

LEKSIČKA ANALIZA#2

Jezici i prevoditelji

prof. dr. sc. Marjan Sikora

SADRŽAJ:

- ponavljanje
- pretvorba regularnog izraza u deterministički konačni automat

PONAVLJANJE

- želimo napraviti **algoritam za leksiranje**
- leksiranje treba ulazni niz znakova **podijeliti** u tokene
- **tokeni** su definirani **regularnim izrazima**
- prikazali smo algoritam podjele ulaznog niza znakova u tokene

ALGORITAM DIJELJENJA v3.0

```
na ulazu su znakovi  $x_1 \dots x_n$   
za svaki  $n \leq i \leq 1$  provjeri  
je li string  $x_1 \dots x_i \in L(R)$   
ako jest  
    ustanovi za koji najmanji  $j$  vrijedi  
         $x_1 \dots x_i \in L(R_j)$   
    ukloni sa ulaza  $x_1 \dots x_i$ 
```

- ovakav kod uzima u obzir:
 - odabir najduljeg tokena
 - prioritete
- greške se rješavaju kroz dodavanje tokena greške

NERIJEŠENI DIO ALGORITMA

na ulazu su znakovi $x_1 \dots x_n$

za svaki $n \leq i \leq 1$ provjeri

je li string $x_1 \dots x_i \in L(R)$

ako jest

ustanovi za koji najmanji j vrijedi

$x_1 \dots x_i \in L(R_j)$

ukloni sa ulaza $x_1 \dots x_i$

- potrebno je još definirati dio algoritma
- **kako odrediti je li neki dio niza token?**

PREOSTALI DIO ALGORITMA

- potrebno je još definirati dio algoritma
- **kako odrediti je li neki dio niza token?**

PREOSTALI DIO ALGORITMA

```
int Transition[MAXSTATE][MAXSIMBOL];    /* stanje klasa-znaka */
char currentChar;                       /* globalna varijabla - sadrži ulazni znak */
char nextChar();                        /* vraća idući znak sa ulaza */
char CharClass(char ch);                /* vraća ulaznu grupu kojoj znak pripada */
char isFinalState(int state);           /* vraća istinu ako je state konačno stanje */
char token(int state);                  /* vraća kojem tokenu pripada koje stanje */
```

```
int getToken() {
    int state = INITIALSTATE;
    while (currentChar != EOF) {
        state = Transition[state][CharClass(currentChar)];
        if (state == ERROR || isFinalState(state))
            break;
        currentChar = nextChar();
    }
    if ( isFinalState(state) && currChar == EOF )
        return(token(state));
    else
        return(ERROR_TOKEN);
}
```

stanje	d	.	konačno
A	D	B	ne
B	C	greška	ne
C	C	greška	da
D	D	C	da

PREOSTALI DIO ALGORITMA

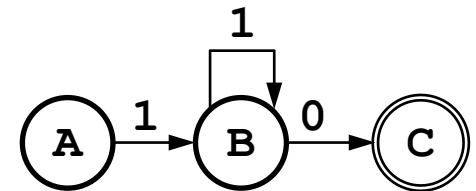
- još ostaje **jedan korak**
- token je na prethodnom slajdu **definiran DKA**
- to je **nezgrapno**
- token treba biti definiran **regularnim izrazom**
- **kako pretvoriti regularni izraz u DKA?**

PROBLEM

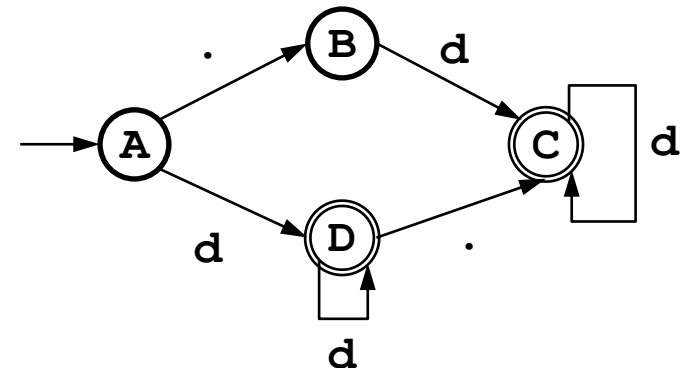
pitanje je kako regularni izraz
pretvoriti u DKA, odnosno
tablicu stanja?

ako to znamo onda imamo
algoritamski riješeno
leksiranje!

1+0



$(d^* . d^+)$ |
 $(d^+ (\epsilon | .))$



RI->NKA->DKA

- pretvorba regularnog izraza u DKA se vrši posredno
- **posrednik je nederministički konačni automat (NKA)**

DKA VS. NKA

prethodno opisani automat naziva se
deterministički konačni automat (DKA)



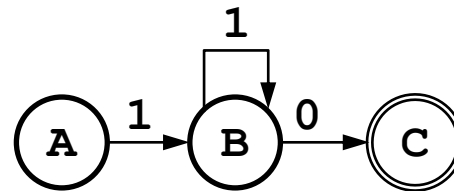
kod DKA automata je uvijek **jednoznačno** određen
prijelaz u neko stanje

kada ulazni znak vodi u više različitih stanja imamo
nedeterministički konačni automat (NKA)

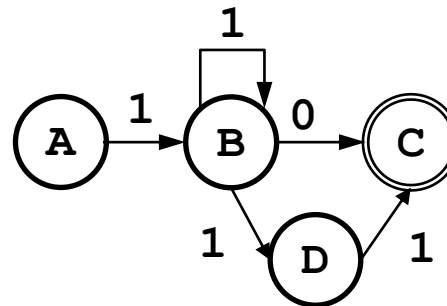
NKA se **ne može** realizirati pomoću **tablice**
prijelaza

NEDETERMINISTIČKI KONAČNI AUTOMAT

- kod DKA određeni simbol na ulazu uvijek uzrokuje prijelaz u samo jedno određeno stanje:

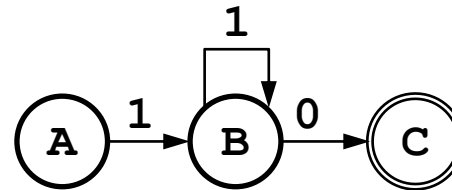


- kod NKA određeni simbol na ulazu može uzrokovati prijelaz u više stanja

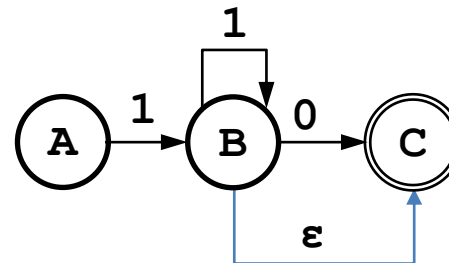


NEDETERMINISTIČKI KONAČNI AUTOMAT

- DKA ne može prijeći u drugo stanje bez da sa ulaza konzumira jedan simbol:



- NKA može prijeći u drugo stanje bez da konzumira simbol sa ulaza:



NEDETERMINISTIČKI KONAČNI AUTOMAT

- mane NKA:
 - **ne može** se realizirati pomoću **tablice** prijelaza
 - **sporiji** je u implementaciji (kod DKA je sve pravocrtno)
- prednosti:
 - NKA je **manji** (eksponencijalno) od DKA
- DKA/NKA predstavljaju kompromis **vrijeme/prostor**
- NKA ćemo koristiti kao **prijelaznu fazu** u pretvaranju regularnih izraza u DKA

RI->NKA->DKA

1. **pretvorba** svakog pojedinačnog **regularnog izraza** u ekvivalentne **NKA**
 2. **spajanje** pojedinačnih automata u jedan **NKA**
 3. **pretvorba NKA u DKA**
 4. **stvaranje tablice** prijelaza iz DKA
(obično se dobiju **velike tablice**, s puno stanja)
 5. **minimiziranje** broja stanja i prijelaza
- ostalo je samo **umetnuti tablicu** u prethodno definirani algoritam

KONSTRUKCIJA LEKSIČKOG ANALIZATORA IZ SPECIFIKACIJE

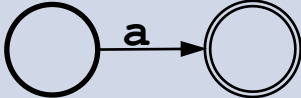
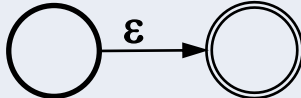
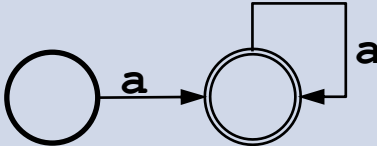
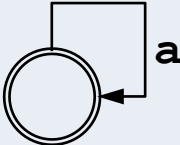
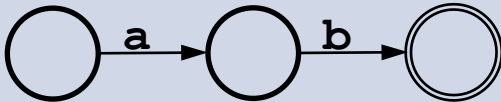
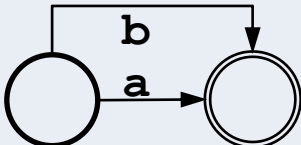
...

4. nakon šta imamo **jedan DKA** vrši se **izgradnja tablice** prijelaza (obično se dobiju **velike tablice**, s puno stanja)
 5. **minimiziranje** broja stanja i prijelaza
- ovime je izgradnje leksičkog analizatora je **završena**
 - ostalo je samo **umetnuti tablicu** u prethodno definirani program koji simulira rad konačnog automata

KONSTRUKCIJA NKA IZ REGULARNIH IZRAZA

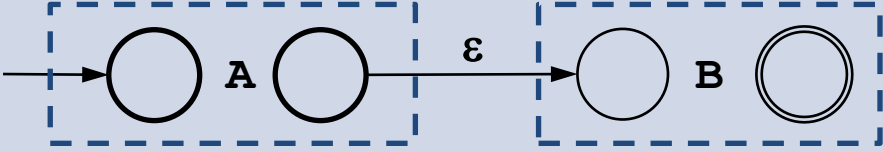
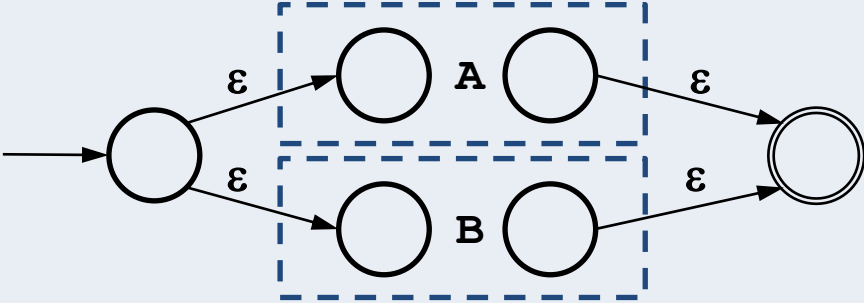
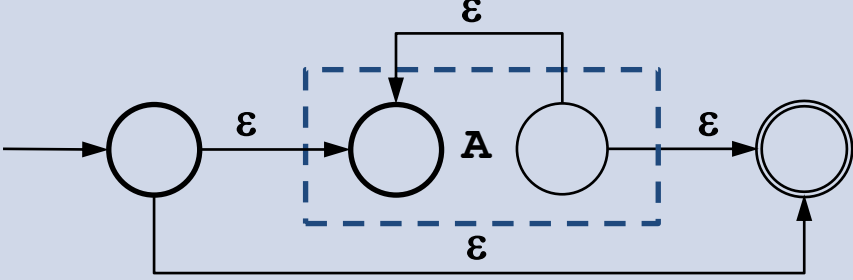
- svaki regularni izraz sastoji se od **više temeljnih regularnih izraza**
- oni se potom **kombiniraju** u nove, složenije regularne izraze
- **najprije** se iz regularnih izraza u automate pretvaraju temeljni regularni izrazi: **a** i **ϵ**
- zatim se oni kombiniraju, pretvarajući u automate:
 - dopune **ab** , alternative **$a | b$** i ponavljanja **a^*** i **a^+**
- na taj način može se formirati **NKA** za kompleksne regularne izraze

KONSTRUKCIJA AUTOMATA IZ TEMELJNIH REGULARNIH IZRAZA

regularni izraz	ekvivalentni DKA
a	
ϵ	
a^+	
a^*	
$a b$	
$a \mid b$	

KONSTRUKCIJA AUTOMATA KOMBINIRANJEM TEMELJNIH REGULARNIH IZRAZA

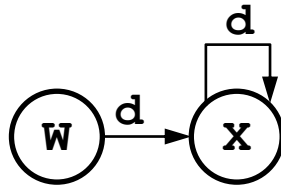
- složeni izrazi** se transformiraju dodavanjem ε -prijelaza

regularni izraz	ekvivalentni DKA
AB	
A B	
A*	

KONSTRUKCIJA NKA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d^+ (\epsilon | .)) \mid (d^* . d^+)$
- najprije se definiraju **dva zasebna automata** koji predstavljaju alternativne regularne izraze $(d^+ (\epsilon | .))$ i $(d^* . d^+)$

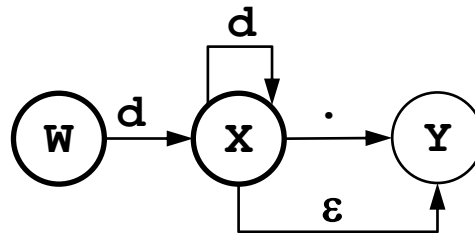
d^+



KONSTRUKCIJA KONAČNOG AUTOMATA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d^+ (\epsilon | \cdot)) \mid (d^* \cdot d^+)$
- najprije se definiraju **dva zasebna automata** koji predstavljaju alternativne regularne izraze $(d^+ (\epsilon | \cdot))$ i $(d^* \cdot d^+)$

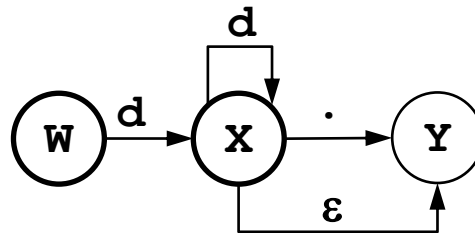
$d^+ (\epsilon | \cdot)$



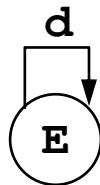
KONSTRUKCIJA KONAČNOG AUTOMATA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d^+ (\epsilon | .)) \mid (d^* . d^+)$
- najprije se definiraju **dva zasebna automata** koji predstavljaju alternativne regularne izraze $(d^+ (\epsilon | .))$ i $(d^* . d^+)$

$d^+ (\epsilon | .)$



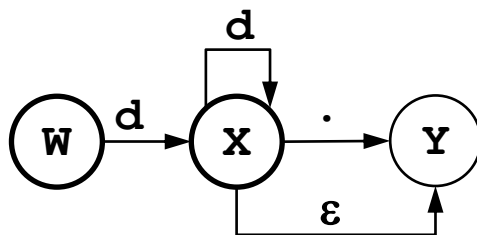
d^*



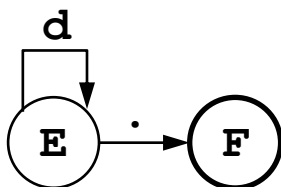
KONSTRUKCIJA KONAČNOG AUTOMATA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d^+ (\epsilon | .)) \mid (d^* . d^+)$
- najprije se definiraju **dva zasebna automata** koji predstavljaju alternativne regularne izraze $(d^+ (\epsilon | .))$ i $(d^* . d^+)$

$d^+ (\epsilon | .)$



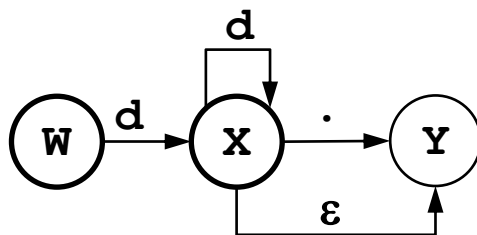
$d^* .$



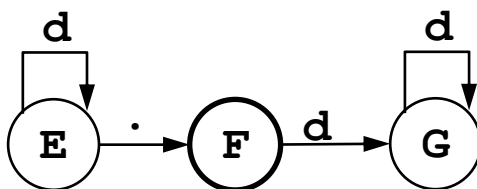
KONSTRUKCIJA KONAČNOG AUTOMATA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d^+ (\epsilon | .)) \mid (d^* . d^+)$
- najprije se definiraju **dva zasebna automata** koji predstavljaju alternativne regularne izraze $(d^+ (\epsilon | .))$ i $(d^* . d^+)$

$d^+ (\epsilon | .)$

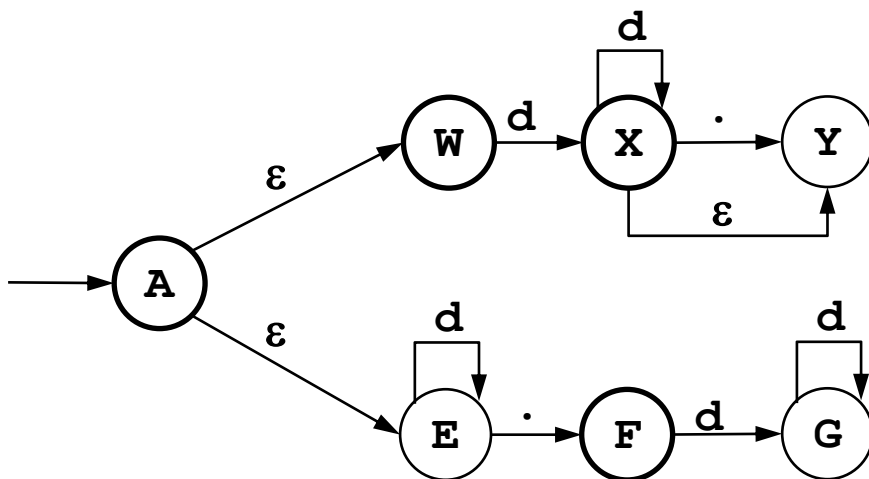


$d^* . d^+$



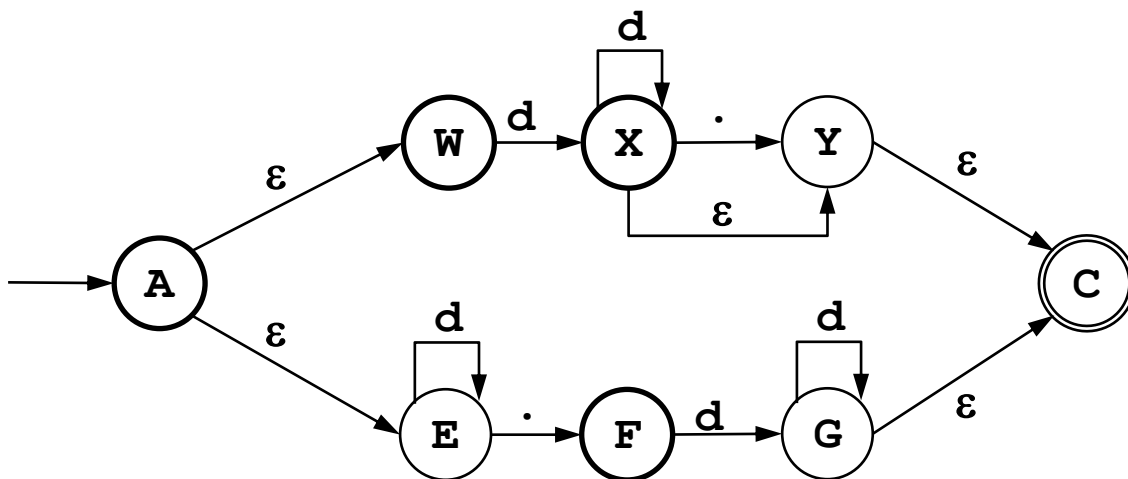
KONSTRUKCIJA KONAČNOG AUTOMATA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d^+ (\epsilon | .)) \mid (d^* . d^+)$
- potom se **povezuju** zasebna automata $(d^+ (\epsilon | .))$ i $(d^* . d^+)$



KONSTRUKCIJA KONAČNOG AUTOMATA IZ REGULARNIH IZRAZA – PRIMJER #2

- proces izgradnje automata za slijedeći regularni izraz:
 $(d+(\epsilon | .)) \mid (d^*.d+)$
- potom se **povezuju** zasebna automata $(d+(\epsilon | .))$ i $(d^*.d+)$



PRETVORBA NKA U DKA

u prethodnom primjeru
konstruirali smo konačni
automat iz **regularnog izraza**

kao rezultat dobili smo **NKA**,
jer sadrži **ϵ -prijelaze**

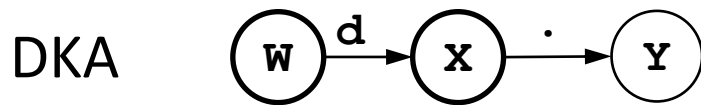
tablica prijelaza za konačni
automat, može se definirati
samo za DKA

nužno je **pretvoriti** NKA u DKA

tek tada će se moći **programski
realizirati** konačni automat
pomoću tablice prijelaza

DKA I TABLICA PRIJELAZA

- kod **DKA** svakom prijelazu odgovara samo jedno stanje
- tu činjenicu u tablici prijelaza opisujemo s **parom (stanje, simbol)**



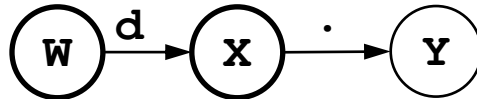
`prijelaz[W, d] = X`
`prijelaz[X, .] = Y`

- tablica DKA **sadrži** odredišno stanje
- ono je pohranjeno pod **dva indeksa** - **izvornim stanjem** i **simbolom** koji uzrokuje prijelaz

`prijelaz[izvorno stanje, simbol] = odredišno stanje`

NKA – PROBLEM S TABLICOM PRIJELAZA

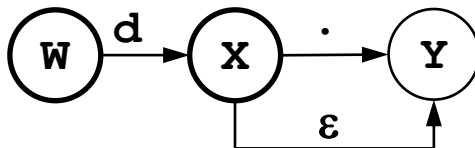
DKA



prijelaz[W, d] = X
prijelaz[X, .] = Y

- kod **NKA** nekom prijelazu pripada **više stanja**
- **problem** jer se u tablici parom (stanje, simbol) može pohraniti samo **jedno stanje**

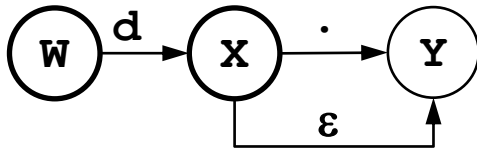
NKA



prijelaz[W, d] = X
prijelaz[W, d] = Y
prijelaz[X, .] = Y

NKA SADRŽI SKUPOVE STANJA

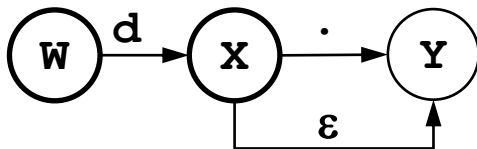
- rješenje je da se tablica prijelaza **preuredi**
- kod **NKA** nekom prijelazu pripada **više stanja**
- **NKA** tablica prijelaza umjesto **jednog stanja** treba sadržavati **skupove stanja**



prijelaz[W, d] = X
prijelaz[W, d] = Y
prijelaz[X, .] = Y

NKA JE DKA KOJI SADRŽI SKUPOVE STANJA

- dakle **tablica NKA se preradi** tako da sadrži skupove stanja umjesto pojedinačnih stanja
- **na taj način ovo postaje DKA**



prijelaz[W, d] = X

prijelaz[W, d] = Y

prijelaz[X, .] = Y

prijelaz[{W}, d] = {X, Y}

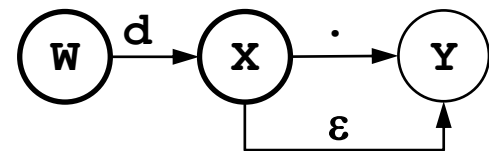
prijelaz[{X}, .] = {Y}

PRETVOBA NKA U DKA

- pretvorba NKA u DKA se provodi pomoću algoritma **konstrukcija podskupa** (eng. *subset construction*)
- njime se dobije DKA koji kao **stanja** ima **podskupove stanja** NKA
- autori algoritma su **Aho, Sethi i Ullman**

PRETVOBA NKA U DKA

- prije samog algoritma uvesti ćemo neke **operacije** i **oznake** za **NKA**:
 - neka **s** predstavlja **stanje** NKA
 - **T** **skup** stanja NKA
- primjerice:
 - **s** = **W** ili **s** = **X**
 - **T** = {**W**} ili **T** = {**X**, **Y**}



OPERACIJE PRETVOBE NKA U DKA

- uvesti ćemo **tri operacije**
- ove operacije **rezultiraju skupovima stanja NKA**

$$\varepsilon\text{-closure}(s) = \{s\} \cup \{T\}$$

$$\varepsilon\text{-closure}(T) = \cup \varepsilon\text{-closure}(s)$$

$$\text{move}(T, a)$$

OPERACIJE PRETVOBE NKA U DKA – OPERACIJA #1

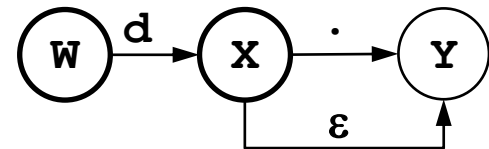
$$\varepsilon\text{-closure}(s) = \{s\} \cup \{T\}$$

gdje je **s** spojen s **T** pomoću ε -prijelaza

- ova operacija daje **skup NKA stanja**, u koja se može doći iz stanja **s** koristeći **ε -prijelaze**
- ovu operaciju prevodimo kao **epsilon-obuhvat**
- primjerice :

$$\varepsilon\text{-closure}(X) = \{X, Y\}$$

$$\varepsilon\text{-closure}(W) = \{W\}$$



OPERACIJE PRETVOBE NKA U DKA – OPERACIJA #2

$$\varepsilon\text{-closure}(T) = \bigcup \varepsilon\text{-closure}(s)$$

za sva stanja $s \in T$

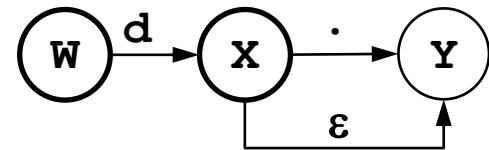
- operacija koja **daje skup NKA stanja**, u koja se može doći iz svih stanja $s \in T$ koristeći **ε -prijelaze**
- **$\varepsilon\text{-closure}(T)$** uključuje i **$T$**

- primjerice :

$$\varepsilon\text{-closure}(X) = \{X, Y\}$$

$$\varepsilon\text{-closure}(W) = \{W\}$$

$$\varepsilon\text{-closure}(\{W, X\}) = \{W, X, Y\}$$



OPERACIJE PRETVOBE NKA U DKA – OPERACIJA #3

move (T , a)

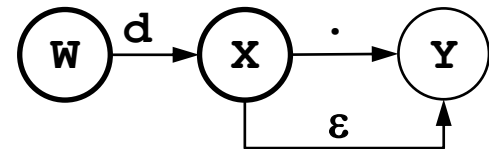
- operacija koja **daje skup NKA stanja**, u koja se može doći ako:
 - automat se nalazi u jednom od stanja $s \in T$
 - trenutni ulazni **simbol** je a

- primjerice :

move ($\{W\}$, d) = $\{X, Y\}$

move ($\{W\}$, $.$) = $\{\}$

move ($\{W, X\}$, $.$) = $\{Y\}$



ALGORITAM PRETVOBE NKA U DKA

algoritam kao **ulaz** prima **NKA**

NKA je **definiran** skupom stanja i prijelaza

s_0 je **početno stanje** NKA

algoritam kao **izlaz** daje **DKA**

izlazni **DKA** ima **skup stanja i prijelaza**

DKA je **definiran** tablicom prijelaza

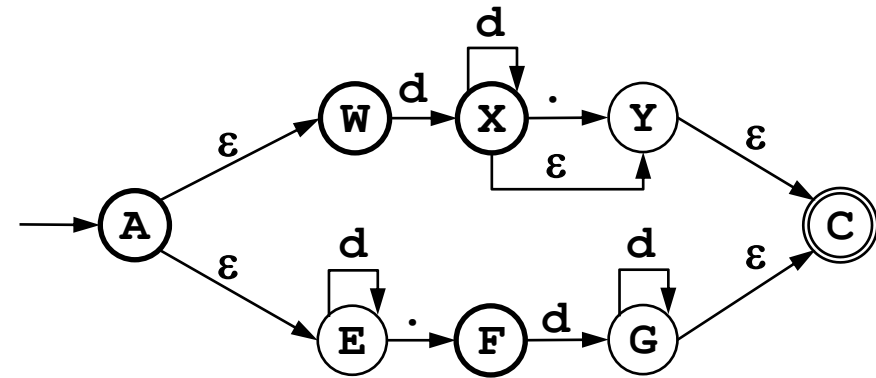
DVIJE KARAKTERISTIKE STANJA DKA

stanja DKA tijekom pretvorbe
imaju **dvije karakteristike:**
kompletnost i konačnost

svako stanje DKA može biti
kompletno ili nekompletno

svako stanje DKA može biti
obično ili konačno

ALGORITAM PRETVOBE NKA U DKA



izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu
za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

$D\text{Tablica}[T, a] = U$

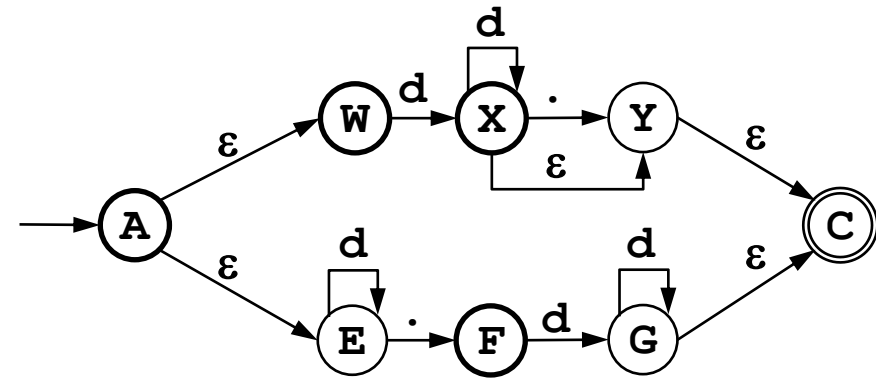
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

DTablica	d	.

ALGORITAM PRETVOBE NKA U DKA



izračunaj ϵ -closure(s_0)

označi ϵ -closure(s_0) kao nekompletno stanje

ubaci ϵ -closure(s_0) kao novi red u DTablicu
za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \epsilon$ -closure(move(T, a))

→ **A, W, E**

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

DTablica[T, a] = U

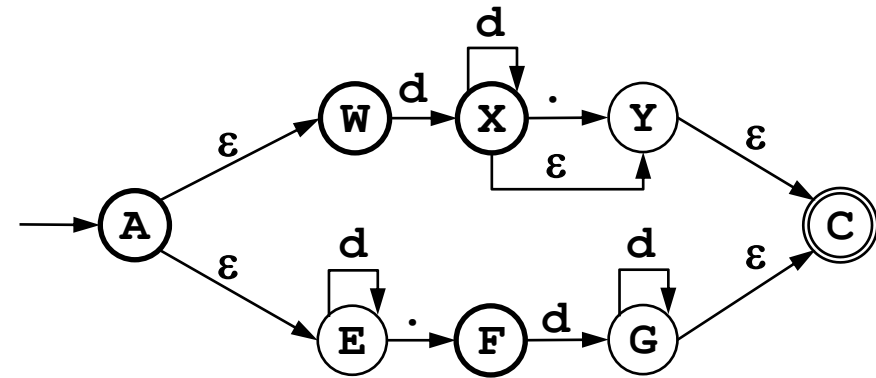
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

DTablica	d	.

ALGORITAM PRETVOBE NKA U DKA



izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

→ **A, W, E**

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

DTablica[T, a] = U

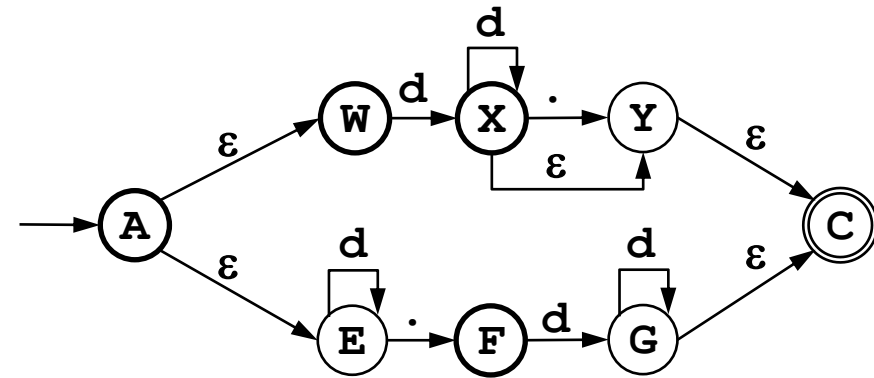
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

DTablica	d	.

ALGORITAM PRETVOBE NKA U DKA



izračunaj ϵ -closure(s_0)

označi ϵ -closure(s_0) kao nekompletno stanje

ubaci ϵ -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \epsilon$ -closure(move(T, a))

→ **A, W, E**

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

DTablica[T, a] = U

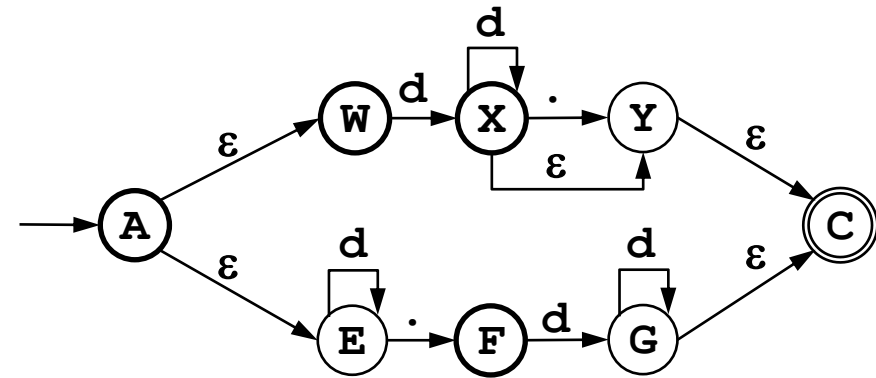
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

DTablica	d	.
A, W, E		

ALGORITAM PRETVOBE NKA U DKA



izračunaj ϵ -closure(s_0)

označi ϵ -closure(s_0) kao nekompletno stanje

ubaci ϵ -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

U = ϵ -closure(move(T, a))

→ **A, W, E**

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

DTablica[T, a] = U

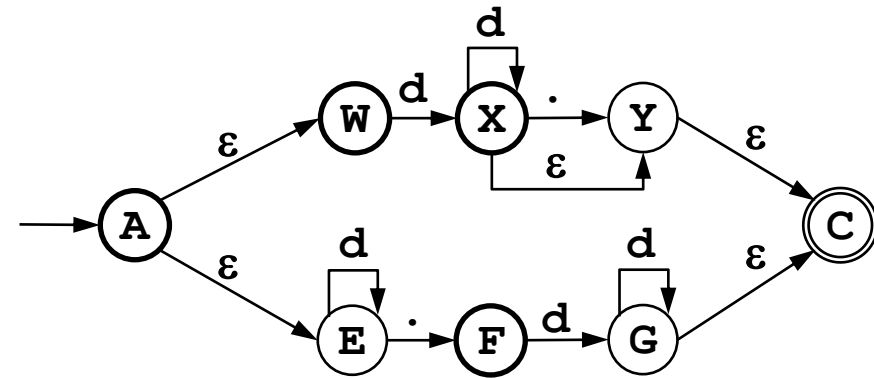
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

DTablica	d	.
A, W, E		

ALGORITAM PRETVOBE NKA U DKA



izračunaj ϵ -closure(s_0)

označi ϵ -closure(s_0) kao nekompletno stanje

ubaci ϵ -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \epsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

DTablica[T, a] = U

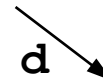
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje



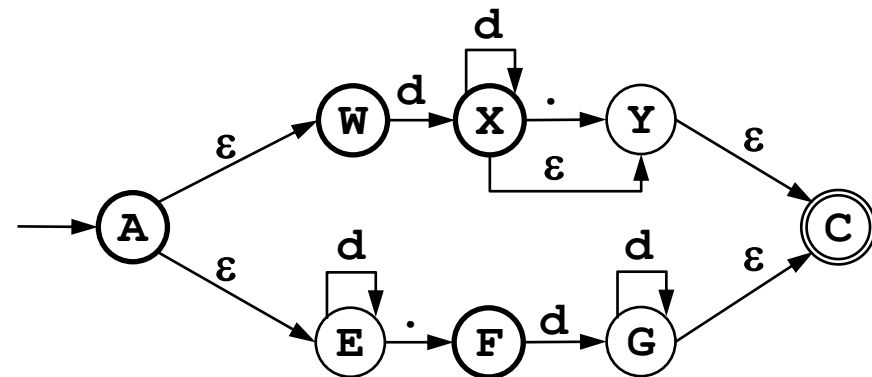
A, W, E



X, E, Y, C

DTablica	d	.
A, W, E		
X, E, Y, C		

ALGORITAM PRETVOBE NKA U DKA



izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

$D\text{Tablica}[T, a] = U$

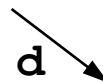
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje



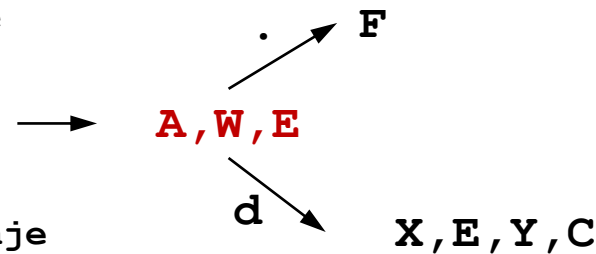
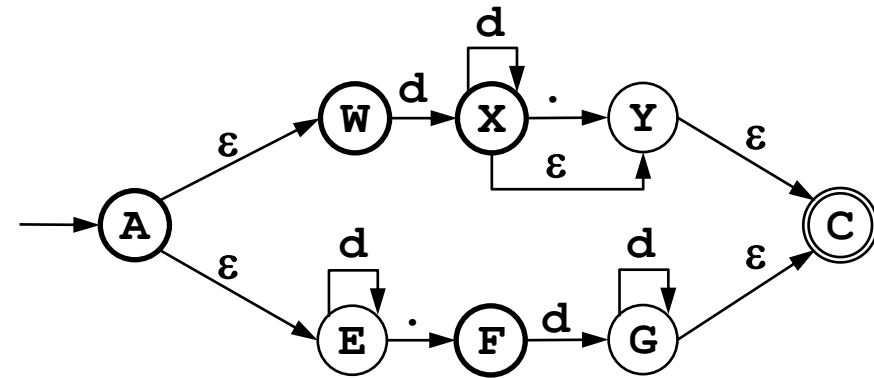
A, W, E



X, E, Y, C

DTablica	d	.
A, W, E	X, E, Y, C	
X, E, Y, C		

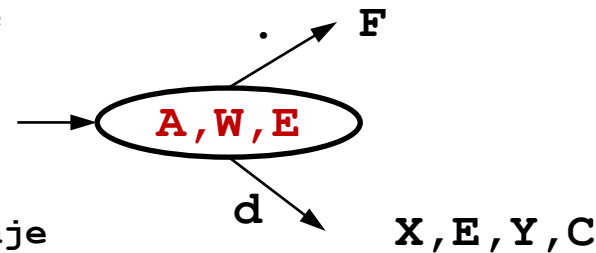
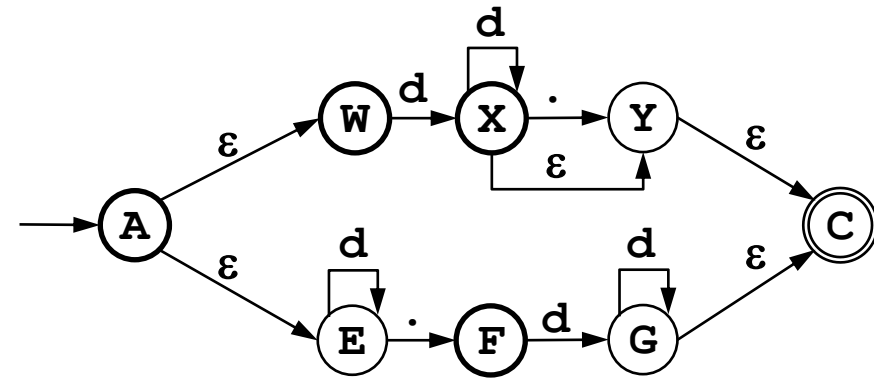
ALGORITAM PRETVOBE NKA U DKA



izračunaj ε -closure(s_0)
 označi ε -closure(s_0) kao nekompletno stanje
 ubaci ε -closure(s_0) kao novi red u DTablicu
 za svako nekompletno stanje T iz DTablice
 za svaki ulazni simbol a
 $U = \varepsilon$ -closure(move(T, a))
 ako U nije element DTablica tada
 označi U kao nekompletno stanje
 dodaj U kao novi red u DTablicu
 DTablica[T, a] = U
 označi T kao kompletno stanje
 ako T sadrži konačno stanje
 označi T kao konačno stanje

DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C		
F		

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C		
F		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

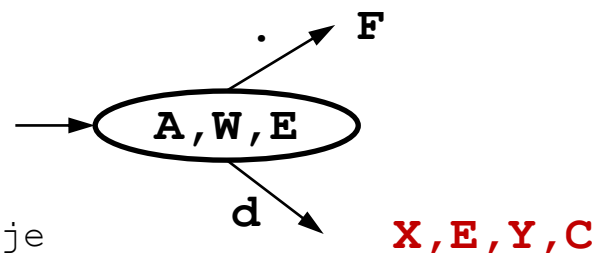
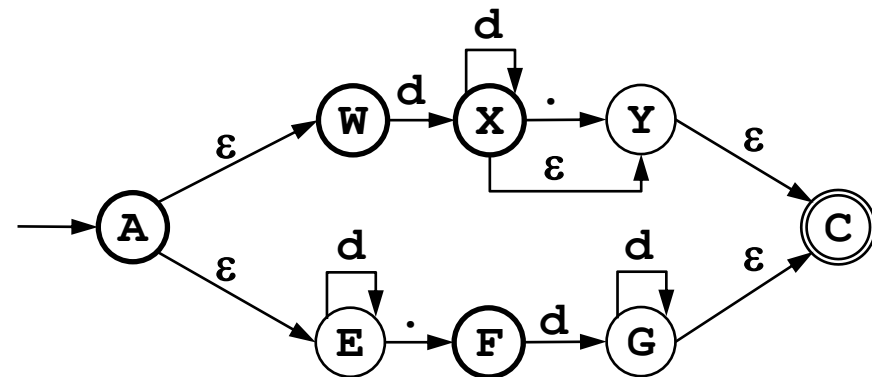
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C		
F		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

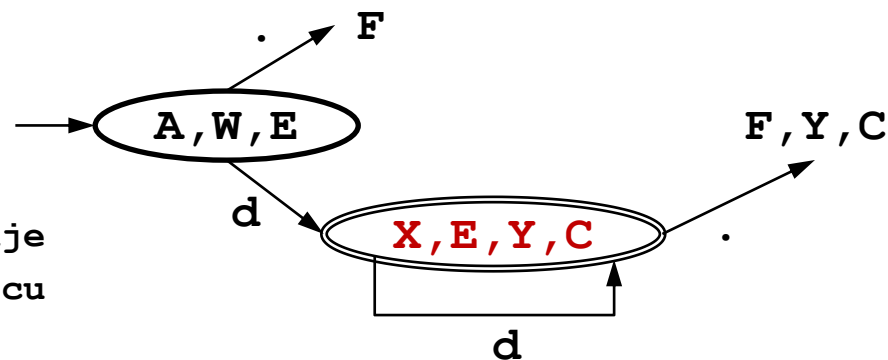
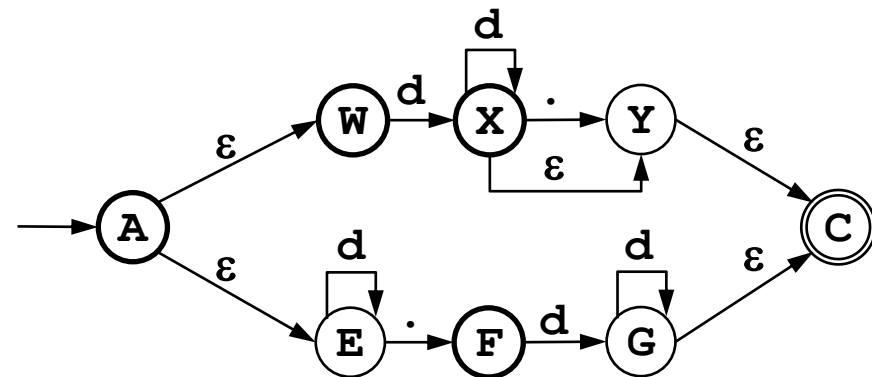
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F		
F, Y, C		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

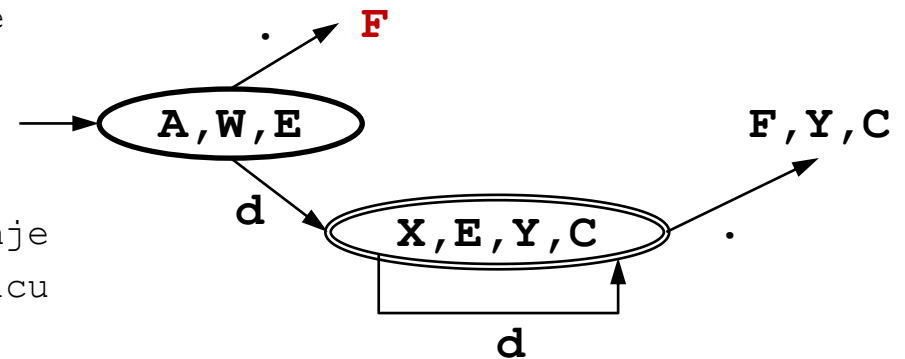
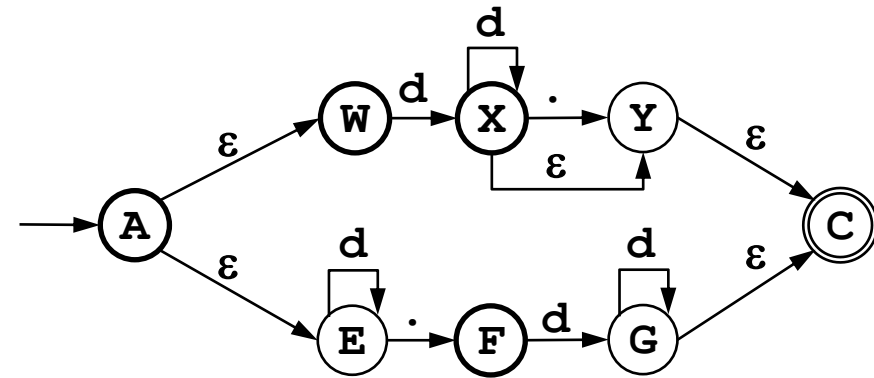
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

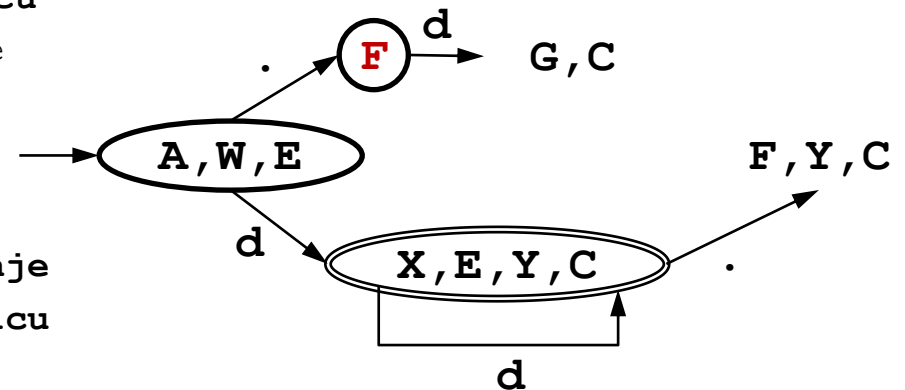
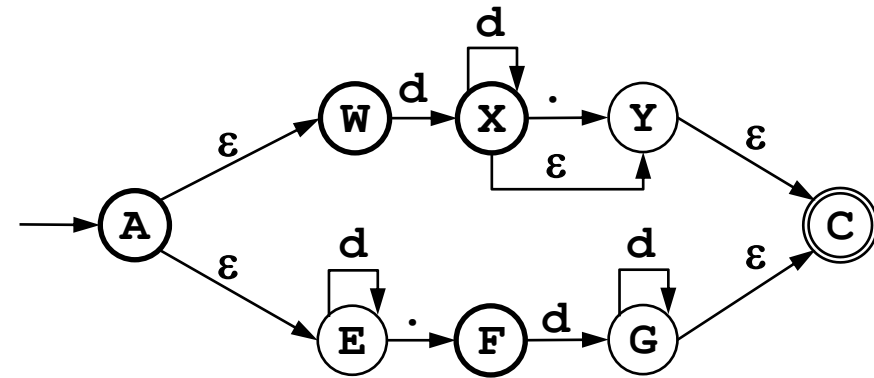
ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F		
F, Y, C		

izračunaj ε -closure(s_0)
 označi ε -closure(s_0) kao nekompletno stanje
 ubaci ε -closure(s_0) kao novi red u DTablicu
 za svako nekompletno stanje T iz DTablice
 za svaki ulazni simbol a
 U = ε -closure(move(T, a))
 ako U nije element DTablica tada
 označi U kao nekompletno stanje
 dodaj U kao novi red u DTablicu
 DTablica[T, a] = U
 označi T kao kompletno stanje
 ako T sadrži konačno stanje
 označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F	G, C	
F, Y, C		
G, C		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

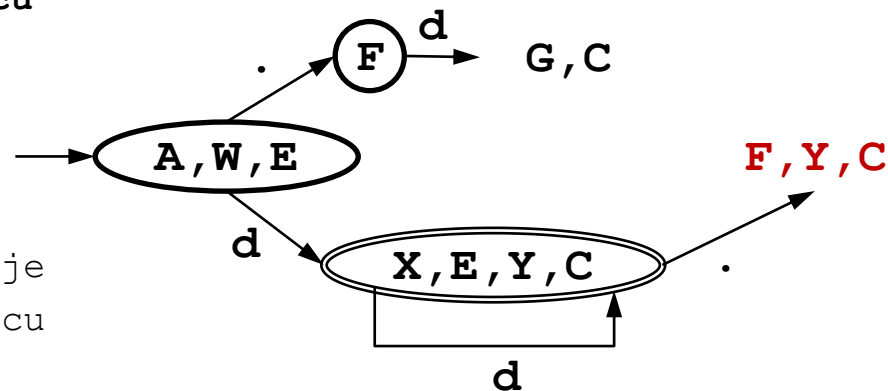
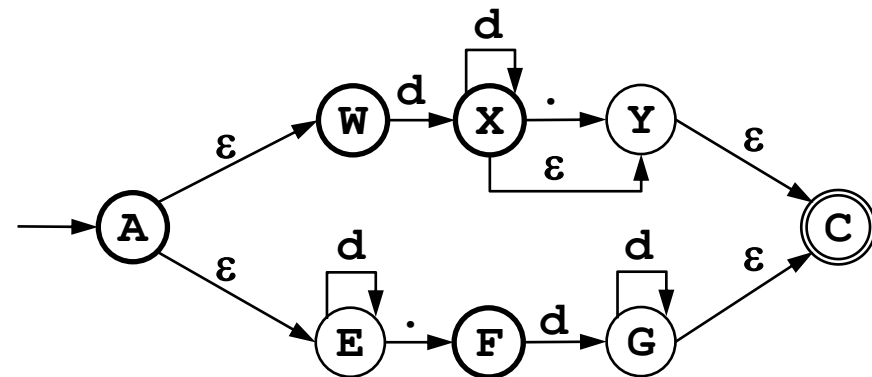
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	·
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F	G, C	
F, Y, C		
G, C		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

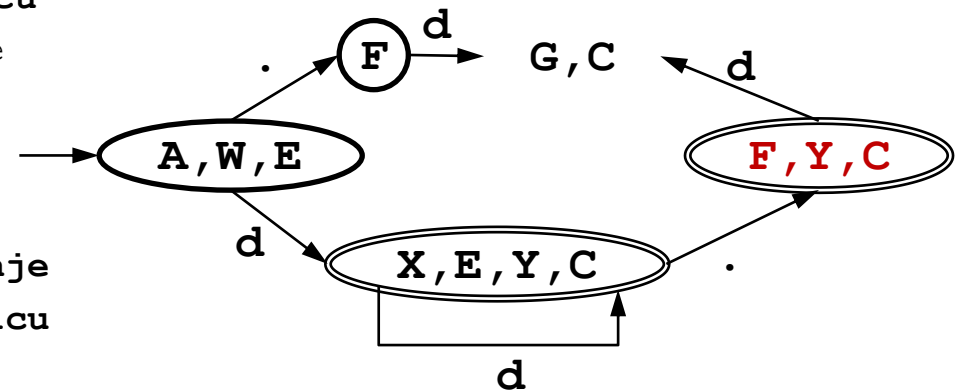
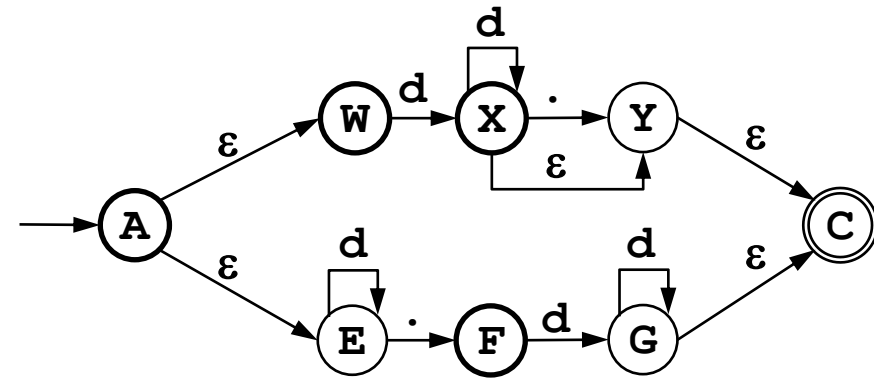
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

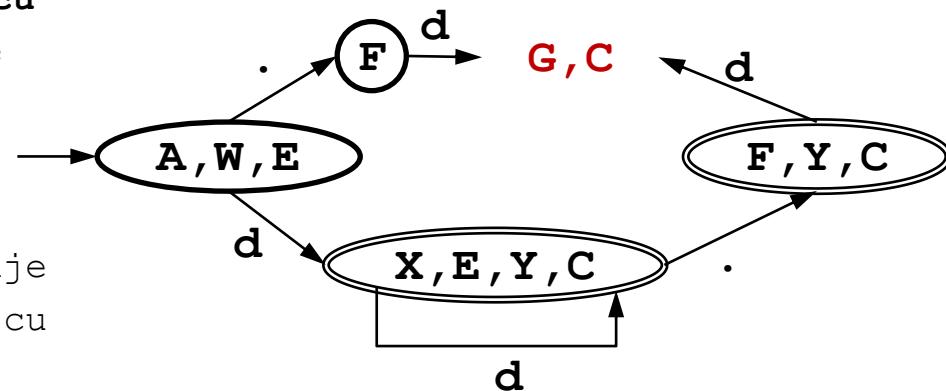
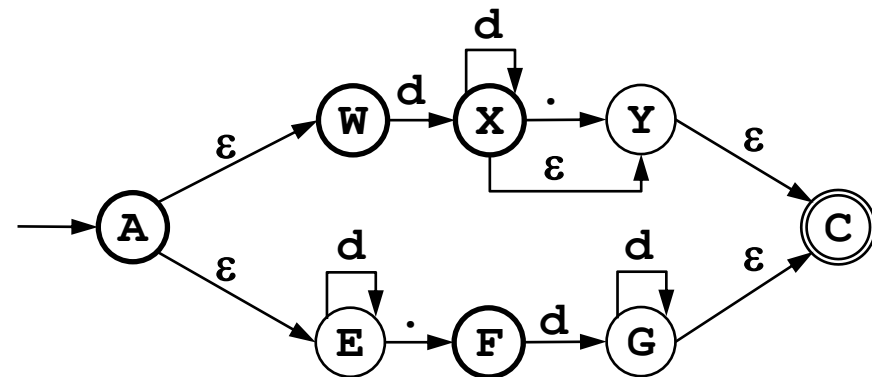
ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F	G, C	
F, Y, C	G, C	
G, C		

izračunaj ε -closure(s_0)
 označi ε -closure(s_0) kao nekompletno stanje
 ubaci ε -closure(s_0) kao novi red u DTablicu
 za svako nekompletno stanje T iz DTablice
 za svaki ulazni simbol a
 $U = \varepsilon$ -closure(move(T, a))
 ako U nije element DTablica tada
 označi U kao nekompletno stanje
 dodaj U kao novi red u DTablicu
 DTablica[T, a] = U
 označi T kao kompletno stanje
 ako T sadrži konačno stanje
 označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F	G, C	
F, Y, C	G, C	
G, C		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

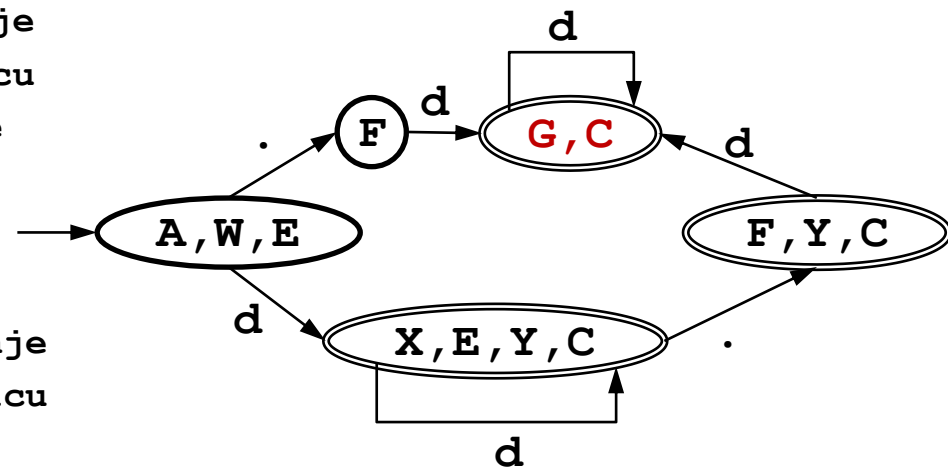
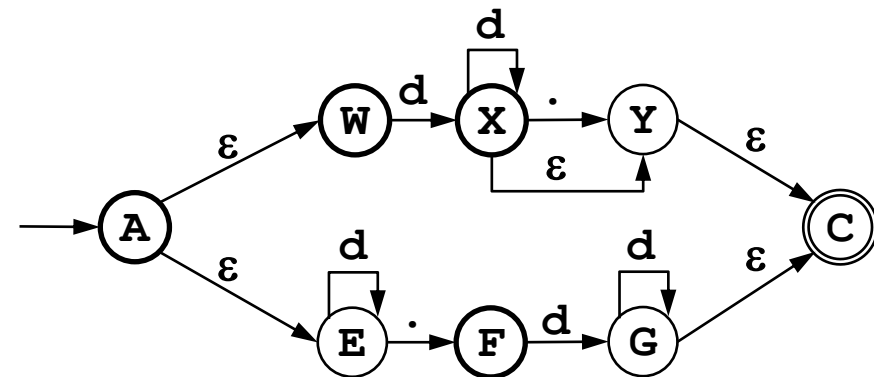
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.
A, W, E	X, E, Y, C	F
X, E, Y, C	X, E, Y, C	F, Y, C
F	G, C	
F, Y, C	G, C	
G, C	G, C	

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

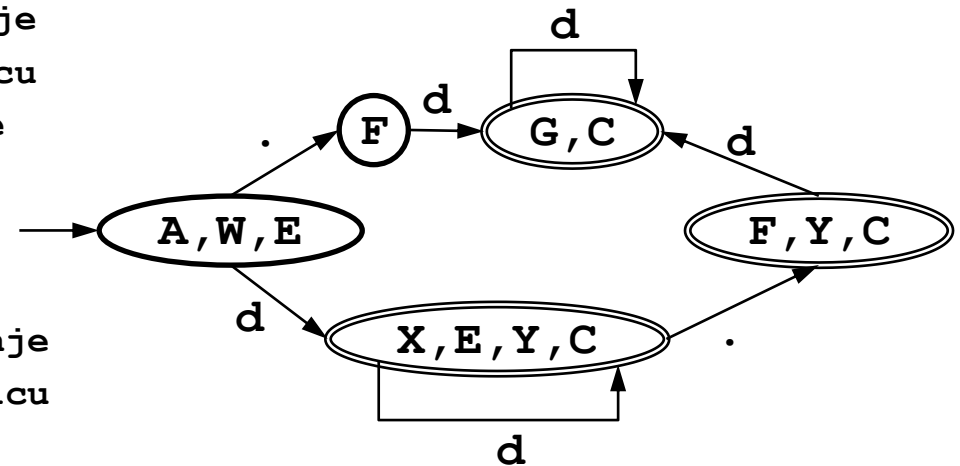
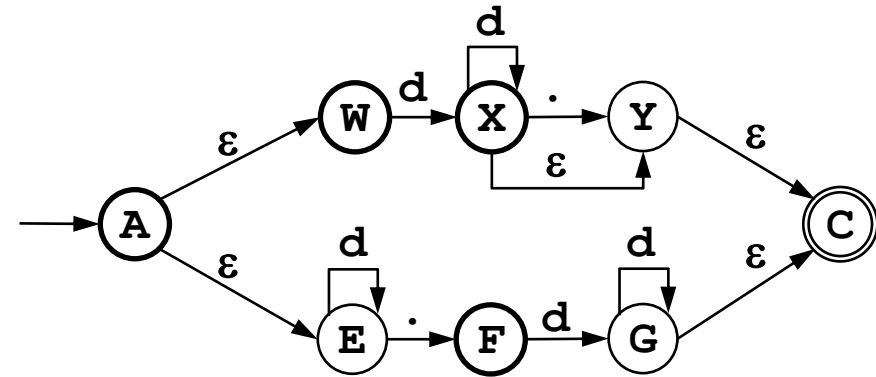
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.	konačno
A, W, E	X, E, Y, C	F	
X, E, Y, C	X, E, Y, C	F, Y, C	
F	G, C		
F, Y, C	G, C		
G, C	G, C		

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

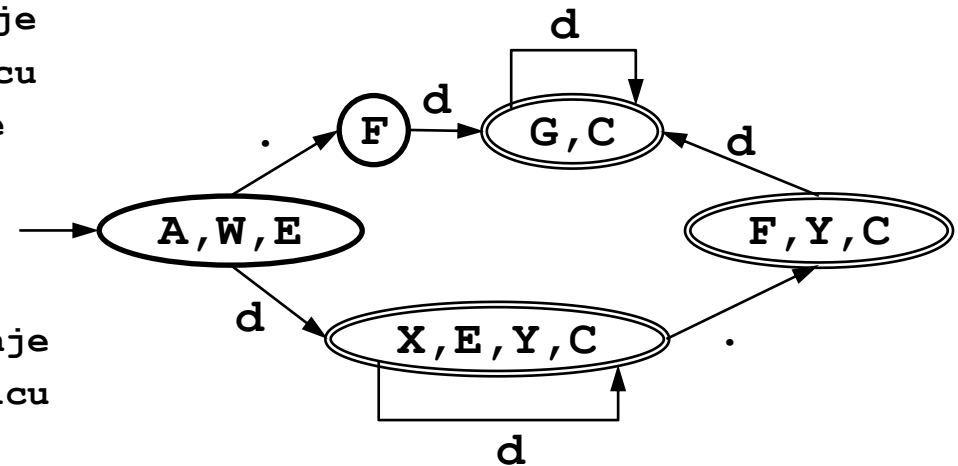
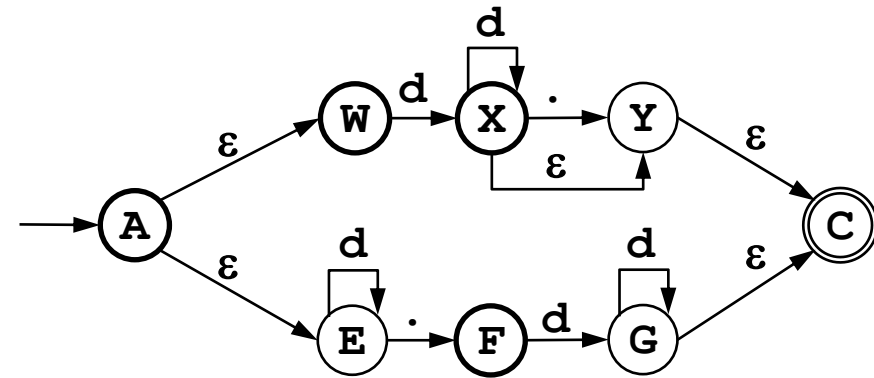
DTablica[T, a] = U

označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA



DTablica	d	.	konačno
A, W, E	X, E, Y, C	F	ne
X, E, Y, C	X, E, Y, C	F, Y, C	da
F	G, C	greška	ne
F, Y, C	G, C	greška	da
G, C	G, C	greška	da

izračunaj ε -closure(s_0)

označi ε -closure(s_0) kao nekompletno stanje

ubaci ε -closure(s_0) kao novi red u DTablicu

za svako nekompletno stanje T iz DTablice

za svaki ulazni simbol a

$U = \varepsilon$ -closure(move(T, a))

ako U nije element DTablica tada

označi U kao nekompletno stanje

dodaj U kao novi red u DTablicu

DTablica[T, a] = U

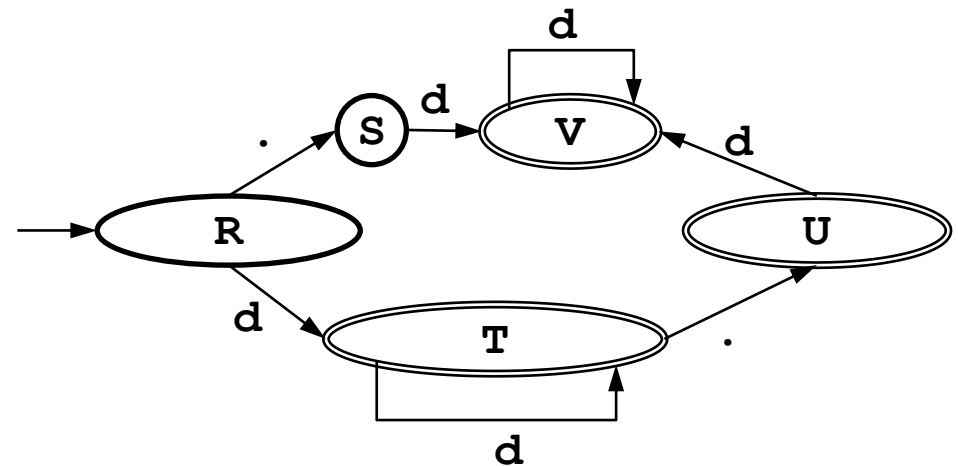
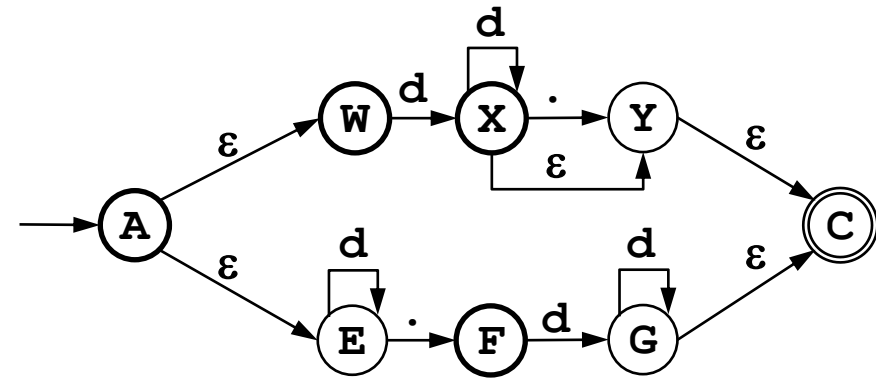
označi T kao kompletno stanje

ako T sadrži konačno stanje

označi T kao konačno stanje

ALGORITAM PRETVOBE NKA U DKA

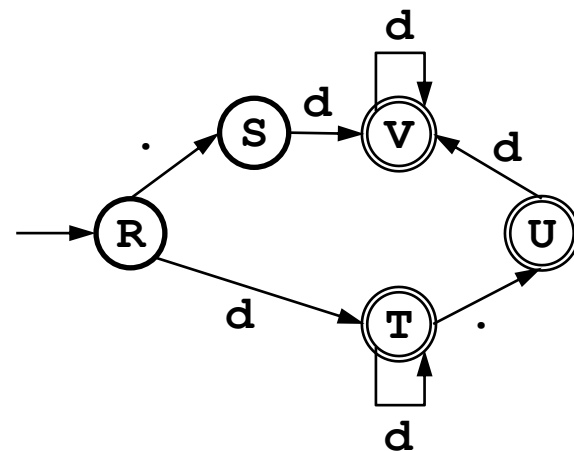
- na kraju **zamijenimo** skupove stanja NKA
- staviti ćemo nova imena radi **čitкости**



DTablica	d	.	konačno
R	T	S	ne
T	T	U	da
S	V	greška	ne
U	V	greška	da
V	V	greška	da

OPTIMIZACIJA DKA

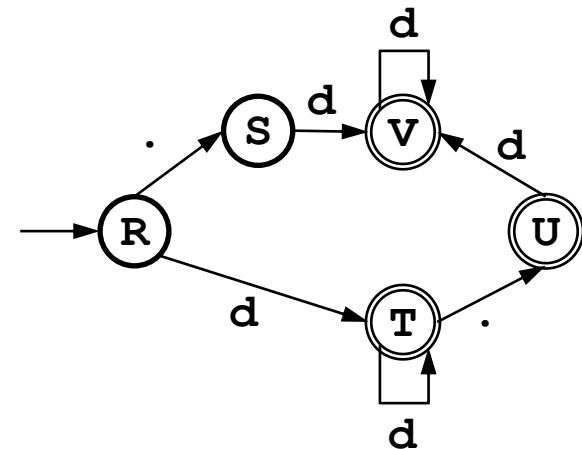
- **NKA** koji ima **n stanja** može rezultirati sa **DKA** koji ima do **$2^n - 1$ stanja**
 - poželjno je izvršiti dodatno **optimiranje** DKA
 - cilj je postići automat sa **manje stanja**
-
- ovdje ćemo prikazati **Mooreov** odnosno **Huffman-Mealy** algoritam



FORMIRANJE SKUPOVA

- optimizacija se vrši u **nadi** 😊 da ćemo moći:
 - **sva konačna stanja** spojiti u jedno stanje
 - **sva ostala stanja** također spojiti u jedno stanje
- najprije se formiraju **dva skupa stanja**
- prvi skup neka sadrži sva **konačna stanja**
- drugi skup neka sadrži sva **ne-konačna stanja**
- za naš DKA dobivamo dva skupa:

[R, S] [T, U, V]



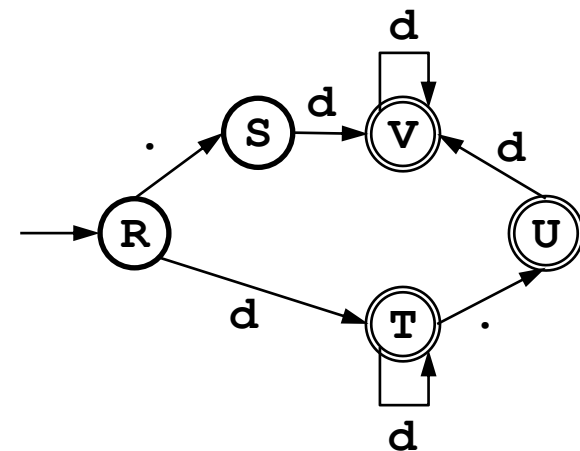
TEST KONZISTENTNOSTI

- potom se za svaki skup provjerava **konzistentnost**
- skup je **konzistentan** ako sva njegova stanja za **isti simbol** prelaze u **isto stanje**

DTablica	d	.	konačno
R	T	S	ne
S	V	greška	ne
T	T	U	da
U	V	greška	da
V	V	greška	da

- najprije vršimo provjeru za skup **[R, S]**

[R, S] [T, U, V]



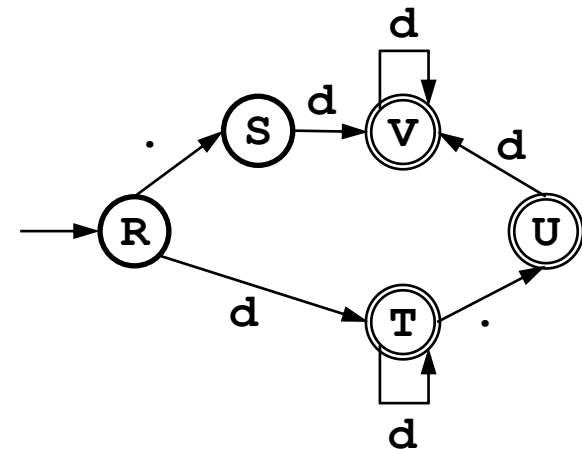
TEST KONZISTENTNOSTI

- zaključujemo da skup **nije konzistentan**
- tada se skup **[R, S]** dijeli na **[R]** i **[S]**

DTablica	d	.	konačno
R	T	S	ne
S	V	greška	ne
T	T	U	da
U	V	greška	da
V	V	greška	da

- **[R]** i **[S]** više ne moramo provjeravati
- razlog je što je u skupu **samo jedno stanje**

[R] [S] [T, U, V]



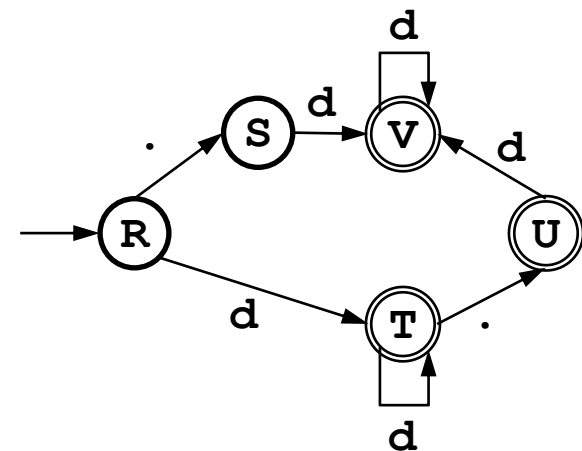
TEST KONZISTENTNOSTI

- provjera se vrši sve dok imamo **barem jedan** skup sa **više od jednog** nekonzistentnog stanja
- za prva dva više ne treba provjera

DTablica	d	.	konačno
R	T	S	ne
S	V	greška	ne
T	T	U	da
U	V	greška	da
V	V	greška	da

- provjeravamo treći skup **[T, U, V]**
- nije konzistentan, dijelimo ga

[R] [S] [T, U, V]



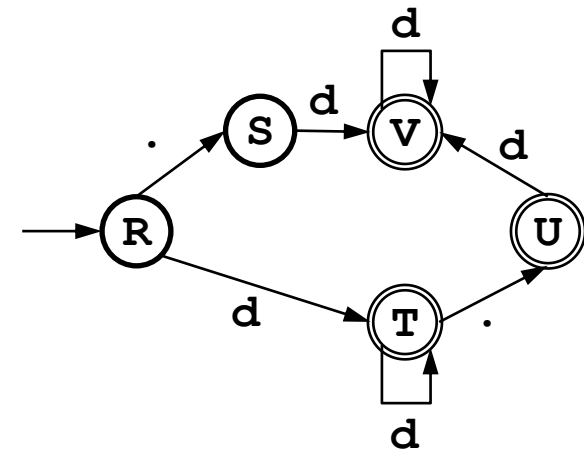
TEST KONZISTENTNOSTI

- provjera konzistentnosti se vrši za jedini preostali skup koji sadrži sa više stanja
- provjeravamo skup $[U, V]$

DTablica	d	.	konačno
R	T	S	ne
S	U	greška	ne
T	T	U	da
U	U	greška	da

$[R] [S] [T] [U, V]$

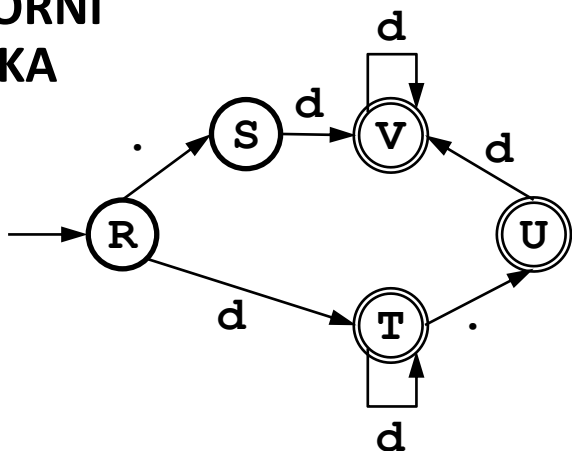
- skup $[U, V]$ je **konzistentan**
- stanja možemo **spojiti** u jedno
- time je proces **završen**



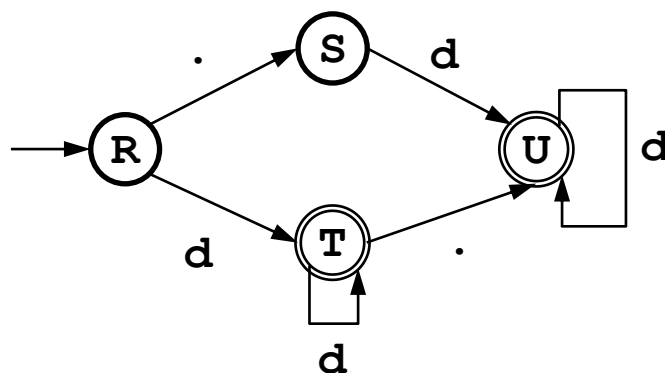
OPTIMIZIRANI DKA

- na kraju optimizacije naši skupovi su: **[R] [S] [T] [U,V]**
- znači možemo ujediniti stanje **U** i **V**

IZVORNI
DKA



OPTIMIZIRANI
DKA



DTablica	d	.	konačno
R	T	S	ne
T	T	U	da
S	U	greška	ne
U	U	greška	da