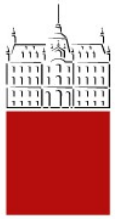
Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Nederminizem, iskanje vzorcev in kontekstno neodvisni jeziki

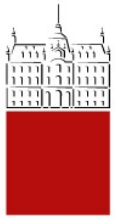
15.marec 2014

Uroš Čibej

Pregled dneva

- Sklop 1: Nedeterministični končni avtomati (8:30 – 10:00)
- Sklop 2: Ekvivalentnost končnih avtomatov (10:15 – 12:00)
- Sklop 3: (Ne)moč končnih avtomatov in vpeljava gramatik (12:15 - 13:30)
- Izdelava priprav (13:30 – 14:15)

Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Nedeterminizem

15.marec 2014

Zgodovina

Michael Rabin

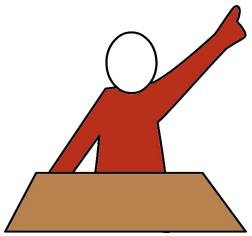
Dana Scott

(1959)

Kaj je nedeterminizem?

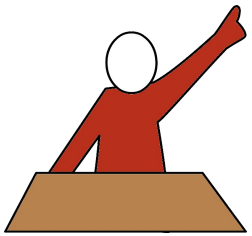
- Nedeterminizem vpeljemo kot razširitev končnih avtomatov
- Kam točno skočimo je nedoločeno (**nedeterminirano**)
- Osnovnim avtomatom zato pravimo deterministični (DKA), razširitvi pa nedeterministični končni avtomati (NKA)
- Dovolimo lahko več prehodov iz istega stanja preko enakega simbola v različna stanja

Zakaj?

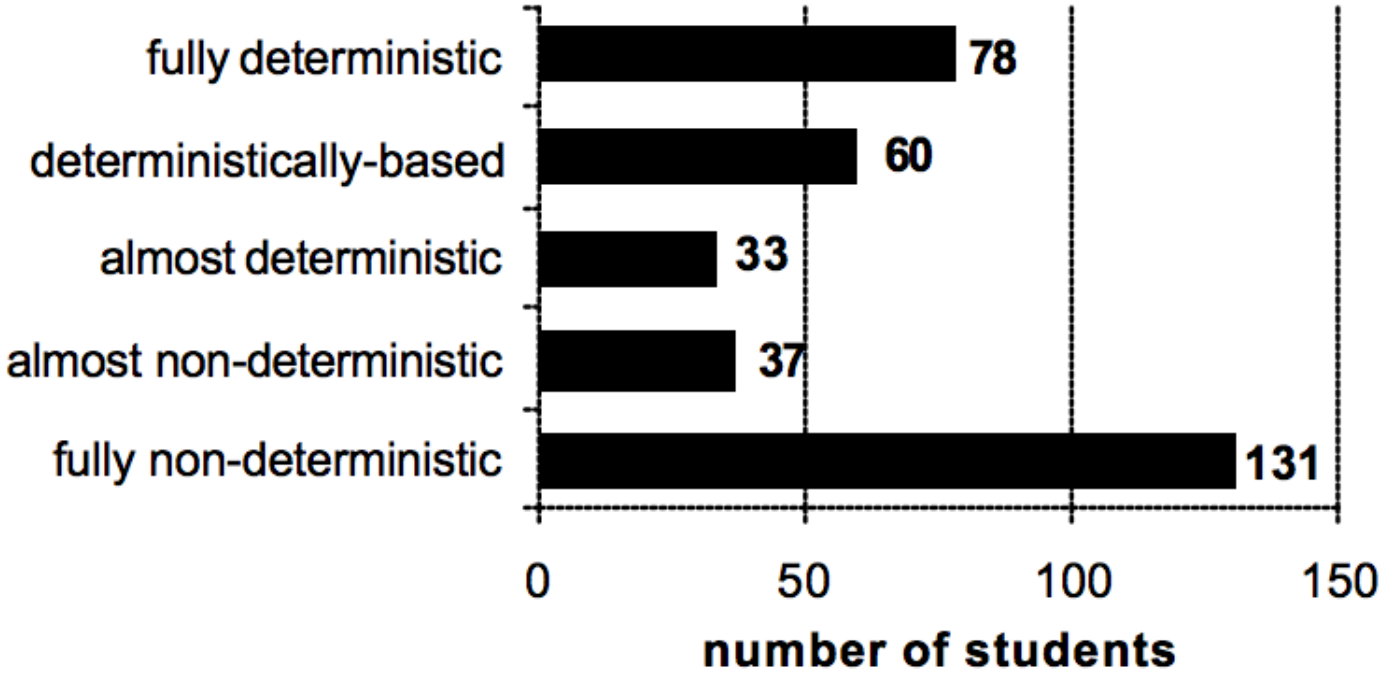


- Nedeterminizem ima izjemno pomembno vlogo v računalništvu
- Teoretična „igra“ - iskanje možnih razširitev nekega modela računanja
- Prvi prikaz dokazovanja enakosti dveh modelov računanja
- Za podajanje nekaterih jezikov je nedeterminizem bolj intuitiven kot determinizem

Intuitiven?

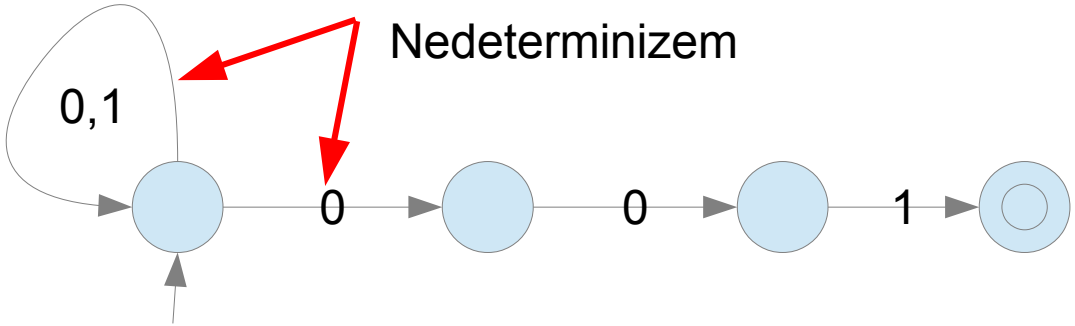


M. Armoni, J. Gal-Ezer: *Non-determinism in CS high-school curricula*, ASEE/IEEE Frontiers in Education Conference, 2003



Primer 1

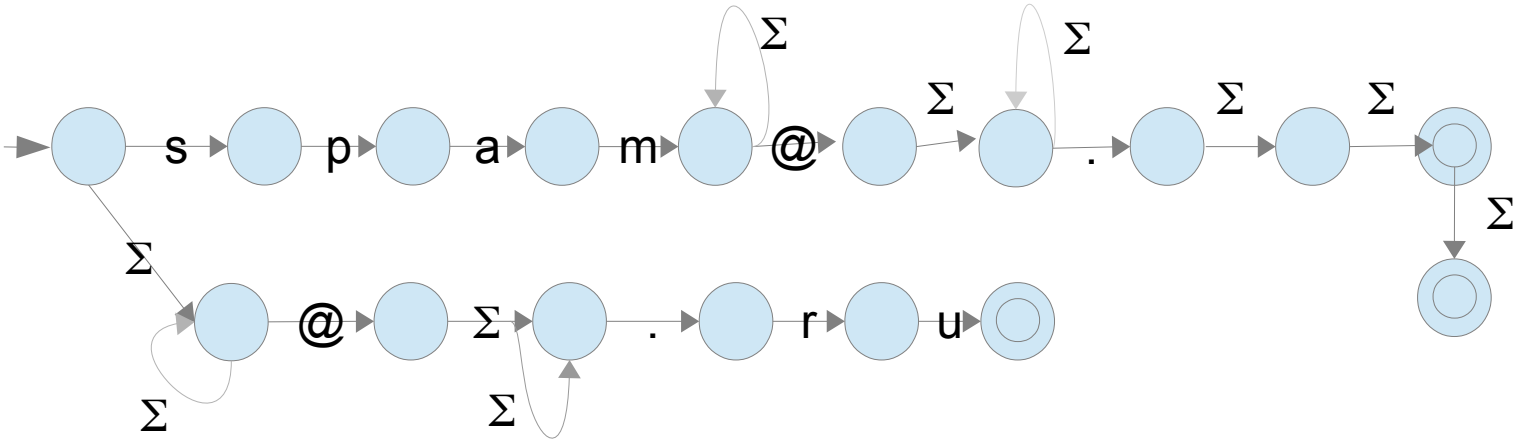
Zapišimo končni avtomat, ki sprejema vse nize, ki se končajo na 001



Primer 2

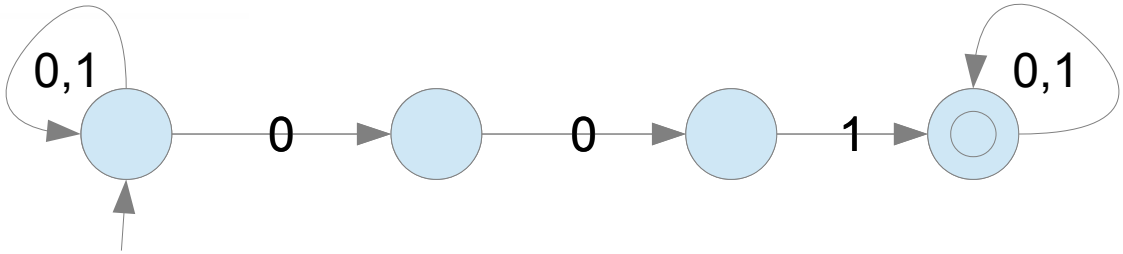
Detektirajmo elektronske naslove, ki so sumljivi in lahko predstavljajo neželjeno pošto. Dve zelo enostavni pravili:

1. Elektronski naslov se začne na **spam**, ali
2. Elektronski naslov se konča na **.ru**

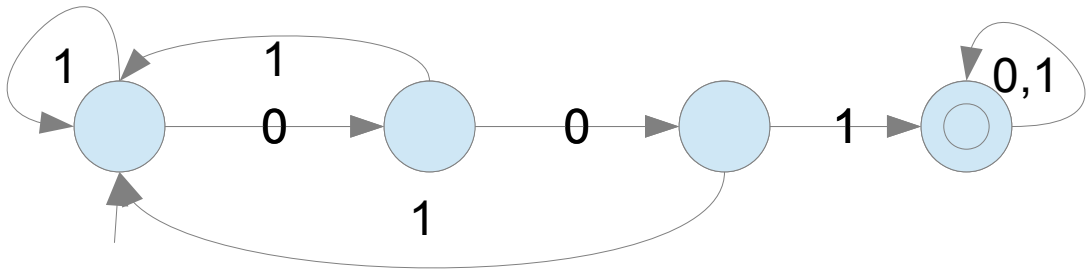


Primer 3

Zapišimo NKA, ki sprejema vse nize, ki **vsebujejo** 001



Deterministični?

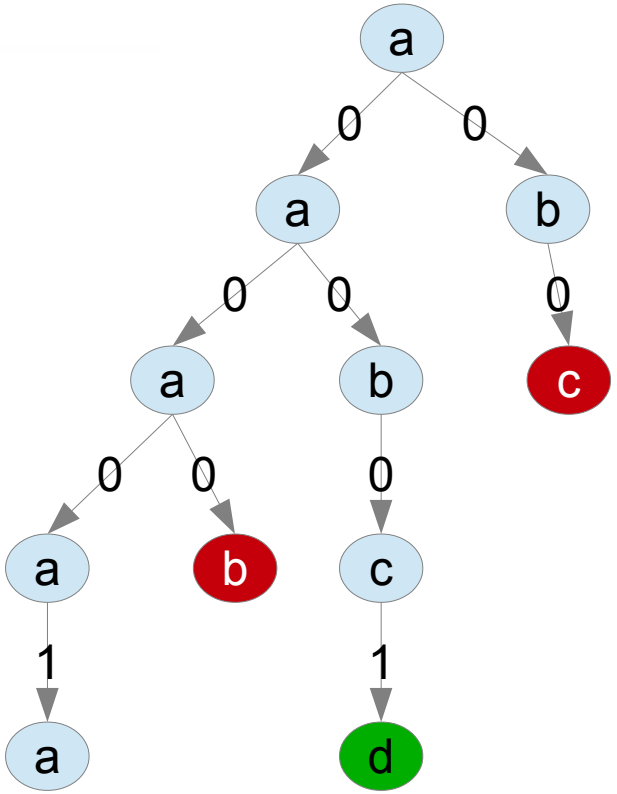


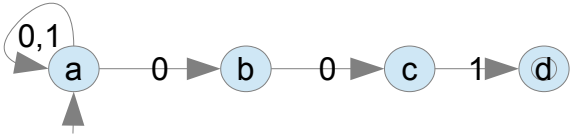
Primerjava z DKA

- Iz enega stanja lahko preko istega simbola pridemo v več različnih stanj – pri DKA samo eden
- Prehod preko nekega simbola je lahko tudi nedefiniran – ne potrebujemo slepega stanja kot pri DKA
- Kako besede sprejmemo?

Sprejemanje besede

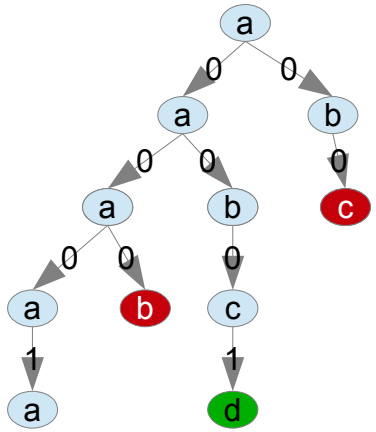
Beseda **0001**





Intuitivna interpretacija nedeterminizma I.

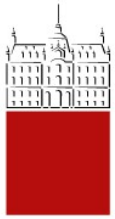
- Vzporedno izvajanje:
  - Vsakič, ko imamo več možnosti, hkrati preizkusimo vse
  - Vsako možnost prevzame en nov računalnik (procesor, jedro, ...)



Intuitivna interpretacija nedeterminizma II.

- Ugibanje:
  - Vsakič, ko imamo več možnost avtomat »ugane« pravo možnost
  - Če je beseda v jeziku, potem takšno zaporedje »ugibanj« obstaja – takemu zaporedju pravimo **certifikat**.
  - Če pa besede ni v jeziku, potem nobeno ugibanje ne pripelje do končnega stanja

Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko



Formalna definicija

15.marec 2014

Formalna definicija NKA

$$M = (Q, \Sigma, \delta, q_{0,} F)$$

Q	Množica stanj
Σ	Abeceda
$\delta : Q \times \Sigma \rightarrow 2^Q$	Funkcija prehodov, ki slika v poljubno podmnožico Q
$q_0 \in Q$	Začetno stanje
$F \subseteq Q$	Množica končnih stanj

*Jezik nedeterminističnega avtomata I.

Razširimo funkcijo prehodov, da slika neko stanje in **besedo w** v novo stanje

$$\hat{\delta}(q, w): Q \times \Sigma^* \rightarrow 2^Q$$

Najprej preslikamo začetek besede **v** (dobimo množico možnih stanj), nato pa iz vsakega stanja poskusimo priti še po simbolu **a**

$$\hat{\delta}(q, va) = \bigcup_{p \in \hat{\delta}(q, w)} \delta(p, a)$$

*Jezik nedeterminističnega avtomata II.

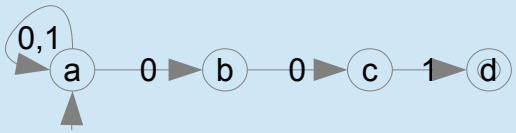
Jezik nedeterminističnega avtomata so vse besede **w**, za katere velja:

$$\hat{\delta}(q, w) \subseteq F \neq \emptyset$$

Neformalno: vse besede preko katerih lahko pridem vsaj do enega končnega stanja.

Podajanje avtomatov

Diagram prehodov



Spisek prehodov

$$\delta(a,0)=\{a,b\}$$

$$\delta(a,1)=a$$

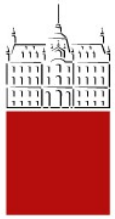
$$\delta(b,0)=c$$

$$\delta(a,0)=a$$

Tabelarično

	0	1
a	a,b	a
b	c	/
c	/	d
d	/	/

Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



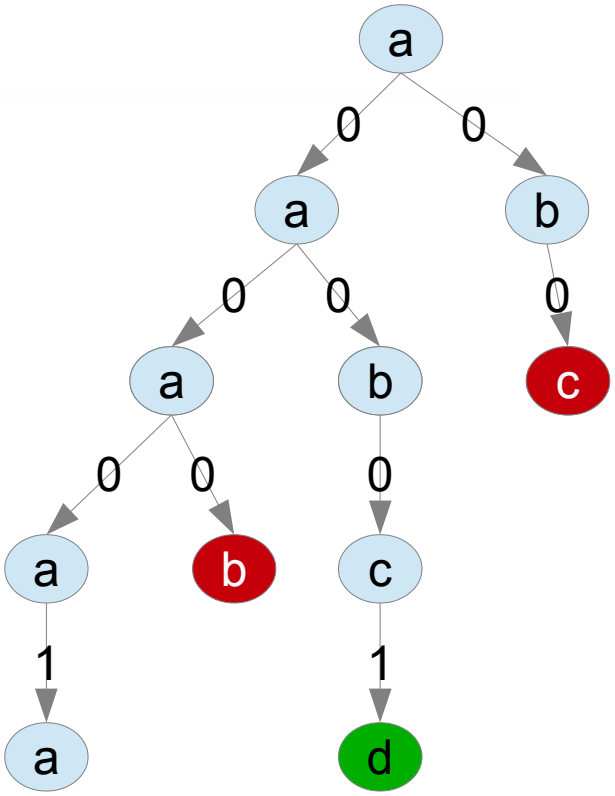
Simulacija

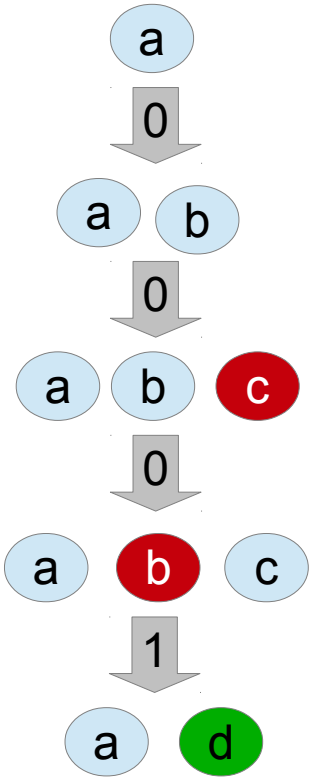
15.marec 2014

Simulacija NKA

- Nedeterminizem ni fizikalno možen model računanja - takega ugibanja ne znamo narediti, poljubnega paralelizma pa tudi ne moremo narediti
- Lahko pa ga (učinkovito?) simuliramo, kar bomo pokazali tako, da bomo tak simulator zapisali
- Razširili bomo program, ki ste ga napisali prejšnji teden

Ideja simulacije

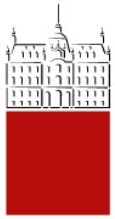




Nekatere `python` opombe

- Delali bomo z množicami, ki so dobro podprte v pythonu
- Ustvarjanje množice: `set([1,2,3])`
- Unija dveh množic:
  - `set([1,2,3]) | set(3,4,5)`
- Presek dveh množic:
  - `set([1,2,3]) & set(3,4,5)`

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*



Uporaba NKA

15.marec 2014

Hitreje, višje, močneje

- Računalniki niso neskončno hitri
- Naivni pristopi pogosto dajo slabe (počasne) rešitve
- Ena najbolj ključnih nalog računalništva je iskanje hitrejših načinov reševanja
- S pomočjo NKA bomo razvili zelo hitro rešitev za iskanje vzorcev v nizih

Iskanje vzorcev

- Poiskati želimo podniz v dolgem nizu
- Npr. v genetiki iščemo ali se podano zaporedje pojavi v človeškem genomu

Primer

CGTCGACCGATCAGGATGAAAATTAGGGGCCCGAGAT
GATCAGGA

Naivno iskanje

- Hkrati primerjamo en znak v vzorcu in en znak v ciljnem nizu:
  - če se ujemata se v obeh premaknemo za en znak naprej,
  - če se ne ujemata se v vzorcu in ciljnem nizu pomaknemo nazaj za isto število znakov.

Manjši primer

Primer

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB

AAAAAAAAAB
AAAAAAAAAB
AAAAAAAAAB
.
.
.

Velikost vzorca: 10
Velikost ciljnega niza: 31
Koliko primerjav?

Primerjav:

$\approx 30 * 10 = 300$

Velik realni primer

Primer

Človeški genom: 3.2 milijarde baznih parov
Vzorec: milijon baznih parov

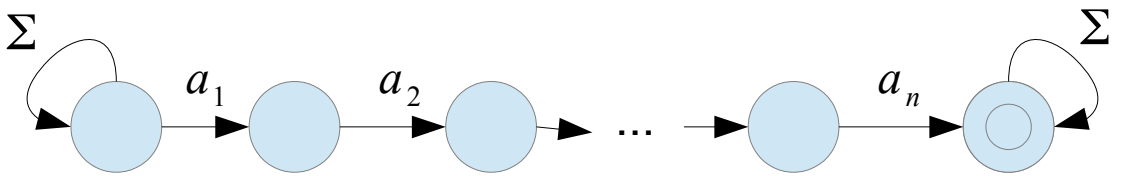
Št. primerjav:

$$\approx 10^6 * 3.2 * 10^9 = 3.2 * 10^{15}$$

Če ena primerjava traja 1ns, koliko časa bi trajalo tako iskanje?

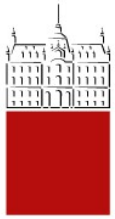
Vzorec zakodiran kot končni avtomat

$$w = a_1 a_2 a_3 \ldots a_n$$



Problem: simulacija NKA je zamudna, hitro razpoznavamo lahko zgolj z DKA

Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko



Pretvorba NKA v DKA

15.marec 2014

Ekvivalenca DKA in NKA

- Pokazali bomo, da sta ta dva modela računanja povsem enakovredna:
  - Najprej bomo pokazali, da lahko vsak DKA pretvorimo v NKA
  - Nato pa še, da lahko vsak NKA pretvorimo v DKA

Pretvorba DKA v NKA

- Vsak DKA lahko zelo enostavno pretvorimo v NKA tako:
  - Vsa stanja ostanejo enaka, abeceda ostane enaka, začetno in končna stanja ostanejo enaka, malenkostno se spremeni zgolj funkcija prehodov.

Pretvorba NKA v NKA

- Pri simulaciji smo videli, da je avtomat po prebranem znaku v hkrati v več stanjih.
- Vsa ta stanja bomo združili in jih obravnavali kot eno stanje.
- Možna stanja so torej vse možne podmnožice množice stanj NKA-ja.
- Za vsa ta stanja (podmnožico) bomo poiskali v katero pomnožico pridemo po posameznih simbolih.

Formalna pretvorba

NKA

$$M = (Q, \Sigma, \delta, q_{0,} F)$$

DKA

$$M = (2^Q, \Sigma, \delta', \{q_0\}, F')$$

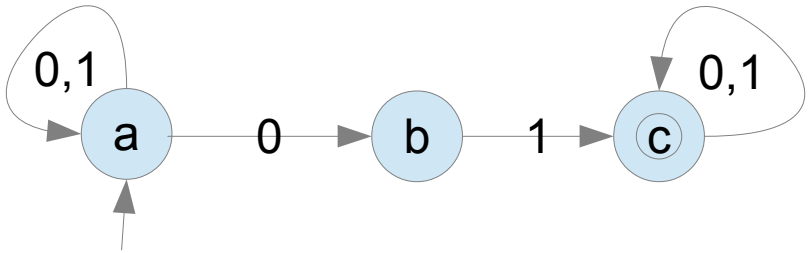
Za vsako podmnožico **P** izračunamo množico stanj v katero pridemo preko simbola **a**.

$$\delta'(P, a) = \bigcup_{q \in P} \delta(q, a)$$

Končna stanja so množice, ki vsebujejo vsaj eno stanje iz končne množice DKA.

$$F' = \{q \in 2^Q \mid q \cap F \neq \emptyset\}$$

Primer



Množica stanj: $\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}$

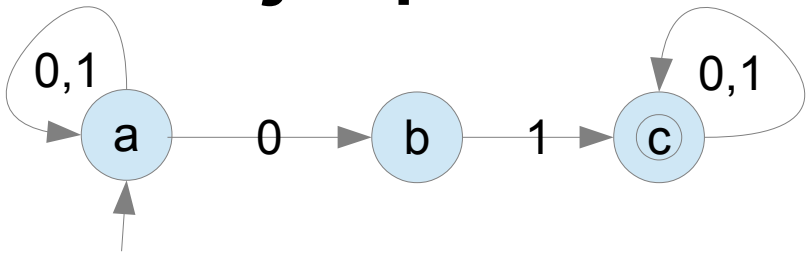
Končna stanja: $\{c\}, \{a,c\}, \{b,c\}, \{a,b,c\}$

Začetno stanje: $\{a\}$

Pretvorba s tabeliranjem podmnožic

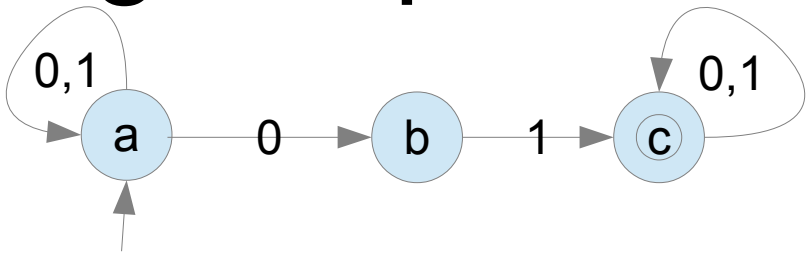
- Pri simulaciji smo videli, da je avtomat po prebranem znaku v hkrati v več stanjih.
- Vsa ta stanja bomo združili in jih obravnavali kot eno stanje.
- Možna stanja so torej vse možne podmnožice množice stanj NKA-ja.
- Za vsa ta stanja bomo tabelirali v katero pomnožico pridemo po posameznih simbolih.

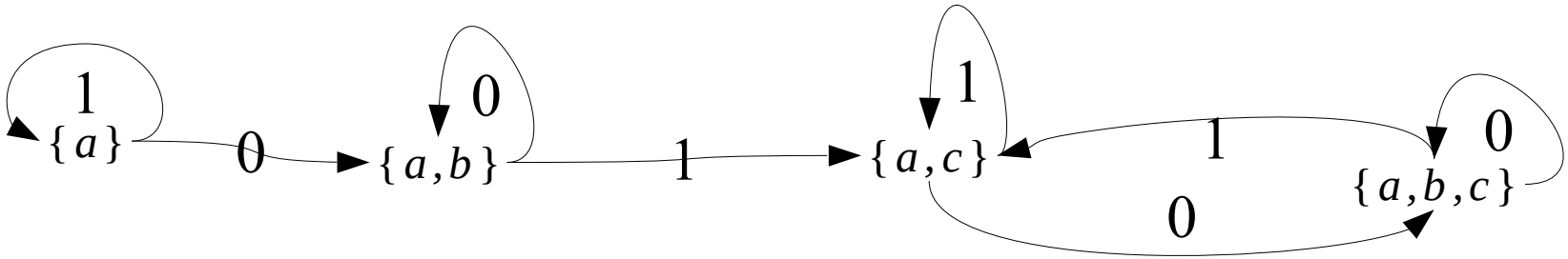
Funkcija prehodov

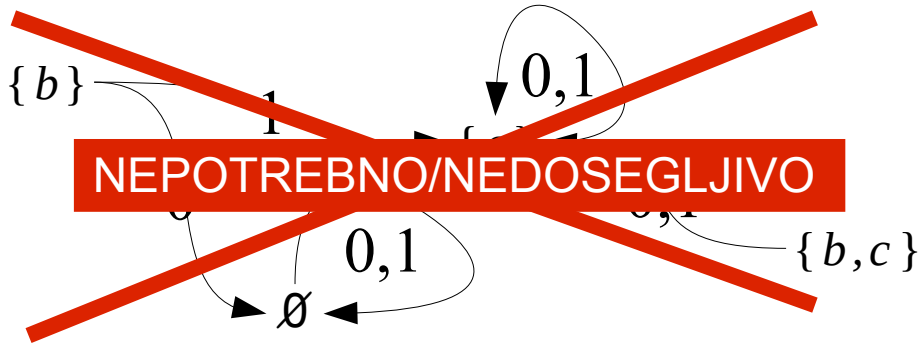


Podmožica	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\{a\}$	$\{a,b\}$	$\{a\}$
$\{b\}$	$\emptyset$	$\{c\}$
$\{c\}$	$\{c\}$	$\{c\}$
$\{a,b\}$	$\{a,b\}$	$\{a,c\}$
$\{a,c\}$	$\{a,b,c\}$	$\{a,c\}$
$\{b,c\}$	$\{c\}$	$\{c\}$
$\{a,b,c\}$	$\{a,b,c\}$	$\{a,c\}$

Diagram prehodov

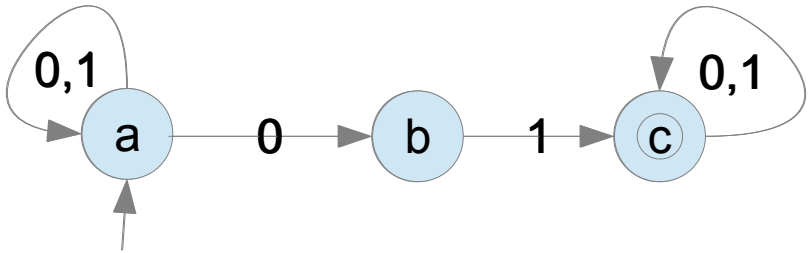




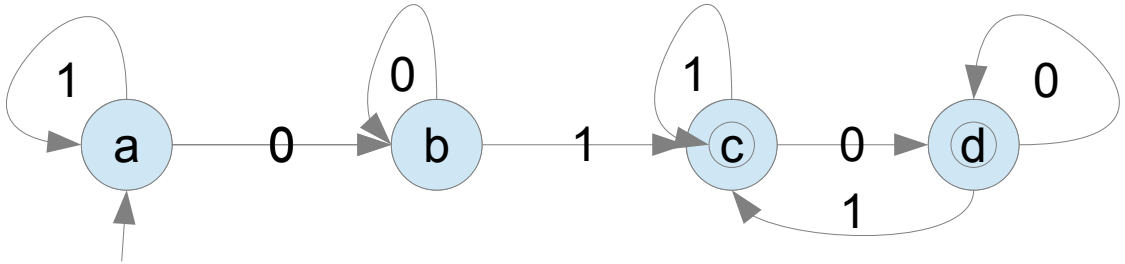


Končni rezultat

NKA:



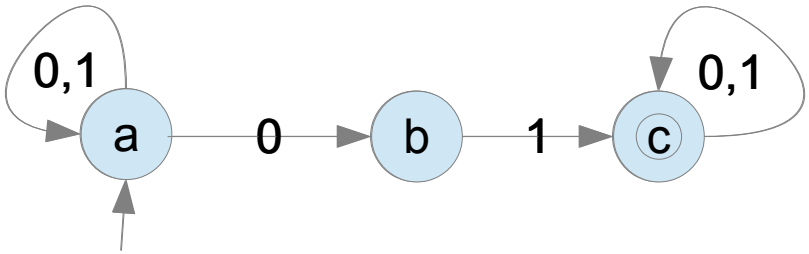
DKA:

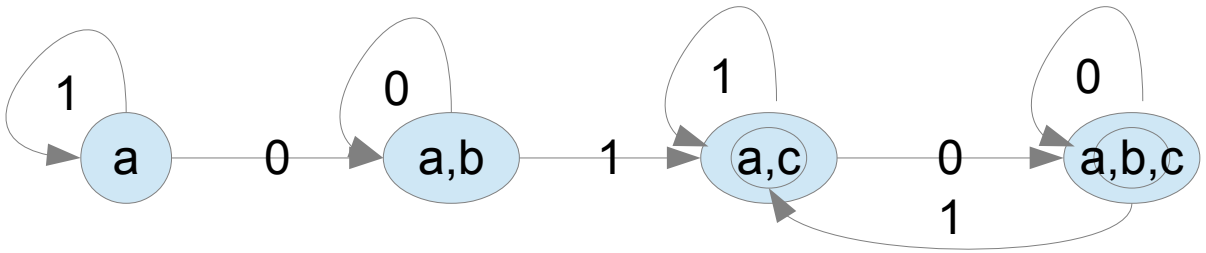


Pretvorba z razširjanjem stanj

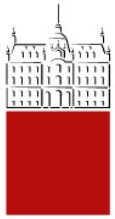
- Videli smo, da so številna stanja nedosegljiva iz začetnega stanja.
- Zato je nekoristno, če izračunamo prehode za vse mogoče podmnožice, ampak samo za dosegljive.
- Začnemo v začetnem stanju, in za vse simbole izračunamo v katere množice nas pripeljejo.
- Postopek ponavljamo, dokler ni več množic, ki jih še nismo obdelali.

Primer





Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



(Ne)moč končnih avtomatov

15.marec 2014

Moč računskega modela

- Moč računskega modela:
  - katere jezike lahko s tem računskim modelom razpoznamo
  - katere jezike lahko **učinkovito** razpoznamo
- O drugi točki več naslednjič
- V tem sklopu bomo nakazali, da obstajajo jeziki (problemi), ki jih končni avtomati ne znajo razpoznati (rešiti)

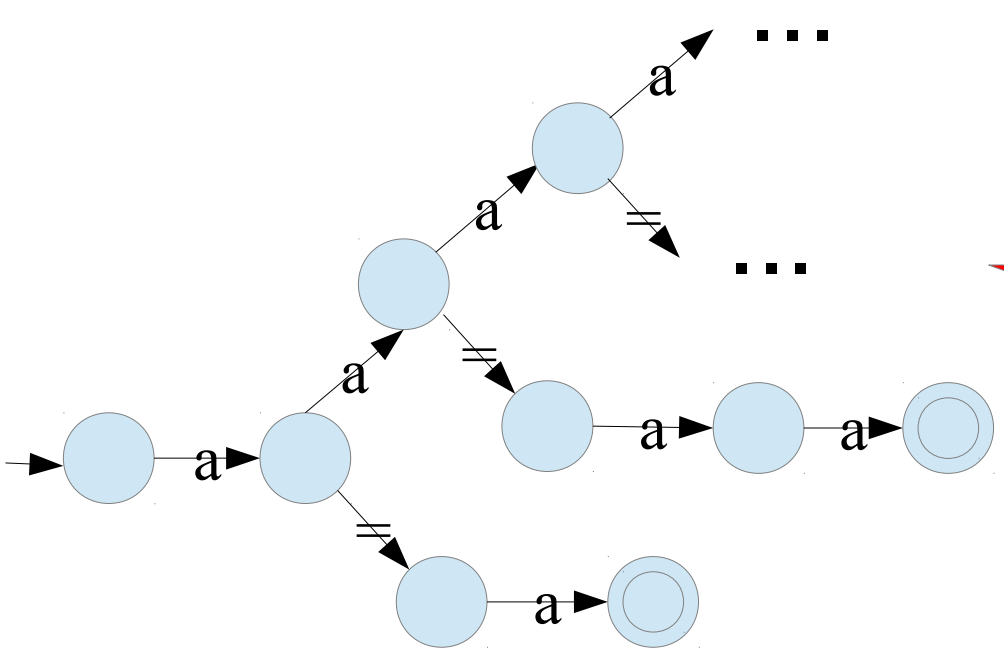
Regularni jeziki

- V prejšnjem sklopu smo pokazali, da nedeterministični avtomati prepoznavajo isto množico jezikov kot deterministični
- Tej množici jezikov pravimo regularni jeziki (ker so to natanko isti jeziki kot jih opisujejo regularni izrazi)
- Če želimo pokazati, da ta množica jezikov ne vsebuje vseh možnosti, poiščimo jezik, za katerega ne moremo zapisati KA

Trd oreh za KA

- Poskusimo sestaviti avtomat, ki bo razpoznal jezik

$$a^n = a^n$$

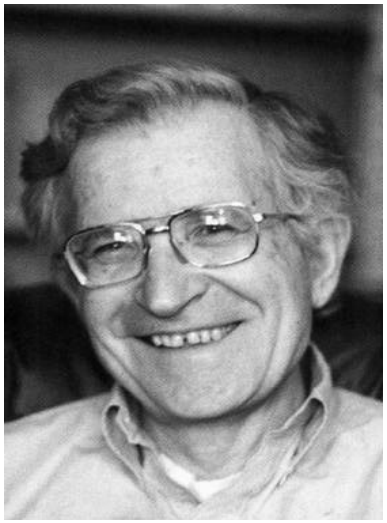


Ali lahko nekje dodamo zanko?

Močnejši formalizem kot so KA

- Končni avtomati predstavljajo formalizem za razpoznavanje jezikov
- V računalništvu pa je pomemben tudi opis in generiranje
- Eden najpomembnejši formalizmov za opis jezikov so **gramatike**.

Zgodovina



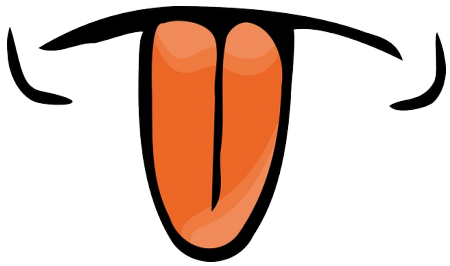
Noam Chomsky

(1956)

Gramatike

- Chomsky je želel zasnovati formalizem s katerim bi opisali naravne jezike
- Iskal je končen nabor pravil, s katerimi bi lahko zgenerirali vse slovnično pravilne stavke.
- Definiral je veliko različnih vrst gramatik.
- Mi si bomo ogledali zgolj eno vrsto gramatik, t.i. **kontekstno neodvisne gramatike**.

Primer gramatike v naravnem jeziku



Janez bere dobro knjigo.

Stavek → *Osebek Povedek Predmet*

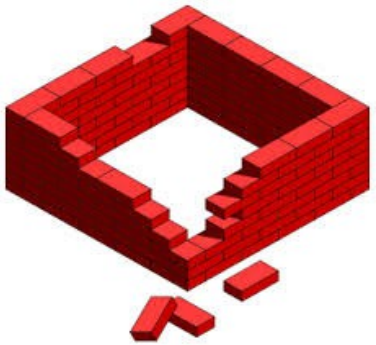
Osebek → *Janez* | *Metka* | *Mojca* | *Kekec*

Povedek → *bere* | *grize* | *meče* | *boža*

Predmet → *Pridevnik Predmet* | *knjigo* | *žogo* | *okno* | *mizo*

Pridevnik → *dobro* | *lepo* | *trdo* | *zeleno* | *nagubano*

Gradniki gramatike



- Gramatike vsebujejo 2 vrsti simbolov
  - Spremenljivke (pišemo z velikimi črkami)
  - Končne simbole (pišemo z malimi črkami)
- Pravilom, ki podajajo dovoljene transformacije spremenljivk, pa pravimo produkcije.
- Eni izmed spremenljivk dodelimo status začetnega simbola (iz nje generiramo naš jezik)

Formalna definicija

$$(V, T, P, S)$$

V Končna množica spremenljivk – npr. A, B C, S

T Končna množica končnih simbolov (abeceda) – npr. a,b, 0, 1

$$P = \{A \rightarrow \alpha : A \in V \wedge \alpha \in (V \cup T)^*\}$$

Končna množica prepisovalnih pravil (produkcij), npr.

S -> abSaS

Na levi strani puščice je spremenljivka, na desni pa poljuben niz, sestavljen iz spremenljivk in končnih simbolov.

S Spremenljivka s posebnim statusom, ki ji pravimo začetni simbol

Primer

Gramatika za jezik: $a^n = a^n$

$$V = \{S\} \qquad T = \{a, =\} \qquad P = \{S \rightarrow aSa, S \rightarrow =\}$$

Ponavadi podajamo samo produkcije, iz konteksta je razvidno vse ostalo.
Npr.

$$S \rightarrow aSa \mid =$$

Zamenjave spremenljivk in izpeljave

Besedo izpeljemo iz začetnega simbola z zamenjavami spremenljivk z desnimi stranmi produkcij. Npr., če

$$A \rightarrow w$$

potem lahko zamenjamo spremenljivko A v nekem nizu uAv (u, v sta poljubna niza) z w kar nam da niz uwv. To zapišemo kot

$$uAv \Rightarrow uwv$$

Niz **aaa=aaa** lahko v gramatiki iz prejšnje prosojnice izpeljemo iz začetnega simbola na naslednji način:

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aaaSaaa \Rightarrow aaa=aaa$$

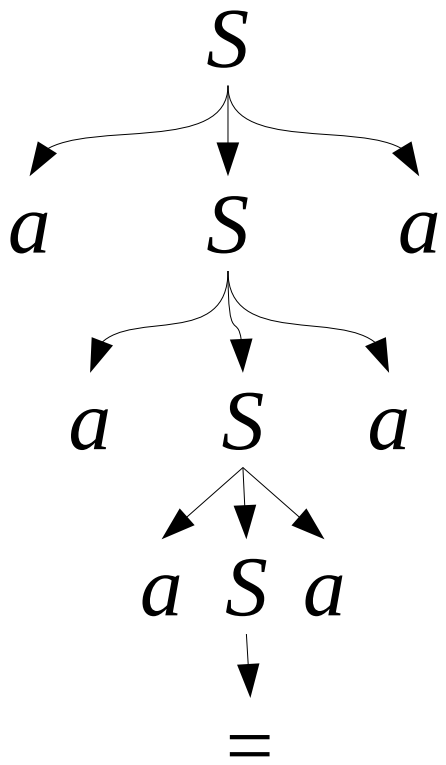
Jezik gramatike

- Jezik gramatike so vsi nizi (sestavljeni iz končnih simbolov), ki jih lahko izpeljemo v poljubnem številu zamenjav.
- Formalno to zapišemo:

$$L(G) = \{ w \mid S \overset{*}{\Rightarrow} w \}$$

Drevo izpeljave

Včasih zapišemo eno izpeljavo tudi kot drevo, npr. drevo izpeljave za niz aaa=aaa:



Dvoumnost gramatik

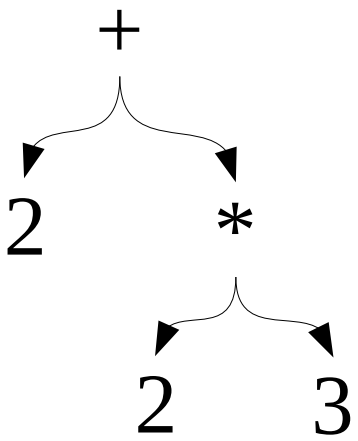
- Gramatiki pravimo dvoumna, če za nek niz v jeziku obstaja več kot eno drevo izpeljave.
- Dvoumnost igra zelo pomembno vlogo, ker z drevesi izpeljav pogosto vplivamo na »pomen« izpeljanega niza.
- Ogledali si bomo primer gramatike za aritmetične izraze in kaj mislimo s pojmom »pomen« niza.

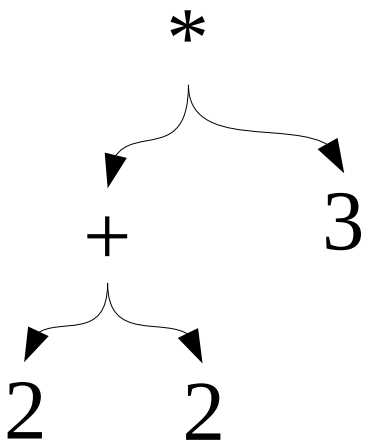
Primer – artimetični izrazi

- Zapisati želimo gramatiko, ki bo predstavljala veljavne aritmetične izraze.
- Omejili se bomo na operaciji + in *.
- Omejili se bomo tudi na števila 1, 2 in 3.
- Z gramatiko hočemo preveriti pravilnost podanega izraza in tudi definirati kako se bo ta izraz izračunal.

Drevo izraza

$$2+2*3$$





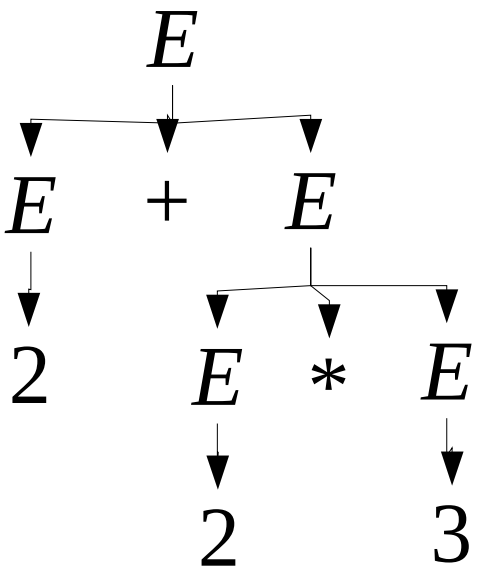
$2+(2*3)$

$(2+2)*3$

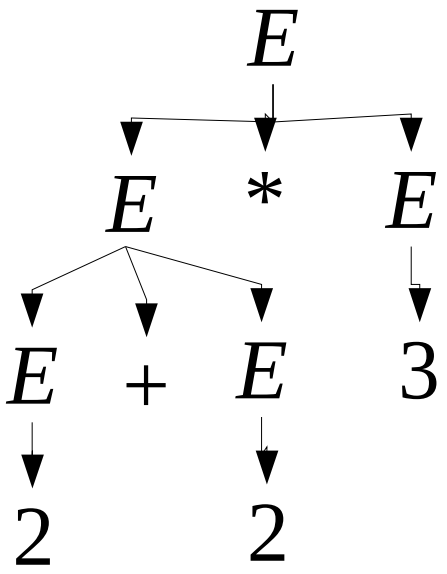
Primer – dvoumna gramatika

$$E \rightarrow E+E \mid E*E \mid 1 \mid 2 \mid 3$$

2+2*3



PRAVILNO



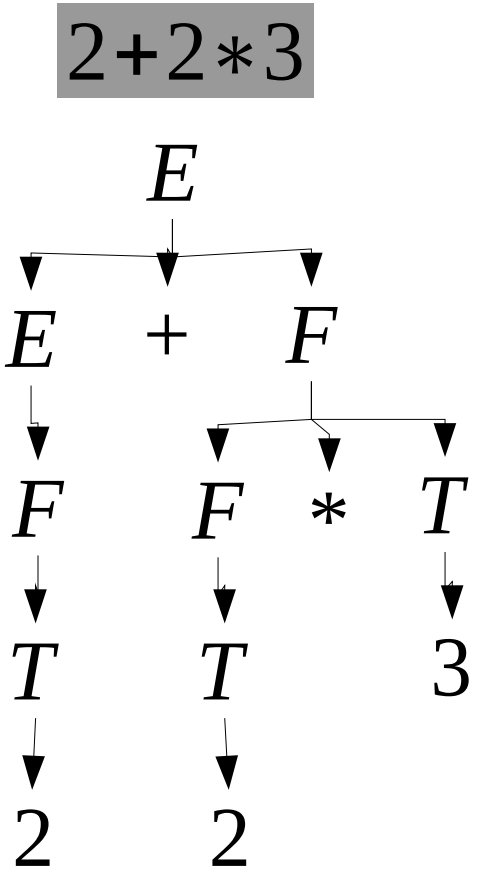
NEPRAVILNO

Primer – nedvoumna gramatika

$$E \rightarrow E + F \mid F$$

$$F \rightarrow F * T \mid T$$

$$T \rightarrow (E) \mid 1 \mid 2 \mid 3$$



Uporaba gramatik v računalništvu

- Največ se uporabljajo pri prevajanju programskih jezikov.
- Definicija oblike podatkov (XML).
- Kompresija podatkov.