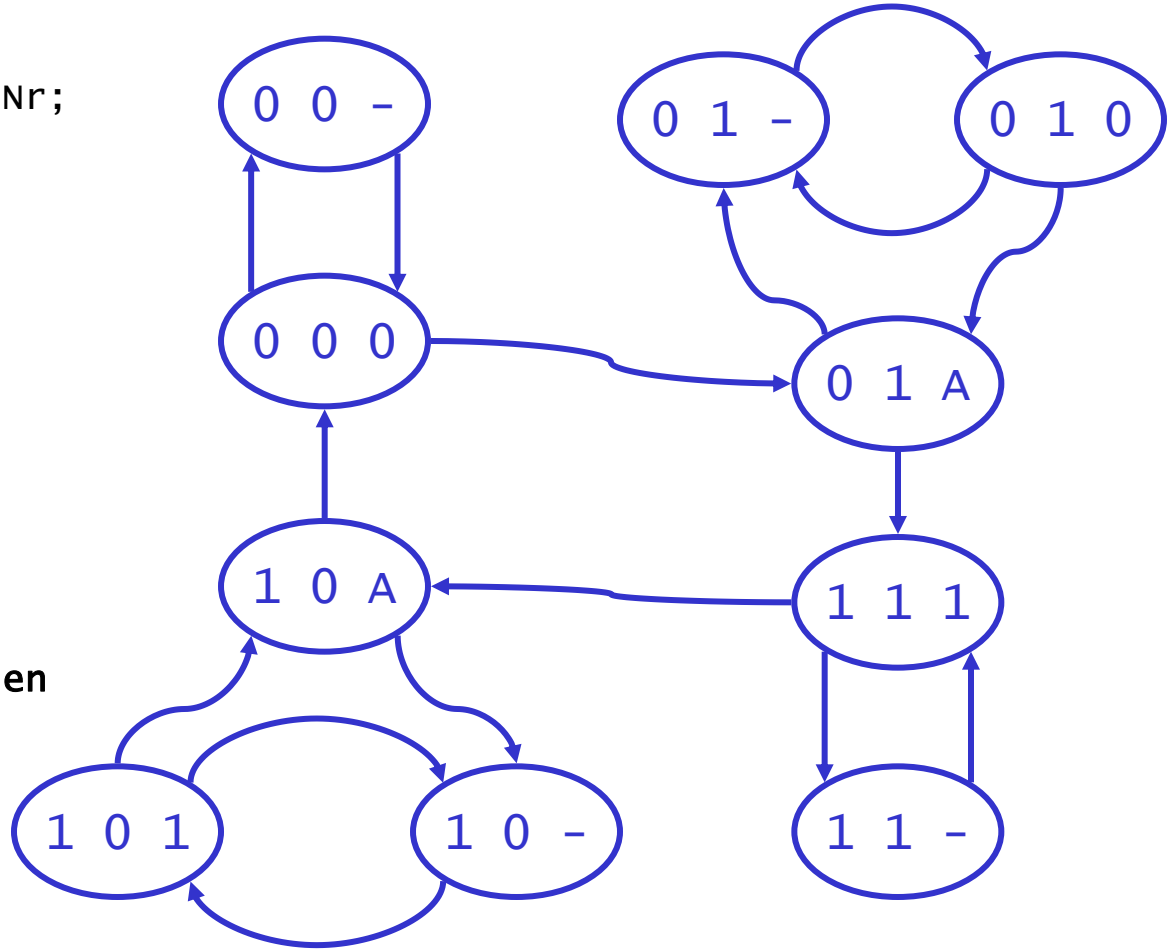
maar:
- 'interne bits'
- uitvoer
- (non)determinisme
- begin- en eindtoestanden
- taal

Reads as:
When at state ***Q1*** with output ***Z1*** and apply input ***X***, we proceed to state ***Q2*** with output ***Z2.***

- Possible states = { 00, 01, 10, 11 }
→ 4 nodes in the diagram
- Possible Transitions = #rows in table
→ 8 edges in the diagram

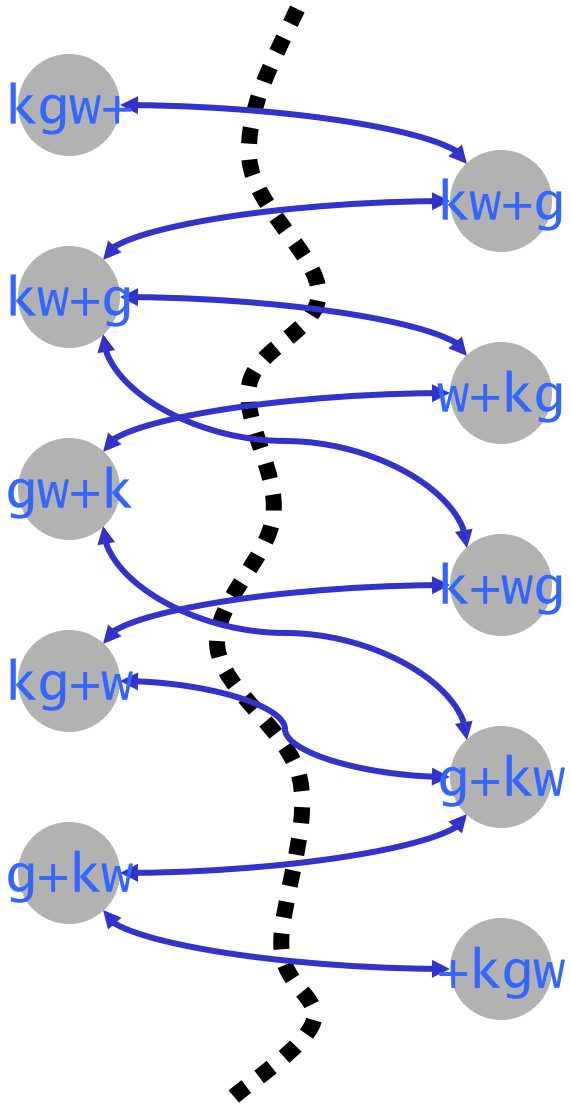


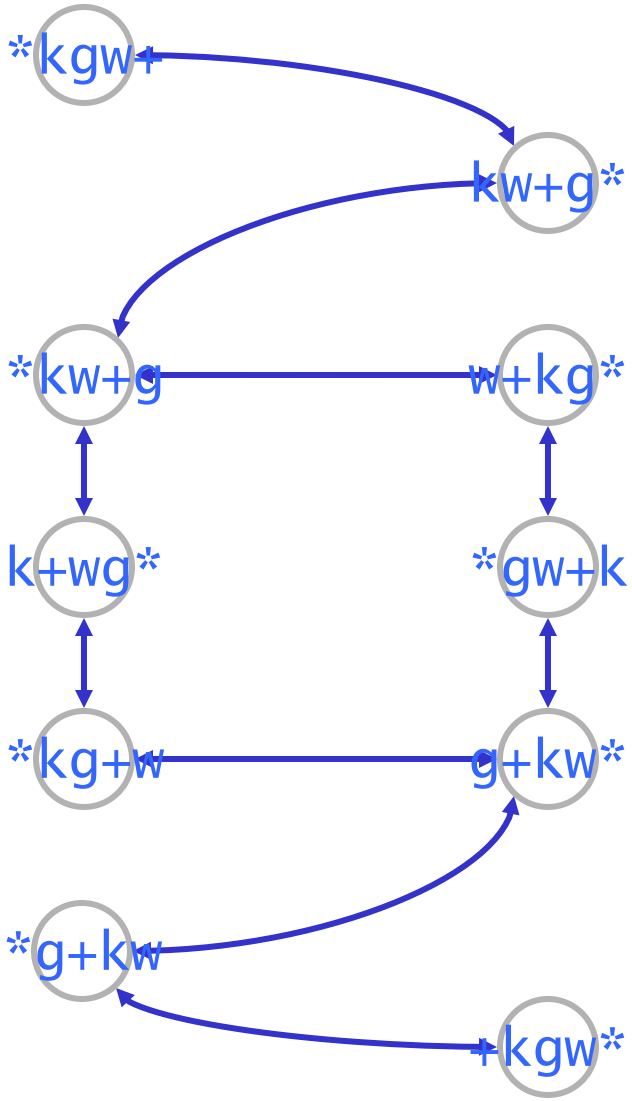
```
const MaxSeq = 1;
type EvType = (FrameArrival, CksumErr, TimeOut );

procedure sender3;
var NextFrameToSend : SequenceNr;
    s : frame;
    buffer : message;
    event : EvType;
begin
    NextFrameToSeend := 0;
    FromHost( buffer );
    repeat
    s.info := buffer;
    s.seq := NextFrameToSend;
    sendf( s );
    StartTimer( s.seq );
    wait( event );
    if event = FrameArrival then
    begin
      FromHost( buffer );
      inc( NextframeToSend );
    end
    until doomsday
end; { sender3 }
```

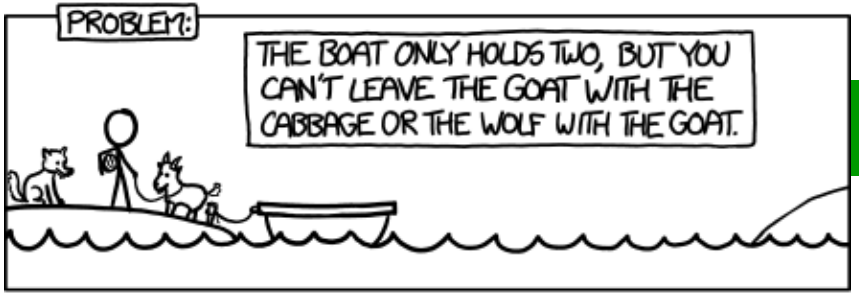


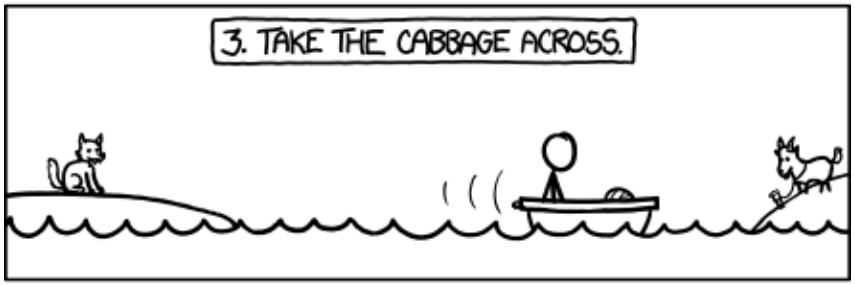
toestanden & acties

kool, geit, wolf

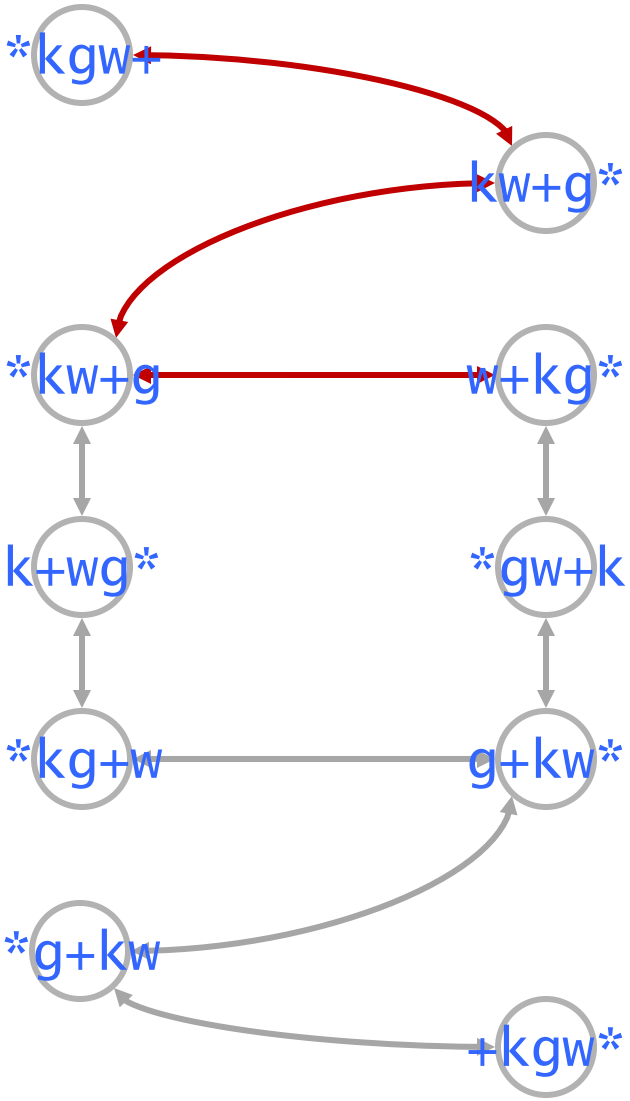








Logic Boat



http://xkcd.com/1134/

Propositiones ad Acuendos Juvenes

Alcuinus van York (York ~735 - Tours 804)

XVIII. PROPOSITIO DE HOMINE ET CAPRA ET LVPO.

Homo quidam debebat ultra fluuium transferre lupum, capram, et fasciculum cauli. Et non potuit aliam nauem inuenire, nisi quae duos tantum ex ipsis ferre ualebat. Praeceptum itaque ei fuerat, ut omnia haec ultra illaesa omnino transferret. Dicat, qui potest, quomodo eis illaesis transire potuit?

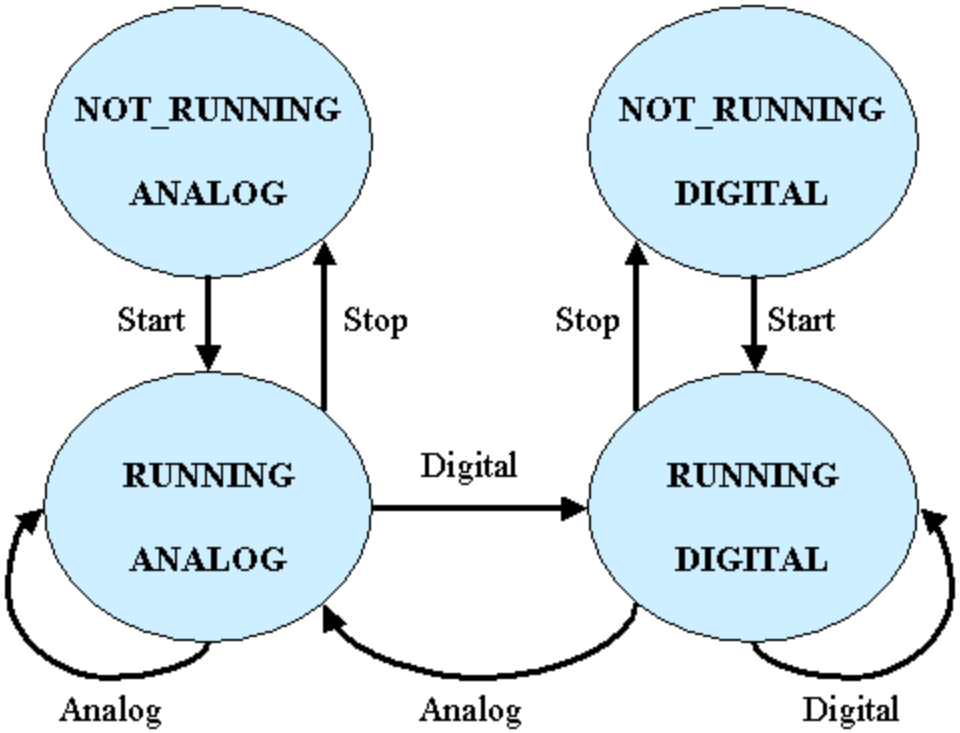
Solutio

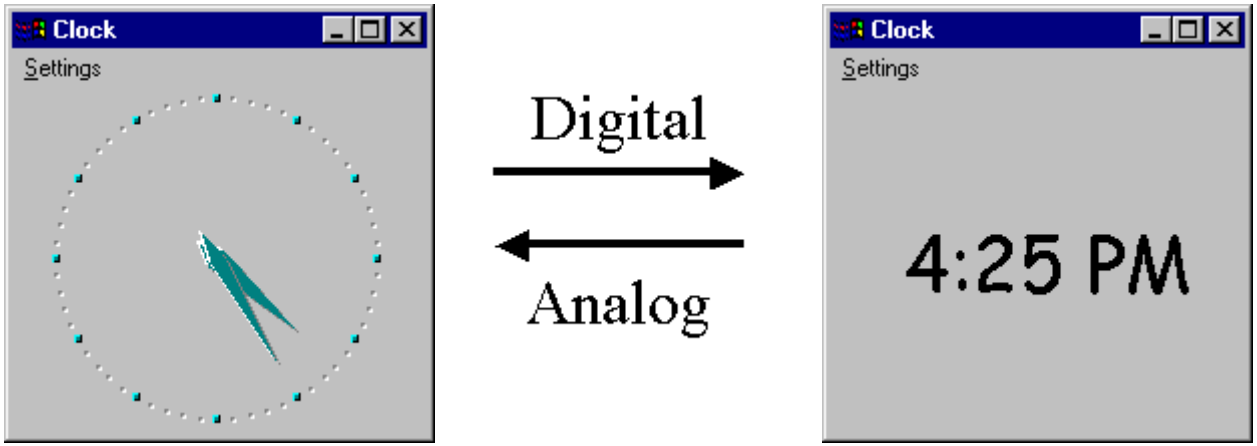
Simili namque tenore ducerem prius capram et dimitterem foris lupum et caulum. Tum deinde uenirem, lupumque transferrem: lupoque foris misso capram naui receptam ultra reducerem; capramque foris missam caulum transueherem ultra; atque iterum remigassem, capramque assumptam ultra duxissem. Sicque faciendo facta erit remigatio salubris, absque uoragine lacerationis.

http://en.wikipedia.org/wiki/Propositiones_ad_acuendos_juvenes

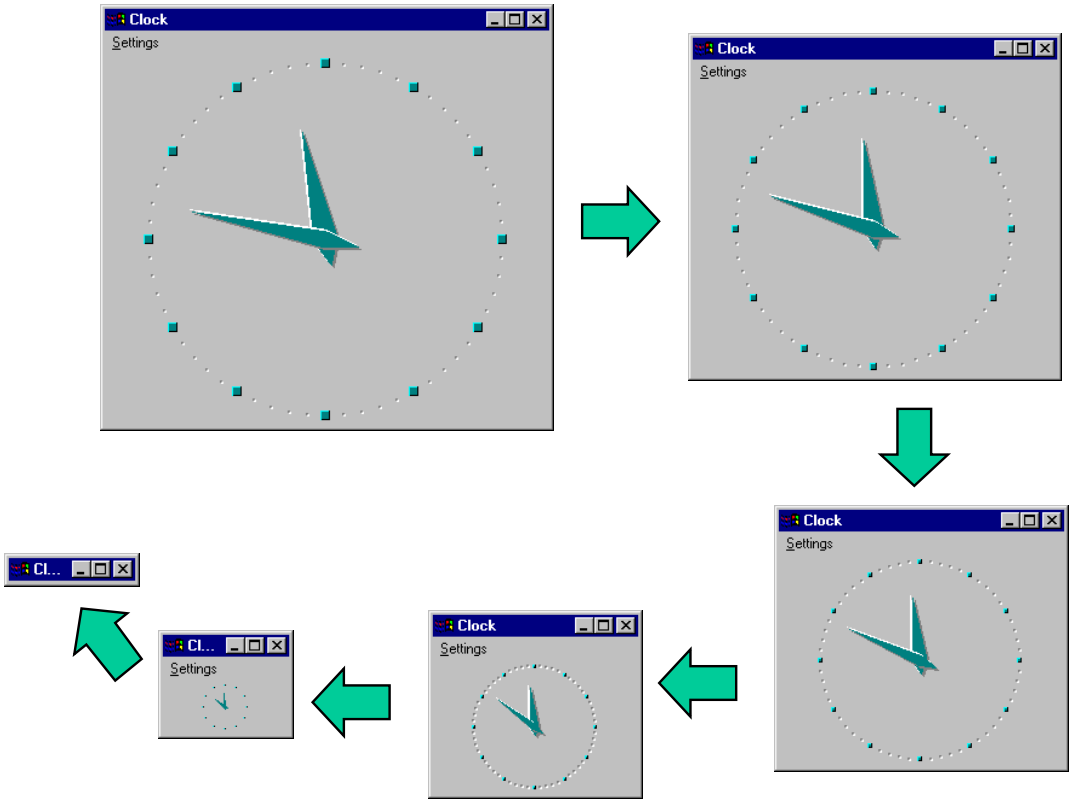
beschrijving

http://www.geocities.com/harry_robinson_testing/shoestring.doc





The Incredible Shrinking Clock Bug

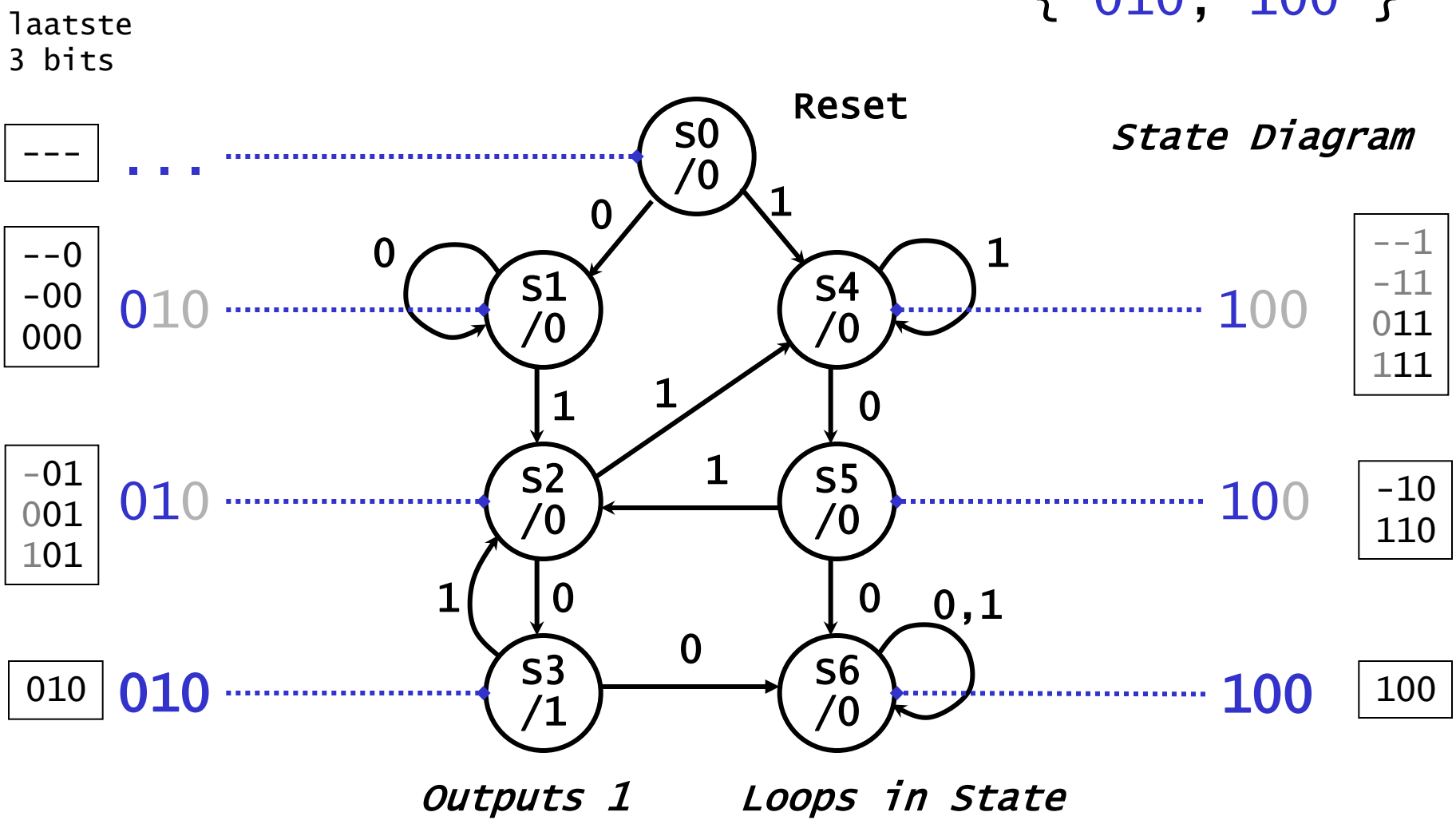


Start
Maximize
Stop
Start
Minimize
Stop
Start
Restore
Stop

… nog een web-voorbeeld

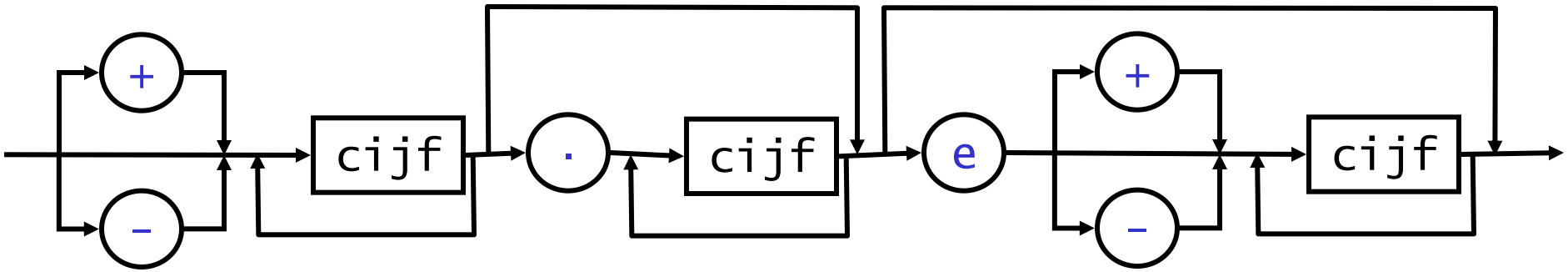
finite State Machine Word Problems

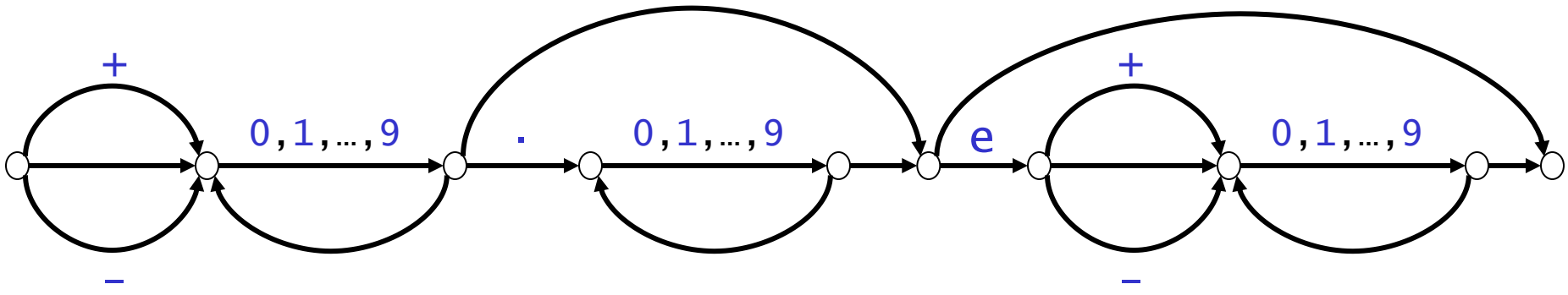
{ 010, 100 }



syntax diagram

getal





$$\{+,-,\lambda\} \cdot \{0,1,\ldots,9\}^{+} \cdot (\{\lambda\} \cup \{.\} \cdot \{0,1,\ldots,9\}^{+}) \cdot (\{\lambda\} \cup \{e\} \cdot \{+,-,\lambda\} \cdot \{0,1,\ldots,9\}^{+})$$

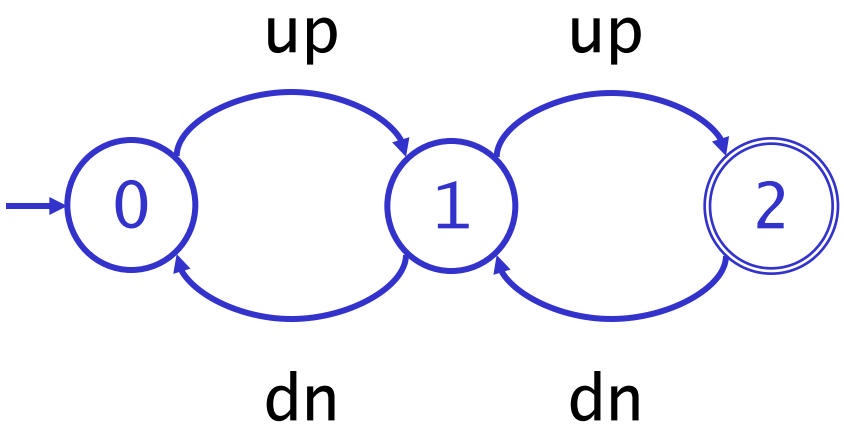
modelleren

- ontwerpen
- gedrag bestuderen
- fouten vinden
- correct bewijzen

eindige automaat

gerichte graaf

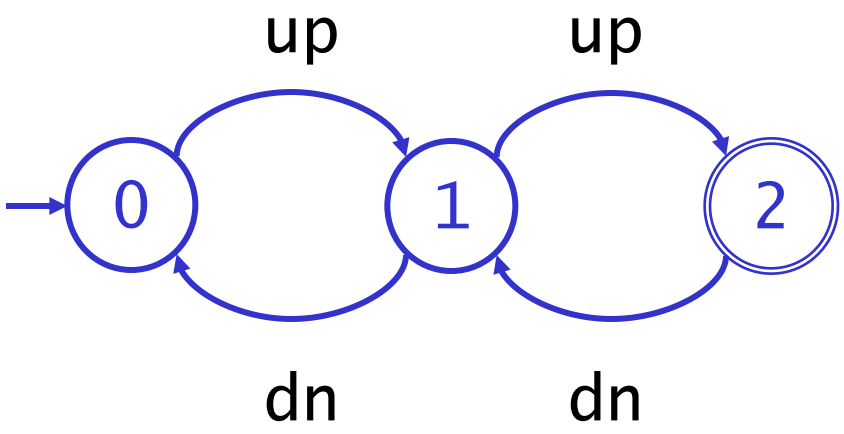
- knopen ~ toestanden
- pijlen met labels ~ acties
- begin- en eindtoestanden



definitie

een *eindige automaat* is een vijf-tupel
$\mathcal{A} = (Q, \Sigma, E, q_{in}, F)$ waarbij

- Q een eindige verzameling *toestanden*
- Σ een *alfabet*
- $E \subseteq Q \times \Sigma \times Q$ verzameling *takken*
- $q_{in} \in Q$ de *begintoestand*
- $F \subseteq Q$ de *eindtoestanden*



$Q = \{ 0,1,2 \}$
$\Sigma = \{ up,dn \}$
$E =$
$\{ (0,up,1), (1,up,2),$
$(2,dn,1), (1,dn,0) \}$
$q_{in} = 0$
$F = \{ 2 \}$

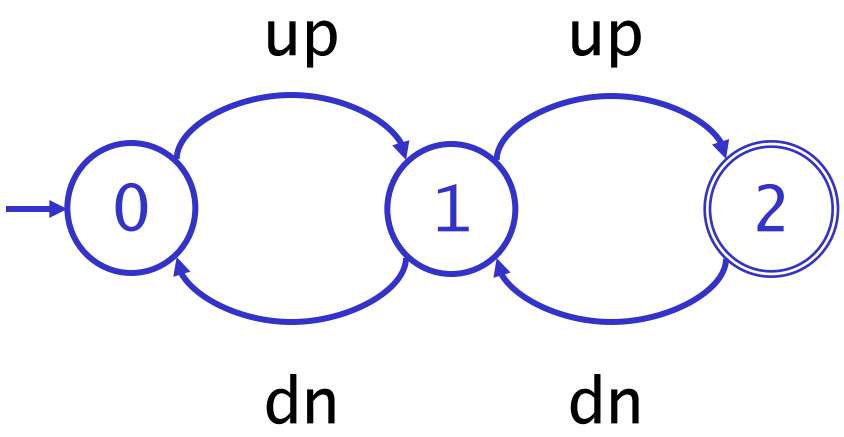
gedrag

$\mathcal{A} = (Q, \Sigma, E, q_{in}, F)$

wandeling : afwisselend knopen (toestanden) en pijlen (takken)

$p_0\ (p_0, a_0, p_1)\ p_1\ (p_1, a_2, p_2)\ \ldots\ p_{n-1}\ (p_{n-1}, a_n, p_n)\ p_n$

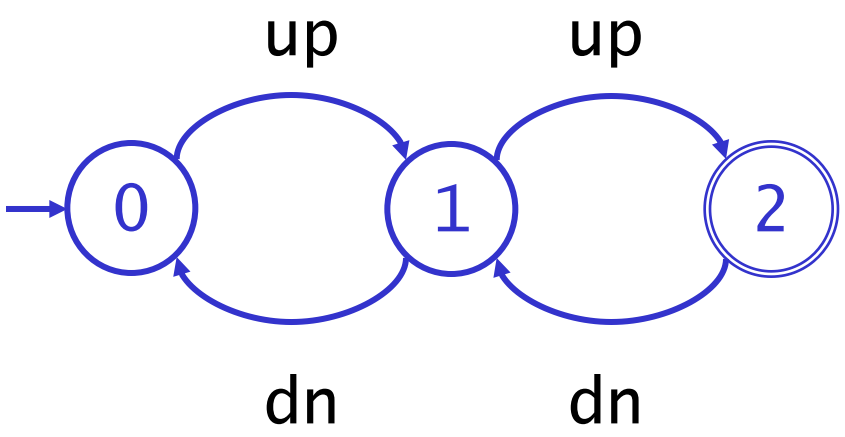
met *label* $a_0 a_2 \ldots a_n$



$1 \xrightarrow{up} 2 \xrightarrow{dn} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1$

gedrag: taal

$\mathcal{A} = (Q, \Sigma, E, q_{in}, F)$ eindige automaat
de *taal* van $\mathcal{A}$, genoteerd $L(\mathcal{A})$, is
{ $x \in \Sigma^*$ | x is een label van een wandeling
van q_{in} naar een toestand in F }



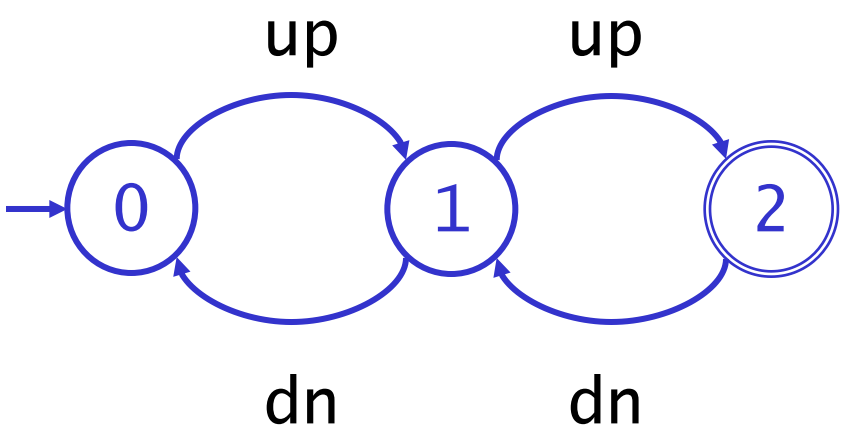
$1 \xrightarrow{up} 2 \xrightarrow{dn} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1$

$0 \xrightarrow{up} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1 \xrightarrow{up} 2$

$0 \xrightarrow{up} 1 \xrightarrow{dn} 0 \xrightarrow{dn} ? \xrightarrow{up} ?$

gedrag: taal

$\mathcal{A} = (Q, \Sigma, E, q_{in}, F)$ eindige automaat
de *taal* van $\mathcal{A}$, genoteerd $L(\mathcal{A})$, is
{ $x \in \Sigma^*$ | x is een label van een wandeling
van q_{in} naar een toestand in F }



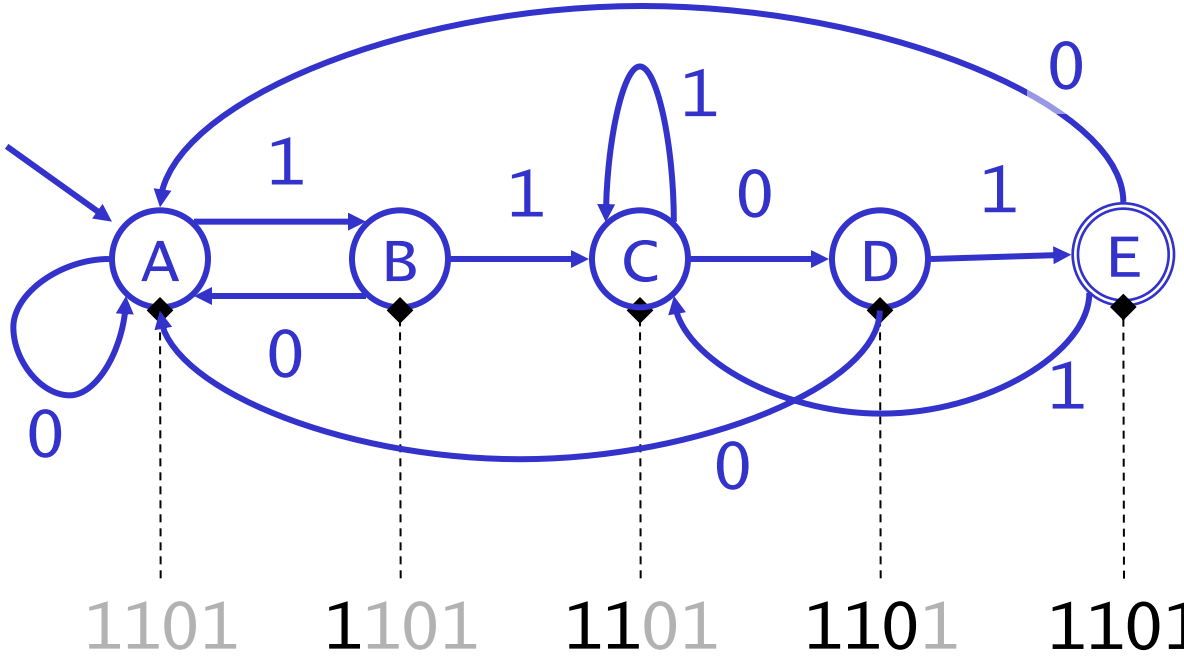
$1 \xrightarrow{up} 2 \xrightarrow{dn} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1$

$0 \xrightarrow{up} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1 \xrightarrow{up} 2$

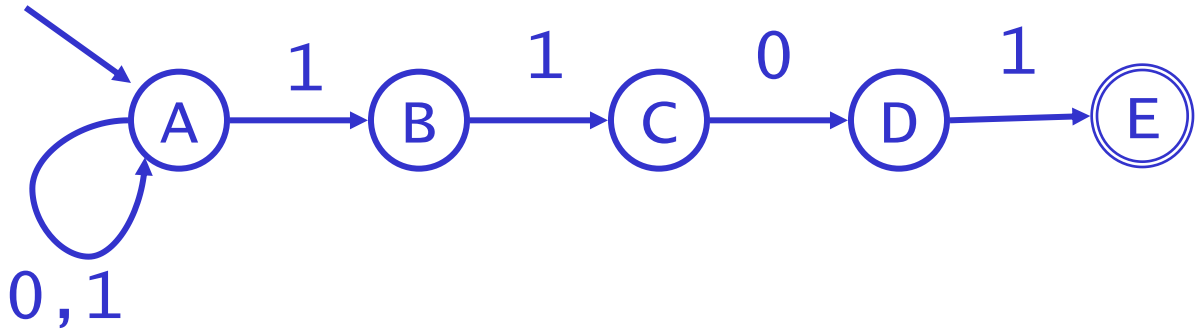
$0 \xrightarrow{up} 1 \xrightarrow{dn} 0 \xrightarrow{dn} ? \xrightarrow{up} ?$

algoritmisch vs. beschrijvend

suffix 1101



algoritme



beschrijving

Dite

Over het opstellen van automaten:

The first thing you have to figure out is precisely how the use of state will help you to solve the given problem.

This is usually the most difficult step. Why?

- *There is not a formal procedure how to derive a state table or state diagram from a problem specification such as the one we have here.*
- *In Step 1 you have to relay on your knowledge and design experience.*
- *Currently, Step 1 is more an art than a science!*

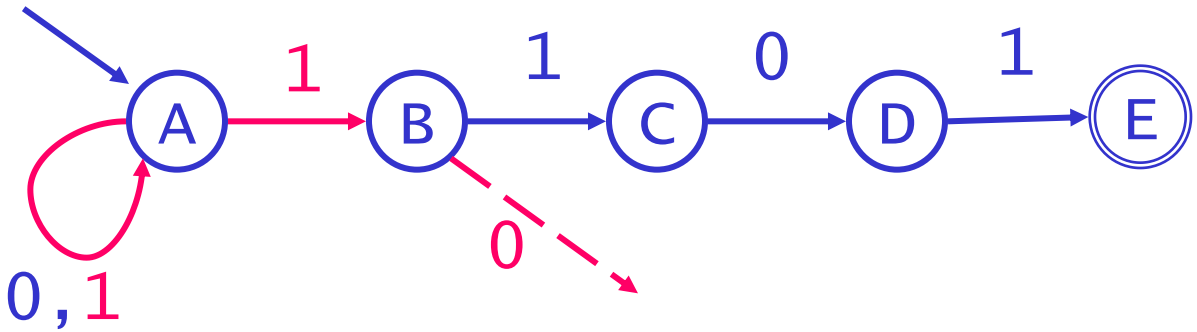
NB. Voor het opstellen van automaten die specifieke patronen herkennen is er wél een algoritme.
(FI2: constructie NFA → DFA,
DATASTR: algoritme Knut-Morris-Pratt.)

algoritmisch

> ‘on the fly’ algoritme
> steeds van bewandeld prefix weten of dat zèlf tot de taal behoort
> geen keuze & niet vastlopen

definitie

een eindige automaat $\mathcal{A} = (Q,\Sigma,E,q_{in},F)$ heet *deterministisch* als voor elke toestand $p \in Q$ en elke letter $a \in \Sigma$ er precies één tak $(p,a,q) \in E$ is.



enige voorbeelden

talen over {0,1}

1) aantal 1-en is een drievoud

2) { 01, 011, 0111 }*

tentamen fi2

$x \in \{0,1\}^*$ $val(x) \in \mathbb{N}$

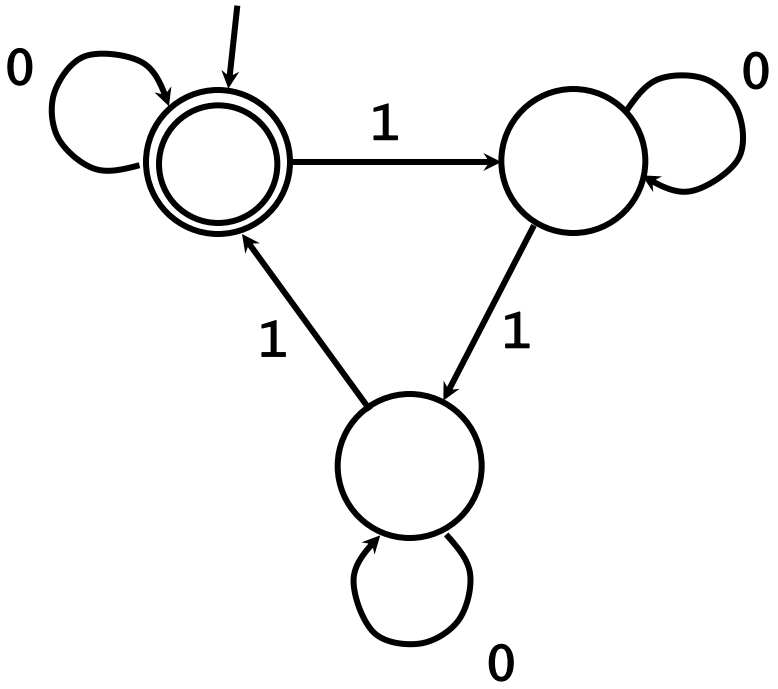
waarde als binair getal

val(1101) = 13

3) val(x) is een drievoud

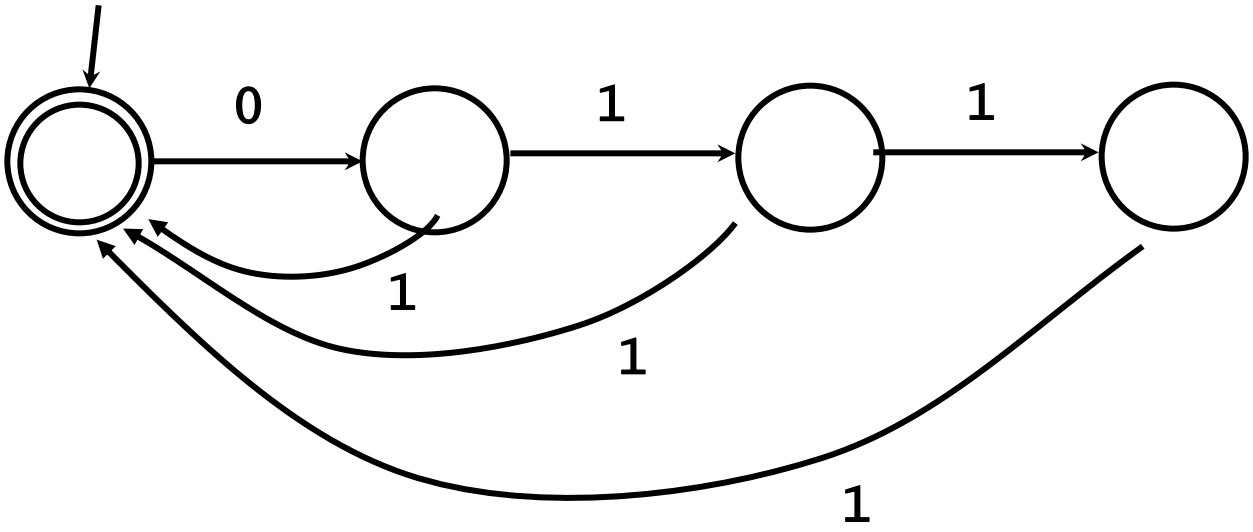
zie dictaat fi1

aantal 1-en is een drievoud



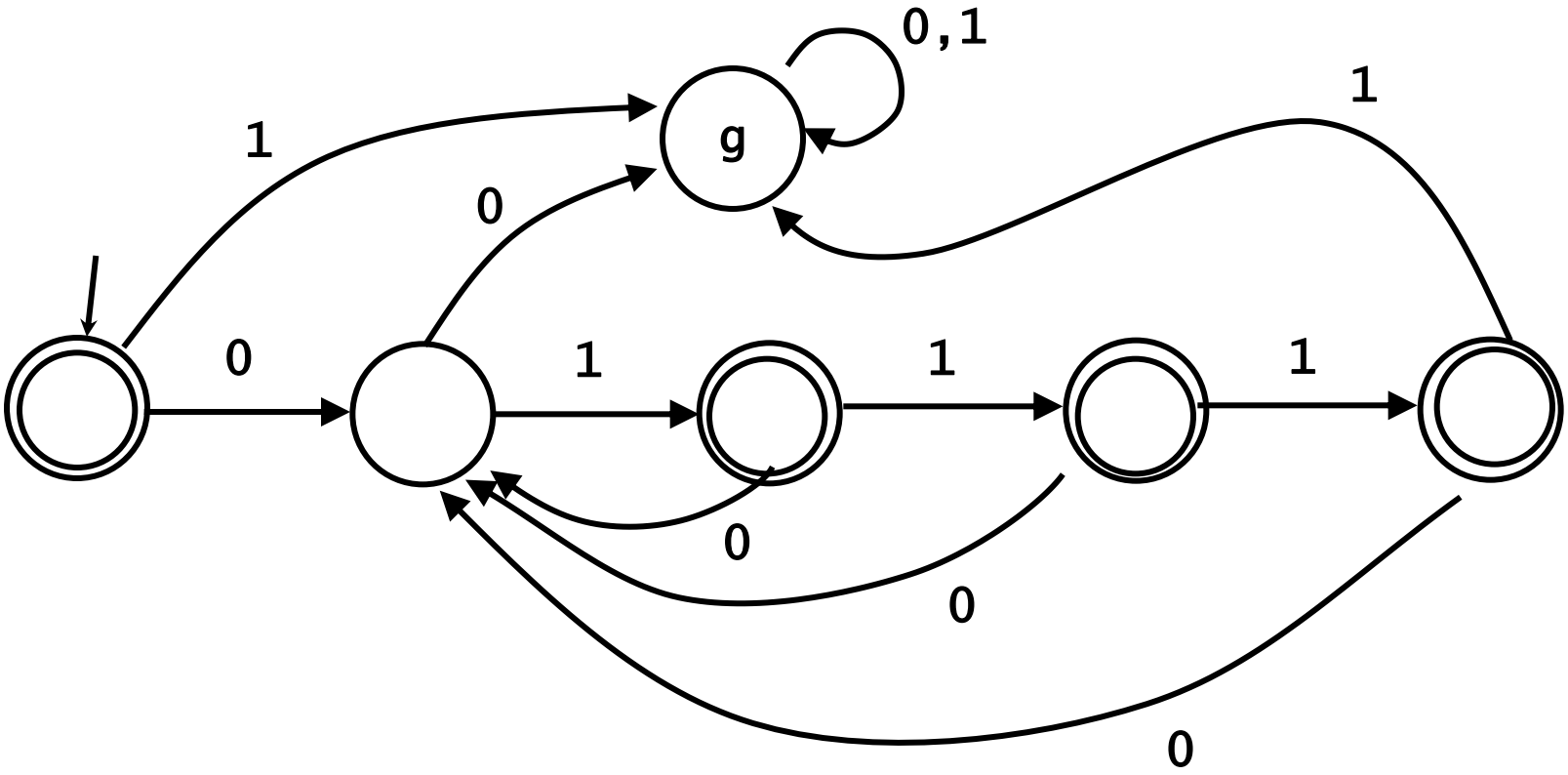
ik maak die automaten liever op het bord;
hier de antwoorden voor wie thuis meeleest.

{ 01, 011, 0111 }*



niet deterministisch

{ 01, 011, 0111 }*



deterministisch

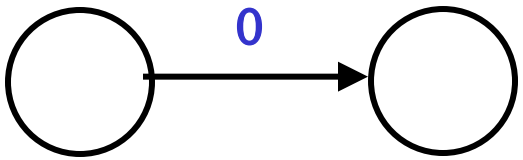
inclusief ‘garbage’ toestand

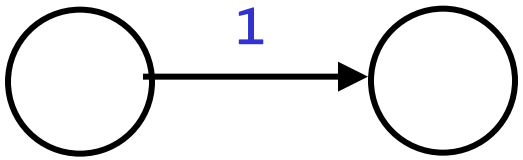
val(x) is een drievoud

recept

toestand na lezen van $x \in \{0,1\}^*$

$v=val(x) \in \mathbb{N}$ modulo 3





x	x0	x	x1
v	2v	v	2v+1
0	0	0	1
1	2	1	0
2	1	2	2

(input) string $\{0,1\}^*$

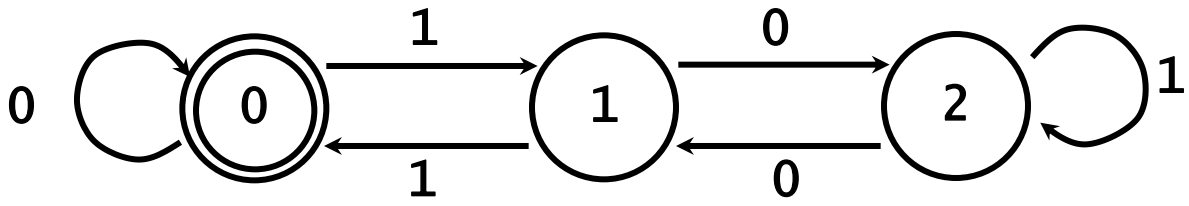
waarde $\mathbb{N}$

waarde (mod 3) toestand

val(x) is een drievoud

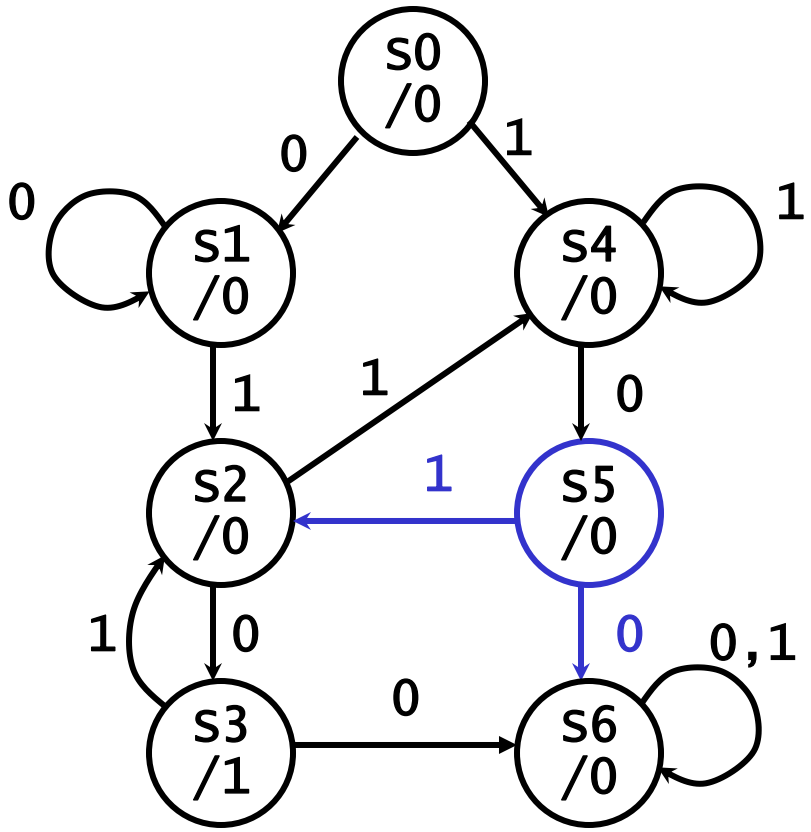
uitgewerkt

toestand na lezen van x
val(x) modulo 3



correctheid

toestand ⇔ eigenschap woord



{ 010, 100 }

woorden |w|≥3 :		
state	suffix	100
s1	000	niet
s2	.01	niet
s3	010	niet
s4	.11	niet
s5	110	niet
s6	(100)	wél

inductiestap

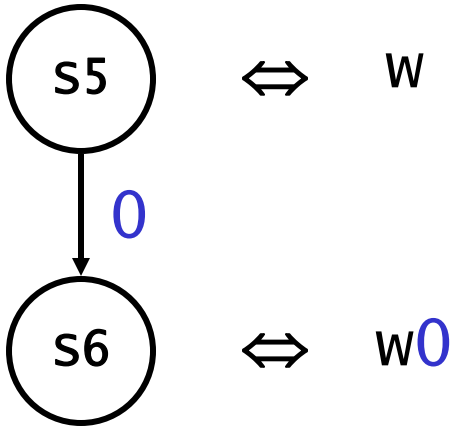
	w ↦	w0	w1
state	s5	s6	s2
suffix	110	1100	1101
subwrd	niet	wel	niet

correctheid

formuleer een ‘**invariant**’

een relatie tussen (eigenschappen van)
woord en bereikte toestand in automaat

bewijs dat die relatie behouden blijft



state	suffix	100
s5	110	niet
s6	(100)	wél

… inductie

meerstapsrelatie

eindige automaat $\mathcal{A} = (Q,\Sigma,E,q_{in},F)$

$E \subseteq Q \times \Sigma \times Q$ éénstapsrelatie

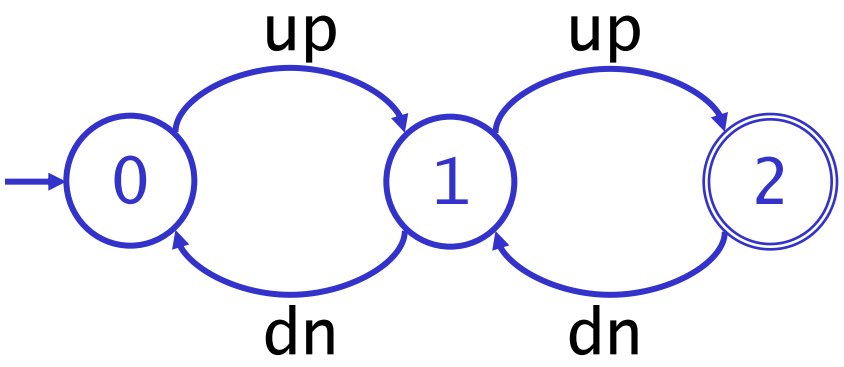
$E^* \subseteq Q \times \Sigma^* \times Q$ meerstapsrelatie

basis $(p,\lambda,p) \in E^*$ voor elke $p \in Q$.

inductiestap

als $(p,w,q) \in E^*$ en $(q,a,r) \in E$

dan $(p,wa,r) \in E^*$



$1 \xrightarrow{up} 2 \xrightarrow{dn} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1$

E	E*
	(1,λ,1)
(1,up,2)	(1,up,2)
(2,dn,1)	(1,up dn,1)
(1,dn,0)	(1,up dn dn,0)
(0,up,1)	(1,up dn dn up,1)

takken rijgen

basis $(p,\lambda,p) \in E^*$ voor elke $p \in Q$.

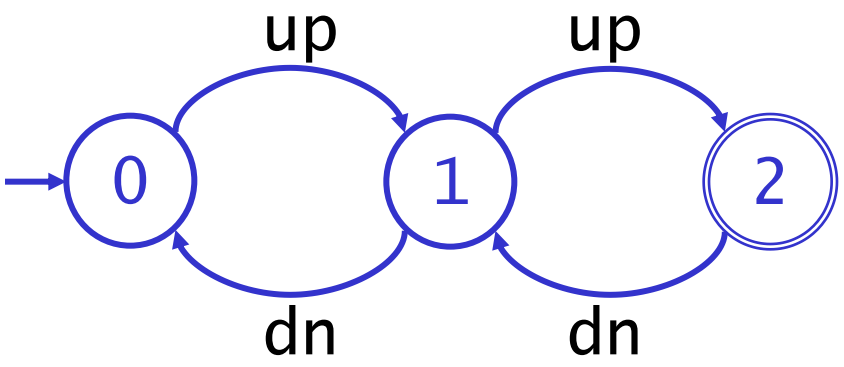
inductiestap

als $(p,w,q) \in E^*$ en $(q,a,r) \in E$

dan $(p,wa,r) \in E^*$

lemma

$(p,w,q) \in E^*$ precies als er een wandeling bestaat van p naar q met label w.



$1 \xrightarrow{up} 2 \xrightarrow{dn} 1 \xrightarrow{dn} 0 \xrightarrow{up} 1$

E	E^*
	$(1,\lambda,1)$
$(1,up,2)$	$(1,up,2)$
$(2,dn,1)$	$(1,up\ dn,1)$
$(1,dn,0)$	$(1,up\ dn\ dn,0)$
$(0,up,1)$	$(1,up\ dn\ dn\ up,1)$

gedrag: taal

{ $x \in \Sigma^*$ | x is een label van een wandeling
van q_{in} naar een toestand in F }

$\mathcal{A} = (Q,\Sigma,E,q_{in},F)$ eindige automaat
de *taal* van $\mathcal{A}$, genoteerd $L(\mathcal{A})$, is
{ $x \in \Sigma^*$ | $(q_{in},x,q) \in E^*$
voor een toestand q in F }

… $q \in \delta(q_{in},x)$ …

representeerbare talen

een taal K heet *representeerbaar* als er een eindige automaat is met $L(\mathcal{A}) = K$

voorbeelden:

- even aantal 1-en
- {01,011,0111}*
- suffix 1101
- x met val(x) drievoud

representeerbare talen

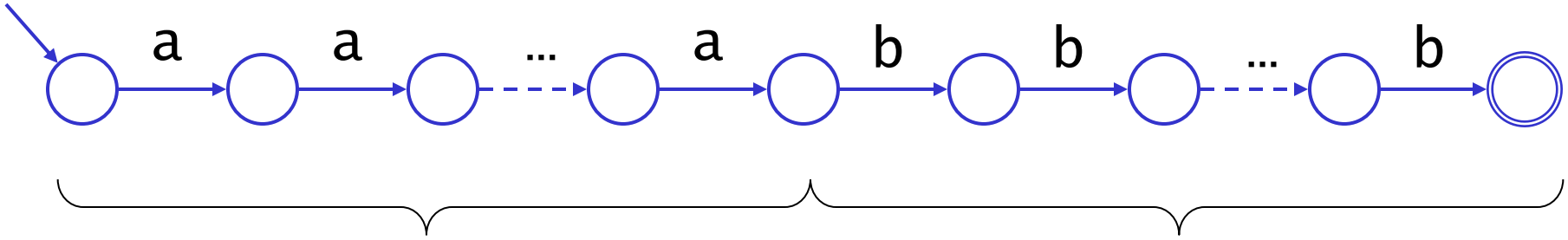
een taal K heet *representeerbaar* als er een eindige automaat is met $L(\mathcal{A}) = K$

eindig veel toestanden … niet elke taal

stelling

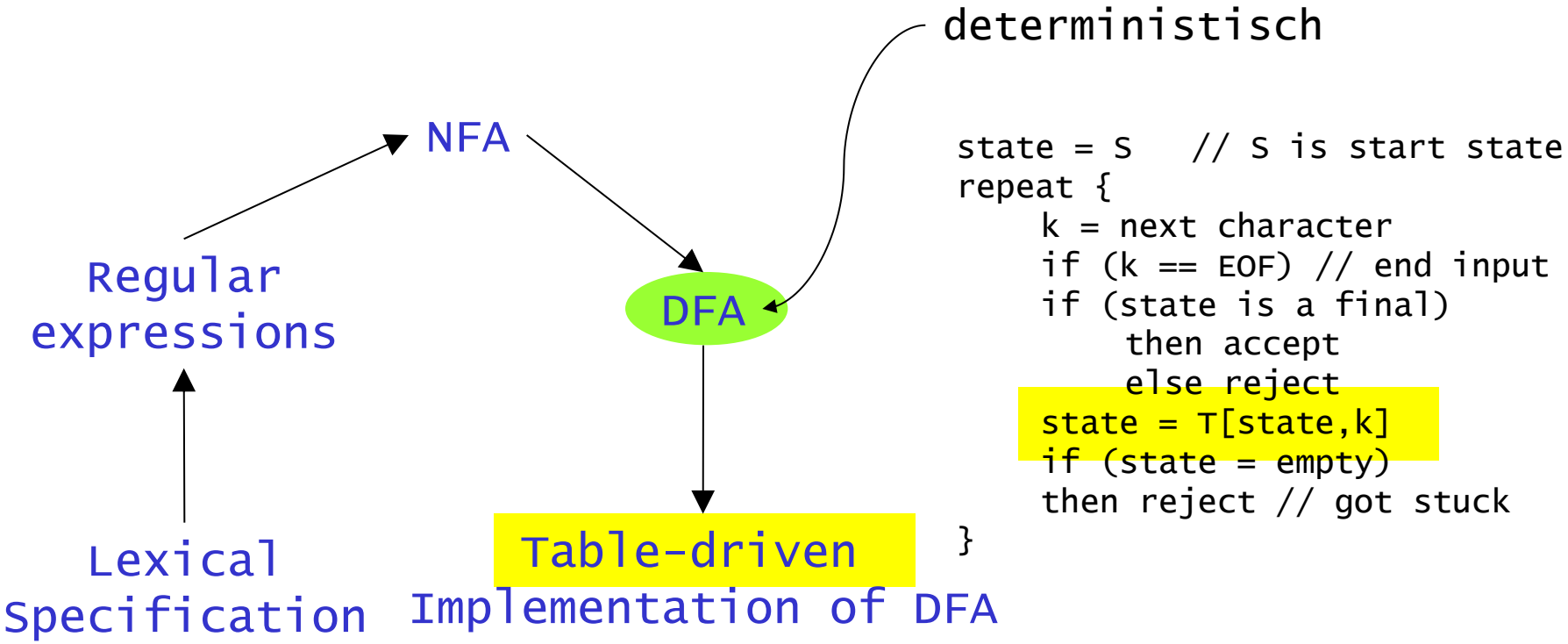
$K = \{ a^n b^n \mid n \geq 1 \}$ is *niet* representeerbaar

stel wél … N toestanden



N takken:
hier een toestand dubbel …

compilers (generating a lexical scanner)

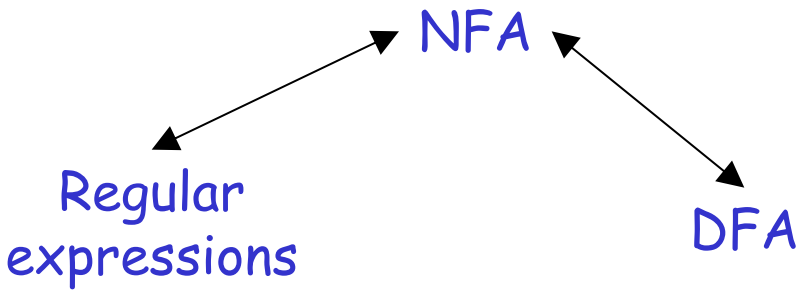


bekende familie

stelling

dezelfde familie van talen wordt gedefinieerd door

- reguliere expressies · ∪ * ‘regulier’
- eindige automaten ‘representeerbaar’
- deterministische eindige automaten
- logische expressies ‘MSO’
- equivalentie op strings ‘Myhill-Nerode’



zie FI2 ‘formele talen’

chomsky hierarchie

automaten	taalfamilies	grammatica's
eindige	regulier	rechtslineair
stapel	context-vrij	context-vrij
lineair begrensd	context-gevoelig	context-gevoelig
Turing machine	recursief recursief opsombaar	type 0

zie FI2 FI3
compilers, complexiteit

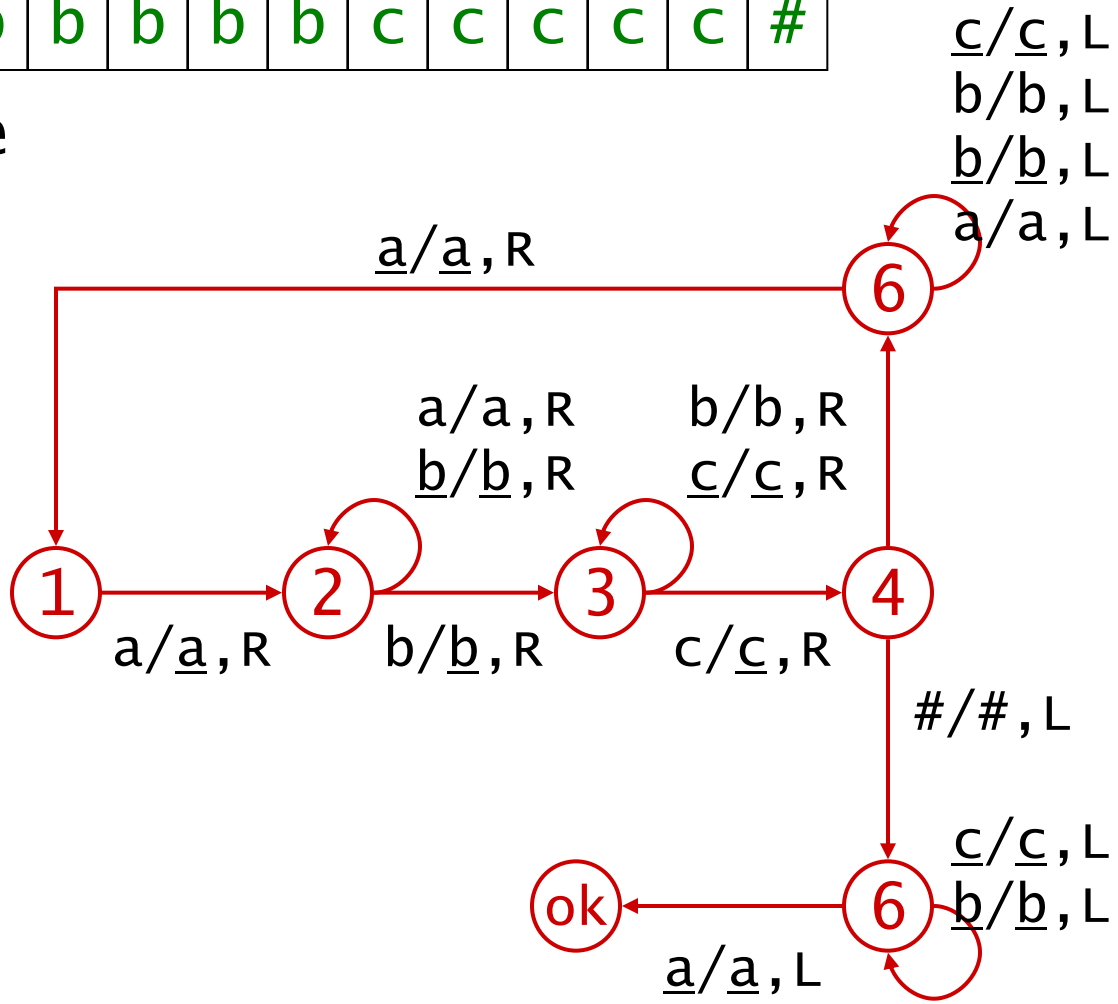
Turing machine

$\{ a^n b^n c^n \mid n \geq 0 \}$

#	a	a	a	a	a	b	b	b	b	b	c	c	c	c	c	#

tape

1. mark a
2. move to b's
   mark b
3. move to c's
   mark c
4. if another c
5. then back to a's
   goto 1.
   else back to a's
6. check marks
   stop



end...

oefen!

http://www.liacs.leidenuniv.nl/~hoogeboomhj/fi1/tentamens/

http://www.liacs.nl/~hoogeboo/fi1/tentamens/