

Izračunljivost in računska zahtevnost

Computability and Computational Complexity

Borut Robič

Fakulteta za računalništvo in informatiko
Univerza v Ljubljani



Borut Robič

Computability & Computational Complexity
13.december, 2023

Predavanja

- ◆ Predavanja v slovenščini
- ◆ Prosojnice v slovenščini in angleščini (na e-Ucilnica)

Literatura

♣ Prosojnice (temeljijo na naslednjih virih)

- ♣ J.E.Hopcroft, J.D.Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1st ed., 1979
- ♣ S.Arora, B.Barak. *Computational Complexity: A Modern Approach*, Cambridge University Press, 2009
- ♣ B.Robič. *The Foundations of Computability Theory*, Springer, 2015, 2020
- ♣ B.Robič. *O rešljivem, nerezljivem, obvladljivem in neobvladljivem* (dostopno na e-Ucilnica)

<https://old.delo.si/znanje/znanost/o-resljivem-neresljivem-obvladljivem-in-neobvladljivem.html>

Vsebina

- 1 Osnovni matematični pojmi
- 2 Končni avtomati in regularni izrazi
- 3 Lastnosti regularnih jezikov
- 4 Kontekstno neodvisne gramatike in jeziki
- 5 Skladovni avtomati
- 6 Lastnosti kontekstno neodvisih jezikov
- 7 Turingovi stroji
- 8 Neodločljivost
- 9 (Hierarhija Chomskega)
- 10 Računska zahtevnost
- 11 Neobvladljivi problemi
- 12 (Reševanje neobvladljivih problemov)
(aproksimacijsko, verjetnostno, vzporedno, kvantno)

Slovar

Finite automata končni avtomati/**regular expressions** regularni izrazi/**context-free grammars** kontekstno neodvisne gramatike/**pushdown automata** skladovni avtomati/**context-free languages** kontekstno neodvisni jeziki/**Turing machines** Turingovi stroji/**undecidability** neodločljivost/**Chomsky hierarchy** hierarhija Chomskega/**computational complexity** računska zahtevnost/**intractable problems** neobvladljivi problemi/**approximation algorithms** aproksimacijski algoritmi/**probabilistic (or randomized) algorithms** verjetnostni (ali naključnostni) algoritmi/**parallel algorithms** vzporedni algoritmi/**quantum algorithms** kvantni algoritmi/

Vaje, izpiti in priporočila

Vaje

asistenta: doc.dr. Uroš Čibej, doc.dr. Luka Fürst,
pričetek: od 17.oktobra dalje

Izpiti

Pravila za ocenjevanje bodo/so dostopna na e-Ucilnica.

Priporočila

Vseh prosojnic je ~ 300 . Snov je relativno zahtevna. Kako uspeti?

- ◆ Ponovite osnovno matematiko: izjavni in predikatni račun, množice, relacije, funkcije, metode dokazovanja.
- ◆ Preberite naslednjih ~ 20 prosojnic vnaprej pred naslednjim predavanjem.
- ◆ Poskusite razumeti in si pripravite vprašanja.
- ◆ Udeležujte se predavanj, sprašujte.
- ◆ Učite se sproti.
- ◆ Udeležujte se vaj, delajte domače naloge, sprašujte, bodite aktivni.

Oris predmeta

Računalništvo lahko razdelimo v dve veji:

- 1 **aplikativno/inženirsko računalništvo.**
razvija *računalniške sisteme (strojno in programsko opremo)*
- 2 **teoretično računalništvo**
raziskuje *osnovne pojme in zamisli in modele računanja*

Cilji teoretičnega računalništva. Komu je potrebno?

- ◆ Cilj teoretičnega računalništva je, da *analizira*
 - ◆ kar so inženirji *uresničili*
 - ◆ kaj bi inženirji *še lahko* (vsaj načeloma) *uresničili*
 - ◆ česa inženirji *ne bodo mogli* (niti načeloma) *uresničiti*

Kako teoretično računalništvo dosega te cilje?

- ◆ Na razne načine:
 - ◆ matematično *modelira* računske probleme in njihovo reševanje
 - ◆ *rešuje* izbrane računske probleme na *algoritmičen* način (razvija algoritme)
 - ◆ *razpoznavajo*, kateri problemi *so algoritmično rešljivi in kateri niso*
 - ◆ *ugotavlja*, kolikšne so *potrebne* in *zadostne količine virov* (čas, prostor, procesorji), da je dani problem algoritmično rešljiv
- ◆ Pri tem uporablja tudi razne *modele računanja* (tj. računske modele)

Kaj je model računanja?

- ◆ **Definicija.** Model računanja je *formalna definicija osnovnih pojmov in sestavin algoritmičnega računanja*. Model računanja *strogo definira*:
 - ◆ pojem *algoritma*,
 - ◆ *okolje*, ki je potrebno za izvedbo algoritma,
 - ◆ *izvajanje* algoritma v tem okolju.
- ◆ Modeli računanja omogočijo, da teoretično računalništvo dosega svoje cilje na *strog, matematični način* (in se tako ogiba pastem varljive intuicije).

Poznamo različne modele računanja. Zakaj?

- Teoretično računalništvo ima korenine v več področjih znanosti:
 - matematiki (problemi v *logiki in osnovah matematike*)
 - lingvistiki (gramatike *naravnih jezikov*)
 - elektrotehniki (preklopna vezja v razvoju *strojne opreme*)
 - biologija (modeli *nevronskeih mrez*)
 - (kvantni) fiziki (kvantni algoritmi in *kvantna mehanika*)
- Na teh področjih so raziskovalci definirali *specifične* modele računanja.

Nekateri modeli računanja danes veljajo za **osrednje**. To so:

- ◆ *končni avtomat*
- ◆ *skladovni avtomat*
- ◆ *Turingov stroj*

Tudi drugi so **pomembni**. Na primer:

- ◆ *dvosmerni končni avtomat, Mooreov končni avtomat, Mealyjev končni avtomat, ...*
- ◆ *linearno omejeni avtomat, ...*
- ◆ *registrski stroji (RAM, RASP, PRAM), ...*
- ◆ *splošne rekurzivne funkcije, λ -račun, μ -rekurzivne funkcije, Postov stroj, Markovski algoritmi*
- ◆ *celični avtomati (Game of Life), DNA-račun, ...*
- ◆ *podatkovno pretokovni grafi, ...*
- ◆ *kvantni Turingov stroj, kvantna vezja, ...*

1

Osnovni matematični pojmi

Vsebina

- ◆ Izjavni in predikatni račun
- ◆ Množice
- ◆ Relacije
- ◆ Formalni jeziki
- ◆ Grafi
- ◆ Metode dokazovanja

1.1 Izjavni in predikatni račun

- ◆ *Izjavni račun*
 - ◆ Logični vrednosti: *resnično (true) \top , neresnično (false) \perp*
 - ◆ Logične spremenljivke: *A, B, C, …, Z, a, b, c, …, z … imajo lahko logično vrednost*
 - ◆ Logični vezniki: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
 - ◆ Logični izrazi:
npr: $a \wedge b \Rightarrow a \vee b$
 - ◆ Tavtologije:
npr: *de Morganov zakon:* $\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$
- ◆ *Predikatni račun*
 - ◆ *Izjavni račun*
 - ◆ Predikati: *P, Q, R, … ima lahko logično vrednost*
 - ◆ Kvantifikatorji: *$\forall, \exists, \exists!$ … za vse (forall), obstaja (exists), obstaja natanko eden*
 - ◆ Formule:
npr: $\forall m \exists n: P(m, n)$ … za vsak m obstaja n tako da je $P(m, n)$ resnično
 - ◆ Tavtologije:
npr: $\neg[\forall x: P(x)] \Leftrightarrow \exists x: \neg P(x)$

1.2 Množice

- Množica je zbirka objektov (**elementov**) brez ponovitev.
- Končna množico lahko podamo kot *seznam* njenih elementov med oklepajema. **Primer.** $\{0, 1\}$ in $\{a, b, c, d, e, f, g, h\}$ sta množici.
- Množico lahko podamo s karakterističnim predikatom P :
or $\{x \mid P(x)\}$... the set of objects x such that $P(x)$ is true
 $\{x \in A \mid P(x)\}$... the set of x in A such that $P(x)$ is true
- **Primer.** $\{i \in \mathbb{N} \mid \text{there is integer } j \text{ such that } i = 2j\}$
 $\{i \in \mathbb{N} \mid \exists j(j \in \mathbb{N}) : i = 2j\}$

- ◆ Če je vsak element množice A tudi element množice B , pišemo $A \subseteq B$ in rečemo, da je A **podmnožica** množice B . ($B \supseteq A$ pomeni isto kot $A \subseteq B$.)
- ◆ Če je $A \subseteq B$ in hkrati $A \neq B$, pišemo $A \subsetneq B$ (oz. $A \subset B$). V tem primeru pravimo, da je A **prava podmnožica** B .
- ◆ Množici A in B sta **enaki** če imata iste elemente. To je, $A = B$ iff $A \subseteq B$ in $B \subseteq A$.
(iff pomeni "če in samo če" ("if and only if").)

Operacije nad množicami:

- $A \cup B$, **unija** množic A in B , je množica $\{x \mid x \in A \text{ or } x \in B\}$
- $A \cap B$, **presek** množic A in B , je množica $\{x \mid x \in A \text{ and } x \in B\}$
- $A \setminus B$, **razlika** množic A in B , je množica $\{x \mid x \in A \text{ and } x \notin B\}$
- $A \times B$, **kartezični produkt** množic A in B , je množica
$$\{(x, y) \mid x \in A \text{ and } y \in B\}$$
- 2^A , **potenčna množica** množice A , je množica $\{X \mid X \subseteq A\}$
(Oznaka $\mathcal{P}(A)$ pomeni isto kot 2^A .)

- ◆ Množici A in B imata *isto moč*, če obstaja *bijektivna* funkcija $f:A \rightarrow B$.
 - ◆ *Končne množice*:
 - ◆ Če je A končna množica, je njena moč *naravno število* (število elementov).
 - ◆ Če sta A, B končni množici in $A \subset B$, potem imata A in B *različni* moči.
 - ◆ *Neskončne množice* :
 - ◆ Če sta A, B neskončni množici in $A \subset B$, potem imata A in B *lahko isto moč!*
 - ◆ *Primer.* Naj bo $A = \text{Soda} \mathbb{N}$ in $B = \mathbb{N}$. Velja $A \subset B$, toda obstaja funkcija $f: A \rightarrow B$, namreč $f: i \mapsto i/2$, ki je *bijektivna*. (Torej je sodih ravno toliko kot vseh naravnih.)
 - ◆ Neskončne množice imajo *lahko različne* moči. *Primer.* \mathbb{N} in \mathbb{R} .
 - ◆ Množica A , ki se da *injektivno* preslikati v \mathbb{N} , je *števna* (če A končna) ali *števno neskončna* (če A neskončna).
 - ◆ *Primer.* \mathbb{Q} je števno neskončna. $2^{\mathbb{N}}$ in množica vseh funkcij $f: \mathbb{N} \rightarrow \{0,1\}$ imata isto moč kot množica \mathbb{R} , zato *nista števni*.

1.3 Relacije

- Binarna relacija R je množica parov:

$$R = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

- Če je (a, b) element R , pišemo tudi aRb .
- Prva komponenta para je element množice A (**domene** relacije R), druga komponenta pa iz množice B (**zaloge vrednosti**, oz. kodomene).
- Če je $A = B = S$, pravimo, da je R relacija **na množici** S .

Pomembne **lastnosti relacij**, ki jih dana relacija R na S lahko ima.
Pravimo, da je relacija R na množici S

- ◆ **refleksivna**, če aRa za vsak $a \in S$ tj. $\forall a \in S : aRa$
- ◆ **irefleksivna**, če aRa ne velja, za vsak $a \in S$ tj. $\forall a \in S : \neg(aRa)$
- ◆ **tranzitivna**, če iz aRb in bRc sledi aRc ... tj. $\forall a, b, c \in S : aRb \wedge bRc \Rightarrow aRc$
- ◆ **simetrična**, če iz aRb sledi bRa tj. $\forall a, b \in S : aRb \Rightarrow bRa$
- ◆ **asimetrična**, če iz aRb sledi da bRa ne velja tj. $\forall a, b \in S : aRb \Rightarrow \neg(bRa)$

Očitno: Če je R asimetrična, potem je tudi irefleksivna.

- ◆ Primer. Relacija $<$ na \mathbb{Z} je tranzitivna in asimetrična (zato irefleksivna).

Relacijski R , ki je refleksivna, simetrična in tranzitivna, pravimo **ekvivalenčna relacija**.

- ◆ Ekvivalenčna relacija R na S razdeli množico S na *disjunktne neprazne* podmnožice S_1, S_2, \dots , imenovane **ekvivalenčni razredi**.
- ◆ Če je R na S ekvivalenčna relacija, potem je $S = S_1 \cup S_2 \cup \dots$, kjer za $i \neq j$ velja
 - ◆ $S_i \cap S_j = \emptyset$
 - ◆ za poljubna $a, b \in S_i$ velja aRb
 - ◆ za poljubna $a \in S_i$ and $b \in S_j$ velja $\neg(aRb)$
- ◆ *Opomba:* ekvivalenčnih razredov je lahko končno ali neskončno mnogo.

Primer.

- ◆ Definirajmo relacijo R na \mathbb{Z} takole: aRb iff $a = b \bmod m$.
- ◆ *Trditev.* R je ekvivalenčna relacija na \mathbb{Z} . (Dokažite za vajo.)
- ◆ Ekvivalenčni razredi relacije R so
 - ◆ $S_0 = \{\dots, -2m, -m, 0, m, 2m, \dots\}$
 - ◆ $S_1 = \{\dots, -2m+1, -m+1, 1, m+1, 2m+1, \dots\}$
 - ◆ \vdots
 - ◆ $S_{m-1} = \{\dots, -2m-1, -1, m-1, 2m-1, 3m-1, \dots\}$

Naj bo P množica (nekaterih poljubno izbranih) lastnosti relacij. **P -zaprtje** relacije R je najmanjša relacija, ki vsebuje R in ki ima vse lastnosti, naštete v P .

Primer.

- ◆ $P = \{\text{tranzitivnost}\}$.

P -zaprtje relacije R je relacija, ki jo označimo z R^+ , in poimenujemo **tranzitivno zaprtje** relacije R . Za R^+ velja:

- 1) Če aRb , potem aR^+b .
- 2) Če aR^+b in bRc , potem aR^+c .
- 3) xR^+y velja natanko tedaj, ko to sledi samo iz 1) and 2).

Intuitivno: R^+ nastane iz R , če v R dodamo najmanjše število parov, da “razširjena množica” postane tranzitivna.

Primer::

- ◆ $P = \{refleksivnost, tranzitivnost\}$.
 P -zaprtje relacije R označimo z R^* , in imenujemo **refleksivno-tranzitivno zaprtje** relacije R . Za R^* velja:
- ◆ $R^* = R^+ \cup \{(a,a) \mid a \in S\}$.

1.4 Formalni jeziki

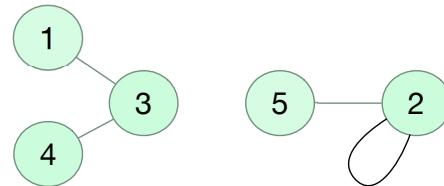
- ◆ **Simbol** je osnovni pojem, ki ga zato ne definiramo formalno.
 - ◆ **Primer.** Črke, števke, ločila in matematični znaki so običajni simboli.
- ◆ **Beseda** (oz. **niz**) je *končno zaporedje staknjenih simbolov*.
 - ◆ **Primer.** $bbac$ je beseda, v kateri so staknjeni simboli a , b in c .
- ◆ **(Besedne) spremenljivke** zavzemajo vrednosti, ki so besede. Pogosto (ne vedno) jih označimo s simboli u , v , w , x , y , z s konca latinice.
 - ◆ **Primer.** $u = bb$; $v = a$; $w = bbac$. **Primer.** $x = u$ (spr. x je enaka spr. u).
- ◆ **Dolžina** $|w|$ besede w je število pojavov simbolov v w .
 - ◆ **Primeri.** $|a| = 1$; $|bb| = 2$; Če $w = bbac$, je $|w| = |bbac| = 4$.
- ◆ **Prazna beseda** je beseda dolžine 0. Označimo jo z ϵ . Torej, $|\epsilon| = 0$.

- ◆ **Predpona** besede je niz vodilnih simbolov besede. **Pripona** besede je niz zadnjih simbolov besede. Predpona (pripona) besede je **prava**, če ni enaka besedi.
 - ◆ **Primer.** Predpone besede $bbac$ so ϵ, b, bb, bba in $bbac$; pripone pa so ϵ, c, ac, bac in $bbac$. Beseda $bbac$ ni niti prava predpona besede $bbac$, niti njena prava pripona.
- ◆ **Stik** dveh besed je beseda, sestavljena iz prve besede, ki ji brez presledka sledi druga beseda.
 - ◆ **Primer.** Stik besed $bbac$ in aa je $bbacaa$. **Primer.** Staknemo lahko tudi besedni spremenljivki: če je npr. $u = bbac$ in $v = aa$, je stik spremenljivk u in v beseda $uv = bbacaa$.
- ◆ Operacija, ki besedama privedi njun stik, se tudi imenuje *stik*. Eksplicitno je ne označimo: če sta u in v besedi, rezultat operacije stik zapišemo z uv (ne pa, npr., z $u \circ v$ ali $u+v$). Prazna beseda ϵ je *nevtralni element* za operacijo *stika* (ker $\epsilon w = w\epsilon = w$, za vsak w).

- ◆ Abeceda je končna množica simbolov.
- ◆ Formalni jezik nad (dano) abecedo je množica (nekaterih) besed, ki jih sestavljajo simboli te abecede. Prazna množica \emptyset in množica $\{\varepsilon\}$ sta tudi formalna jezika (različna!) (čeprav ne zelo zanimiva).
 - ◆ Primer. Množica palindromov (besed, ki se naprej in nazaj berejo enako) nad abecedo $\{0,1\}$ je neskončen jezik $\{\varepsilon, 0, 1, 00, 11, 000, 010, 101, 111, \dots\}$
- ◆ Pomemben jezik je množica vseh besed nad dano abecedo Σ . Označimo ga s Σ^* .
 - ◆ Primer. Če je $\Sigma = \{a\}$, potem je $\Sigma^* = \{\varepsilon, a, aa, aaa, aaaa, \dots\}$. Če je $\Sigma = \{0,1\}$, potem je $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \dots\}$.

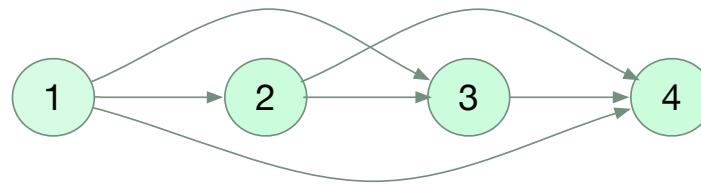
1.5 Grafi

- ◆ Neusmerjen graph $G = (V, E)$ sestavlja množica V točk (oz. vozlišč) in množica E parov točk, imenovanih povezave.
 - ◆ Primer. $V = \{1, 2, 3, 4, 5\}$, $E = \{\{n, m\} \mid n+m = 4 \text{ ali } n+m = 7\}$.



- ◆ Pot v grafu je zaporedje točk v_1, v_2, \dots, v_k , $k \geq 1$, kjer je $\{v_i, v_{i+1}\}$ za vsak i , $1 \leq i < k$ povezava v grafu. Dolžina poti je $k - 1$.
 - ◆ Primer. Zaporedje 1,3,4 je pot dolžine 2; tudi zaporedje 5 je (trivialna) pot (dolžine 0).
- ◆ Če je (v zgornji definiciji) $v_1 = v_k$, je pot cikel.

- Usmerjen graf $G = (V, A)$ sestavlja množica točk V in množica A urejenih parov točk, imenovanih **usmerjene povezave**. Usmerjeno povezavo (u, v) označimo tudi z $u \rightarrow v$.
- Primer.

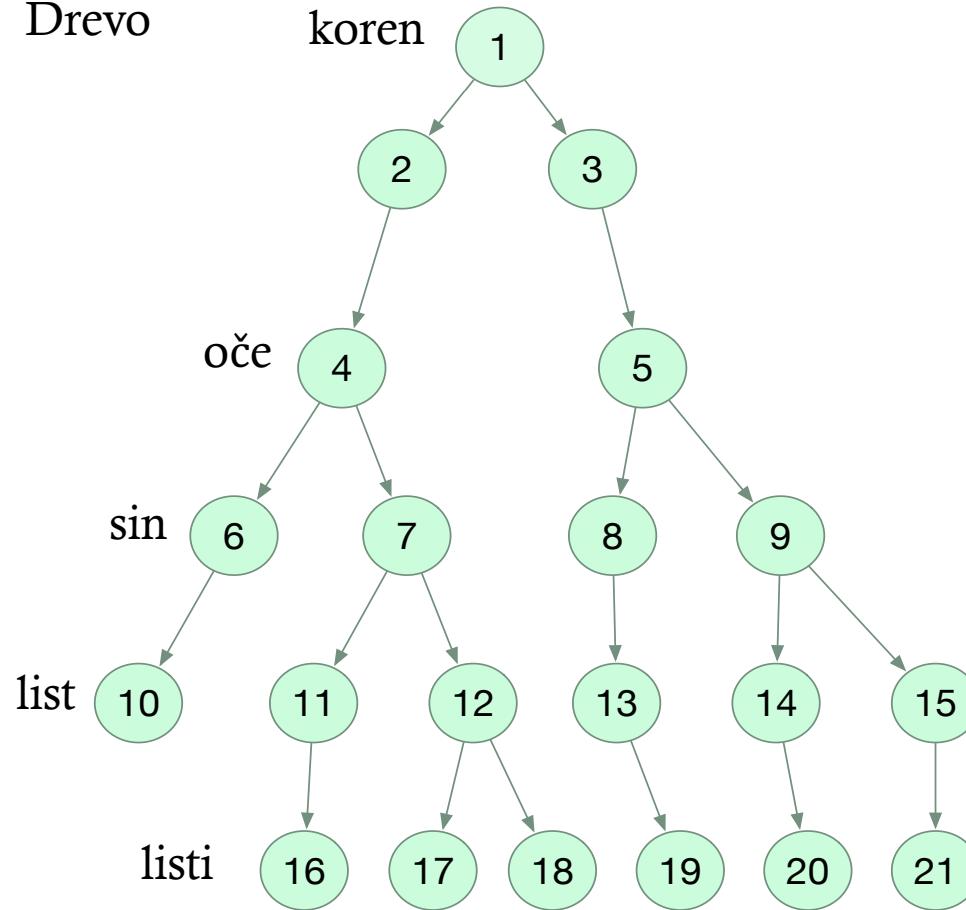


- Pot v usmerjenem grafu je zaporedje točk v_1, v_2, \dots, v_k , $k \geq 1$, kjer je $v_i \rightarrow v_{i+1}$ za vsak i , $1 \leq i < k$, usmerjena povezava grafa. Rečemo, da gre ta pot iz v_1 v v_k . Če je $u \rightarrow v$, je u **neposredni prednik** točke v (oz. v **neposredni naslednik** točke u).
- Primer. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ je pot iz 1 v 4. Tu je 3 predhodnik 4.

- ◆ Drevo je usmerjen graf z (dodatnimi) lastnostmi:
 - ◆ 1) Obstaja natanko ena točka, imenovana **koren**, ki nima prednikov in iz katerega obstaja pot do vsake točke.
 - ◆ 2) Vsaka točka, ki ni koren, ima natanko enega prednika.
 - ◆ 3) Če ima točka naslednike, so ti urejeni (običajno od leve proti desni).
- ◆ Neposrednemu nasledniku točke rečemo tudi **sin**, neposrednemu predniku pa **oče**. Če obstaja pot iz v_i v v_j , rečemo, da je v_i **prednik** točke v_j (oz. v_j **potomec** točke v_i). Če točka nima sinov, ji rečemo **list**, sicer pa **notranja točka drevesa**.

Primer.

Drevo



1.6 Metode dokazovanja

- Nekatere izreke lahko dokažemo z matematično indukcijo.
- Princip matematične indukcije:

Naj bo $P(n)$ trditev o naravnih številih n . Tedaj:

Če $P(0)$ velja in če velja $P(k-1) \Rightarrow P(k)$ za poljuben $k \geq 1$, potem $P(n)$ velja za vsak naravni n . (Tu 0 štejemo za naravno št.)

- $P(0)$ je **osnova** indukcije, $P(k-1)$ je **induktivna hipoteza**, $P(k-1) \Rightarrow P(k)$ pa **induktivni korak**.

Primer. Trditev:

$$P(n) \equiv \sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

DOKAZ.

Osnova [preverimo $P(0)$] $\sum_{i=0}^0 i^2 = 0 = \frac{0(0+1)(2 \cdot 0 + 1)}{6}$. Torej $P(0)$ velja.

Ind. hipoteza [predpostavimo, da $P(k-1)$ velja] $\sum_{i=0}^{k-1} i^2 = \frac{(k-1)k(2k-1)}{6}$

Ind. korak [dokažemo, da velja $P(k-1) \Rightarrow P(k)$. Pri tem se sklicujemo na ind. hiptezi]

$$\sum_{i=0}^k i^2 = \sum_{i=0}^{k-1} i^2 + k^2 \stackrel{\text{Ind. hyp.}}{=} \frac{(k-1)k(2k-1)}{6} + k^2 = \dots = \frac{k(k+1)(2k+1)}{6}$$

Torej $P(k)$ velja. Ugotovili smo, da velja $P(k-1) \Rightarrow P(k)$.

Sklep: $P(n)$ velja za vsa naravna števila n .

QED.

1.7 Vaje

1. A palindrome can be defined as a string that reads the same forward and backward, or by the following definition:

- a) ϵ is a palindrome.
- b) If a is any symbol, then the string a is a palindrome.
- c) If a is any symbol and x is a palindrome, then axa is a palindrome.
- d) Nothing is a palindrome unless it follows from (1, 2, 3).

Prove by induction that the two definitions are equivalent.

2. The strings of balanced parentheses can be defined in at least two ways.

A string w over alphabet $\{(,)\}$ is balanced if and only if:

- 1) a) w has an equal number of '('s and ')'s, and
b) any prefix of w has at least as many '('s as ')'s.
- 2) a) ϵ is balanced.
b) If w is a balanced string, then (w) is balanced.
c) If w and x are balanced strings, then so is wx
d) Nothing else is a balanced string.

Prove by induction on the length of a string that (1) and (2) define the same class of strings.

3. Show that the following are equivalence relations and give their equivalence classes.

- a) The relation R_1 on integers defined by iR_1j if and only if $i = j$.
- b) The relation R_2 on people defined by pR_2q if and only if p and q were born at the same hour of the same day of some year.
- c) The same as (b) but "of the same year" instead of "of some year."

4. Find the transitive closure, the reflexive and transitive closure, and the symmetric closure of the relation

$$\{(1,2), (2, 3), (3, 4), (5, 4)\}$$

1.8 Slovar

Symbol simbol letter črka digit števka string niz word beseda length dolžina empty string prazna beseda prefix predpona suffix pripona proper pravi concatenation stik to juxtapose stakniti, pripeti juxtaposition stik alphabet abeceda formal language formalni jezik palindrome palindrom graph graf vertex node vozlišče edge povezava path pot cycle cikel directed graph usmerjen graf predecessor predhodnik successor naslednik tree drevo ancestor prednik descendant potomec leaf list interior vertex notranje vozlišče mathematical induction matematična indukcija inductive hypothesis induktivna hipoteza basis osnova inductive step korak indukcije, indukcijski korak set množica member pripadnik, element contain vsebovati properly contain strogo vsebovati equal enak operation operacija union unija intersection presek difference razlika Cartesian product kartezični produkt power set potenčna množica cardinality moč countable števna countably infinite števno neskončna binary relation binarna relacija domain domena range zaloga vrednosti reflexive refleksivna irreflexive irefleksivna transitive tranzitivna symmetric simetrična asymmetric asimetrična equivalence relation ekvivalenčna relacija partition razbitje, razdelitev equivalence class ekvivalenčni razred transitive closure tranzitivna ovojnica (tr. zaprtje) reflexive and transitive closure refleksivna tranzitivna ovojnica (refl. tr. zaprtje) model of computation računski model finite automaton končni avtomat regular expression regularni izraz pushdown automaton skladovni avtomat context-free grammar kontekstno neodvisna gramatika Turing machine Turingov stroj intractable (or hard) problem neobvladljiv (oz. težek) problem

2

Končni avtomati in regularni izrazi

Vsebina

- ◆ Sistemi s končno mnogo stanji
- ◆ Deterministični končni avtomati, DKA
- ◆ Nedeterministični končni avtomati, NKA
- ◆ Ekvivalentnost DKA in NKA
- ◆ Končni avtomati s tihimi prehodi, NKA_{ϵ}
- ◆ Ekvivalentnost NKA in NKA_{ϵ}
- ◆ Regularni izrazi
- ◆ Ekvivalentnost končnih avtomatov in regularnih izrazov

2.1 Sistemi s končno mnogo stanji

- ◆ Sistem s končno mnogo stranji je entiteta, ki
 - ◆ je vedno v kakem od *končno mnogih stanj*,
 - ◆ korakoma bere *diskretne* podatke iz svoje okolice,
 - ◆ in po vsakem branju lahko spremeni svoje stanje v novo stanje.
 - ◆ Stanje, v katerem je sistem v nekem trenutku, predstavlja dotej prebrane podatke. Na podlagi stanja bo sistem določil svoj odziv na naslednji podatek iz okolice.

Končni avtomat je a *matematični model* sistemov s končno mnogo stanji.

► **Primeri.** Poznamo razne sisteme, ki imajo končno mnogo stanj.

- **Preklopna vezja** (npr. procesor v računalniku)
 - A switching circuit consists of a *finite* number of *gates*, each of which can be in one of *two* conditions, 0 and 1 (different voltage levels at the gate output).
 - The *state* of a circuit with n gates can be any one of 2^n assignments of 0 or 1 to the gates.
 - (Comment. The circuitry is so designed that only the two voltages corresponding to 0 and 1 are stable; other voltages immediately adjust themselves to one of these voltages. Switching circuits are intentionally designed in this way, so that they can be viewed as FSSs, thereby separating the *logical design* of a computer from the *electronic implementation*.)
- **Nekateri programi** (npr. urejevalnik besedil, leksikalni analizator pri prevajalniku)
 - A *lexical analyzer* scans the symbols of a computer program to locate the strings of characters corresponding to *identifiers*, *numerical constants*, *reserved words*, and so on. In this process the lexical analyzer needs to remember only a *finite* amount of information (e.g., the length of a prefix of a reserved word that has seen since startup).
- **Še mnogo več ...** (npr. dvigalo; TV + daljinec; kavomat, pralni stroj ...)
- Razvoj takih sistemov se naslanja na *teorijo končnih avtomatov*.

Opomba.

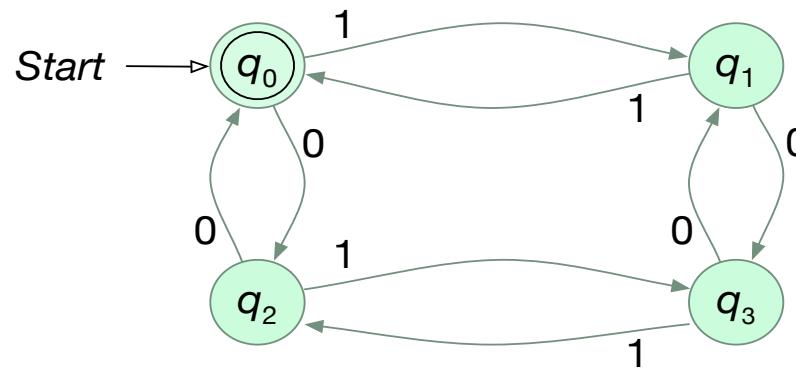
- ◆ Kasneje bomo videli, da je *končni avtomat* model računanja, ki pa ne zajema vsega, kar si *intuitivno* predstavljamo kot računanje.
 - ◆ Zakaj? Da bi zajeli vse, kar si intuitivno predstavljamo kot računanje, potrebujemo model računanja s *potencialno neskončnim pomnilnikom*.
Takšen model bo na primer Turingov stroj.

2.2 Deterministični končni avtomat

- ◆ Intuitivno: deterministični končni avtomat (DKA) sestavlja:
 - ◆ končna množica stanj in
 - ◆ končna množica prehodov med stanji, ki se zgodijo po branju simbolov iz vhodne abecede (npr. Σ)
- Pri tem velja:
 - ◆ za vsak vhodni simbol in vsako stanje obstaja natanko en prehod;
 - ◆ eno stanje, označeno s q_0 , je začetno stanje. DKA začne delovanje v q_0 ;
 - ◆ nekatera stanja so končna .
- ◆ Pravimo, da DKA sprejme besedo x , če zaporedje prehodov, ki utreza zaporednim simbolom besede x , vodi DKA iz začetnega stanja q_0 v kako končno stanje.

- DKA lahko predstavimo z diagramom prehodov, v katerem
 - točke predstavljajo stanja DKA;
 - usmerjene povezave pa prehode DKA: povezava $q_i \xrightarrow{a} q_j$ obstaja, če DKA lahko preide iz stanja q_i v q_j pri prebranem simbolu a .

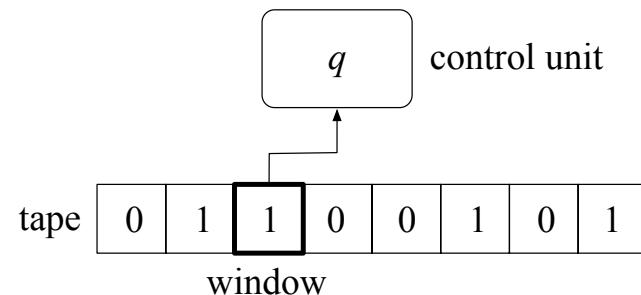
Primer.



q_0 is the initial state. Final states are in double circles (here q_0).

- ◆ **Definicija.** Deterministični končni avtomat (**DKA**) je peterka $(Q, \Sigma, \delta, q_0, F)$, kjer so:
 - ◆ Q končna množica **stanj**,
 - ◆ Σ **vhodna abeceda**,
 - ◆ $q_0 \in Q$ **začetno stanje**,
 - ◆ $F \subseteq Q$ množica **končnih stanj**,
 - ◆ δ **funkcija prehodov**, $\delta : Q \times \Sigma \rightarrow Q$.Torej je $\delta(q, a)$ stanje, v katerega DKA preide iz stanja q , če je prebral simbol a .
- ◆ *Opomba:* δ je *program* DKA. Vsak DKA ima svoj specifični program δ .

- DKA lahko **predstavimo** tudi takole: **nadzorna enota** bere vhodno besedo ($\in \Sigma^*$) s **traku** in pri tem prehaja med stanji.



- Če DKA v stanju q prebere vhodni simbol a , potem v eni **potezi**:
 - *preide* v naslednje stanje, tj. v stanje $\delta(q, a)$,
 - *pomakne okno* v desno, na naslednji simbol vhodne besede.
- Če je $\delta(q, a)$ končno stanje, pravimo, da je DKA **sprejel** predpono vhodne besede vključno do prebranega a . DKA lahko sprejme več predpon vhodne besede, običajno pa zahtevamo, da prebere celo vhodno besedo.

- Izkaže se za koristno (nasl.prosojnjica), če razširimo δ tako, da je lahko njen drugi argument niz simbolov (ne le en simbol).
- Iščemo funkcijo $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$, takšno, da bo $\hat{\delta}(q, x)$ stanje, kamor pride DKA po tem, ko je v stanju q začel brati in potem prebral niz x . V diagramu prehodov bo $\hat{\delta}(q, x)$ stanje, do katerega vodi pot iz q , označena z nizom x .
- Razširjeno funkcijo prehodov $\hat{\delta}$ definiramo rekurzivno tako:
 - $\hat{\delta}(q, \varepsilon) = q$
 - $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$, za vsak niz w in vhodni simbol a



- Ker je $\hat{\delta}(q, a) = \delta(q, a)$ (dokaži!), ni neujemanja med $\hat{\delta}$ in δ .
Zaradi enostavnosti običajno pišemo kar δ namesto $\hat{\delta}$ in pomnimo, da zdaj δ deluje nad nizi.

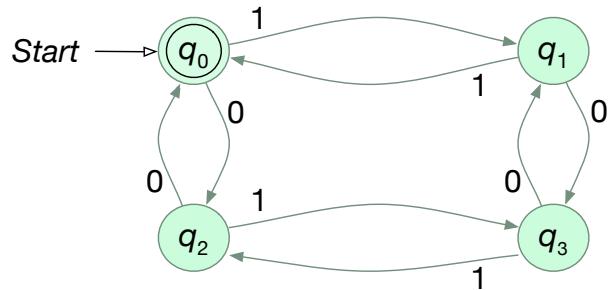
♦ Definicije.

- ♦ DKA $M = (Q, \Sigma, \delta, q_0, F)$ sprejme besedo (niz) x , če je $\delta(q_0, x) = p$ za neki $p \in F$.
- ♦ Jezik, sprejet z DKA M je množica $L(M)$ vseh besed, ki jih sprejme M :

$$L(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \in F\}$$

- ♦ Jezik L' je regularen (oz. je regularna množica), če L' sprejme *kak* DKA (tj. če \exists DFA $M : L' = L(M)$).

Primer.



q_0 is the initial state. Final states are in double circles (here q_0).

- Zgoraj je diagram prehodov DKA $M = (Q, \Sigma, \delta, q_0, F)$, kjer so

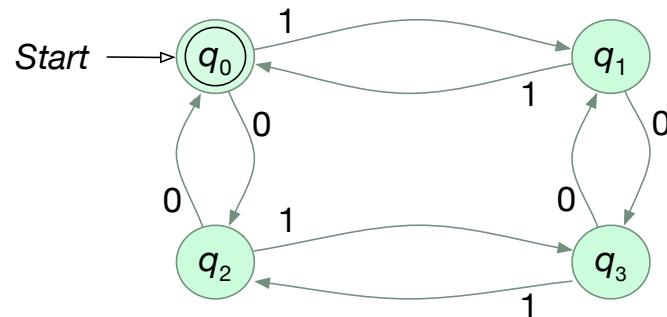
$$Q = \{q_0, q_1, q_2, q_3\}$$

$$\Sigma = \{0, 1\}$$

$$F = \{q_0\}$$

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Primer (nadaljevanje)



δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

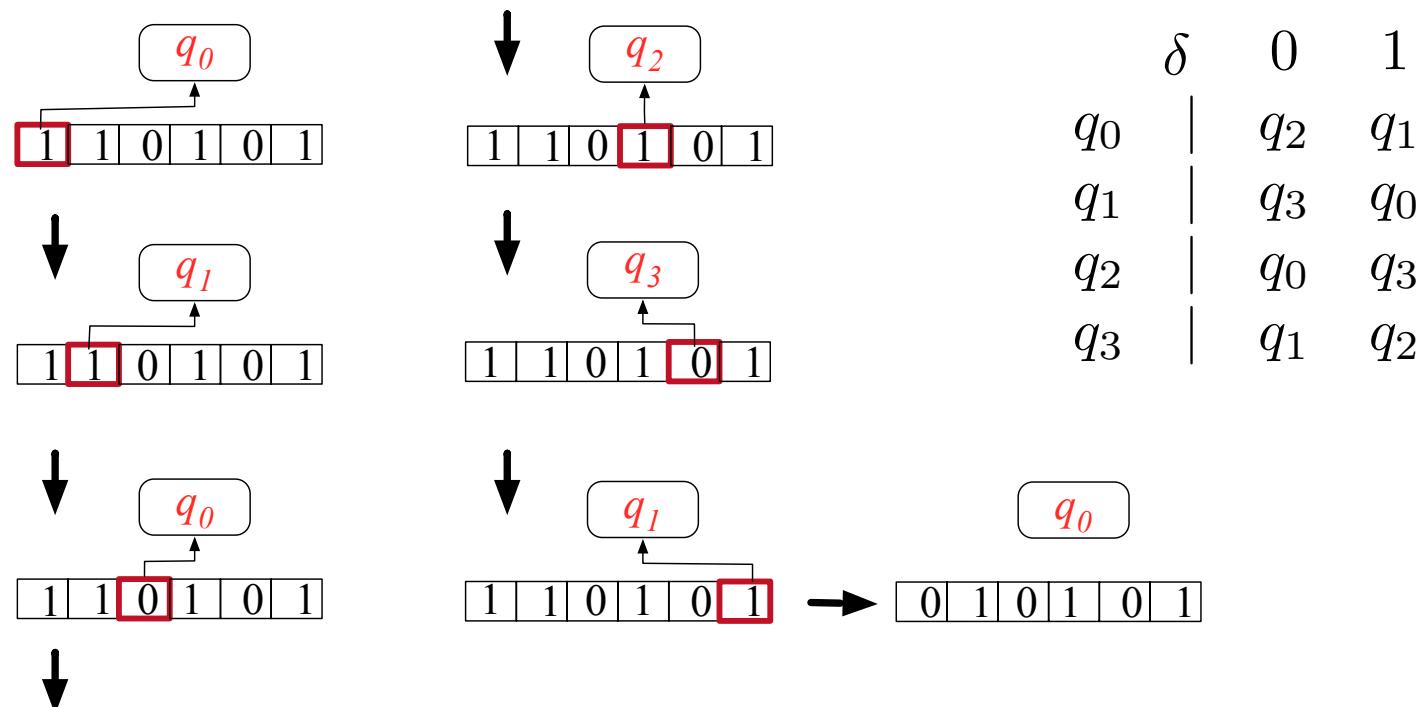
q_0 is the initial state. Final states are in double circles (here q_0).

- ♣ Naj bo $x = 110101$ vhodna beseda (niz) v DKA M . Ali je $x \in L(M)$?
 - ♣ Izračunati moramo stanje $\delta(q_0, x) = \delta(q_0, 110101)$ in preveriti, ali je v F .
 - ♣ $\delta(q_0, 110101)$

$$\begin{aligned}
 &= \delta(q_1, 10101) \\
 &= \delta(q_0, 0101) \\
 &= \delta(q_2, 101) \\
 &= \delta(q_3, 01) \\
 &= \delta(q_1, 1) = q_0 \in F
 \end{aligned}$$

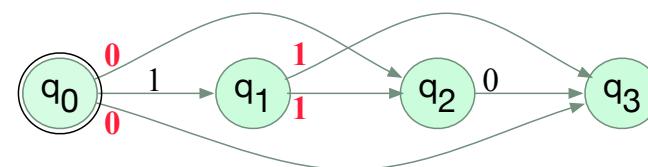
Primer (nadaljevanje).

- Izvajanje avtomata M pri vhodni besedi $x = 110101$.



2.3 Nedeterministični končni avtomat

- Nedeterministični končni avtomat (NKA) dobimo iz DKA, če dopustimo *nič*, enega ali celo več prehodov iz kakega stanja pri istem vhodnem simbolu. Npr.



- NKA **sprejme** vhodno besedo $x = a_1 a_2 \cdots a_n$, če **obstaja** (vsaj eno) zaporedje prehodov, označeno z $a_1 a_2 \cdots a_n$, ki vodi iz začetnega stanja v (katerokoli) končno stanje.
- Pri **DKA** velja: za dano stanje q in besedo x gre iz q natanko ena pot označena z x . Za ugotovitev ali DKA sprejme x zadošča, da preverimo, ali ta pot vodi v končno stanje. Pri **NKA** je bolj zapleteno: tu gre lahko iz q več poti, označenih z x . Za ugotovitev ali NKA sprejme x , moramo preveriti, ali vsaj ena vodi v končno stanje. Torej moramo v najslabšem primeru preveriti vse!

◆ Nedeterminizem

- ◆ **Vprašanje:** Če je dana vhodna beseda $x = a_1 a_2 \cdots a_n$, **kdo bo ugotovil ali obstaja** zaporedje prehodov, ki se konča v končnem stanju?
Odgovor: NKA sam!
- ◆ **Vprašanje:** **Kako** bo NKA to ugotovil?
Odgovor: NKA *ni realističen* model računanja: *predpostavljamо namreč, da NKA to vedno pravilno ugane.* Lahko si mislimo, da ima NKA *magično sposobnost*, da med več možnimi prehodi vedno izbere pravega (če je med njimi kakšen tak), tj. tistega, ki ga vodi k sprejetju. Če pravega prehoda ni, NKA to magično ugotovi in se ustavi.
- ◆ Zaradi te svoje magične lastnosti je NKA *neuresničljiv, nerealističen* model.

◆ Nedeterminizem (nadaljevanje)

◆ Vprašanje: Čemu koristi nerealistični model računanja NKA?

Odgovora:

- ◆ NKA in tudi drugi nedeterministični modeli računanja, ki jih bomo še spoznali, koristijo pri *določanju spodnjih meja časa*, potrebnega za rešitev računskega problema. Podlaga pri tem je naslednji razmislek:

Če reševanje danega problema P z *nedeterminističnim* modelom računanja M zahteva T časa, potem bo reševanje problema P s *kakršnokoli deterministično različico* D modela M zahtevalo *vsaj* T časa (ker D nima magične sposobnosti pravilnega ugibanja).

- ◆ Pogosto je problem P *lažje sestaviti* NKA (ali kak drug *nedeterministični* model M). Ko je M sestavljen, poskušamo iz M sestaviti *ekvivalentno deterministično* različico D (ekvivalentno v smislu, da *tudi D reši problem P*, čeprav v nekem drugem času).

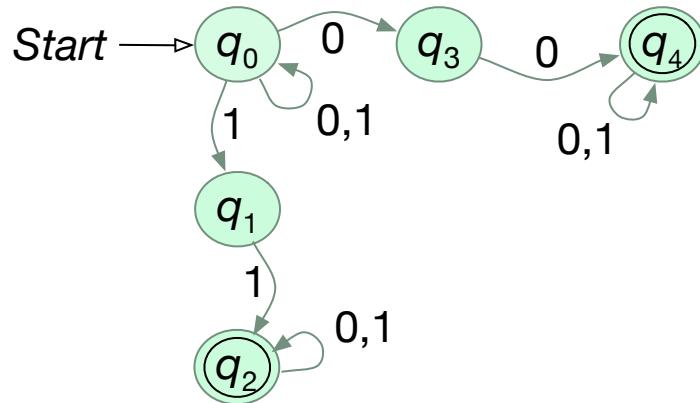
- ◆ **Definicija.** A nedeterministični končni avtomat (NKA) je peterka $(Q, \Sigma, \delta, q_0, F)$, kjer so:

- ◆ Q končna množica **stanj**,
- ◆ Σ (končna) **vhodna abeceda**,
- ◆ $q_0 \in Q$ **začetno stanje**,
- ◆ $F \subseteq Q$ množic **končnih stanj**, in ¶
- ◆ δ **funkcija prehodov** tj. $\delta : Q \times \Sigma \rightarrow 2^Q$

Torej so v množici $\delta(q, a)$ natanko tista stanja, v katera NKA lahko preide iz stanja q , če je prebral simbol a .

- ◆ *Opomba:* δ je *program* NKA. Vsak NKA ima svoj specifični program δ .

Primer.



- Zgoraj je diagram prehodov za NKA $M = (Q, \Sigma, \delta, q_0, F)$, definiran takole:

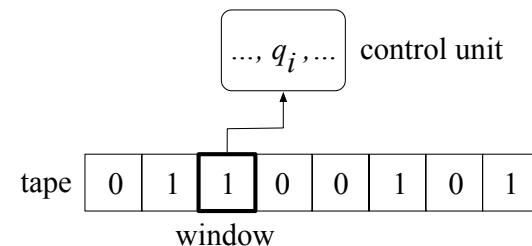
$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\Sigma = \{0, 1\}$$

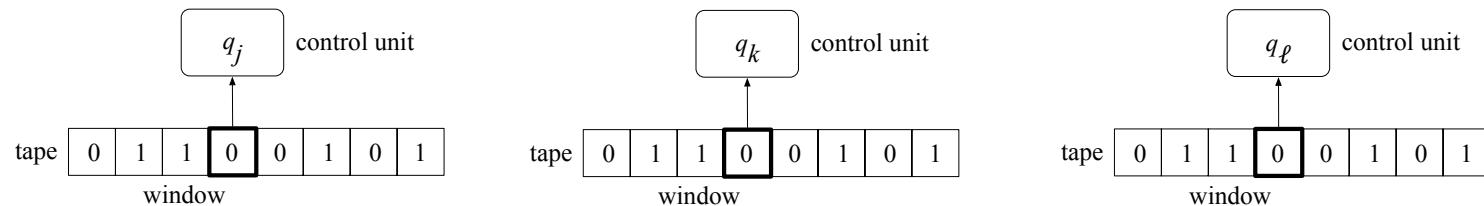
$$F = \{q_2, q_4\}$$

δ	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	\emptyset	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$

- NKA si lahko **predstavljamo** podobno kot DKA. Tudi NKA bere s traku. Magično uganjevanje nadzorne enote pa nadomestimo s vzporednim (paralelnim) delovanjem. Spodaj je narisana konfiguracija, kjer NKA v stanju q_i vidi v oknu symbol 1.



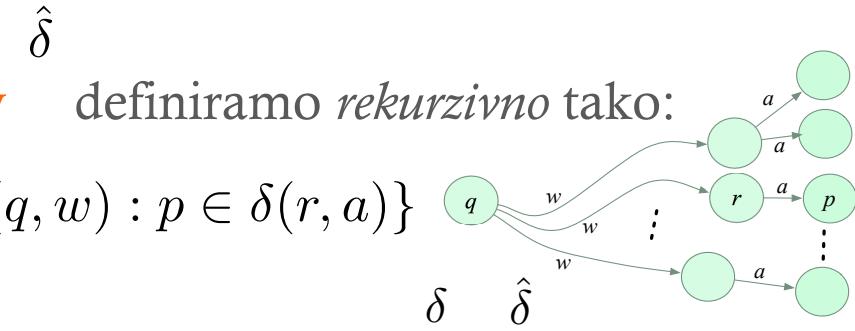
- Ker mora NKA *izbrati* naslednje stanje, si mislimo, da se za vsako *možno naslednje stanje* ustvari kopija NKA z nadzorno enoto v tem možnem stanju in oknom na naslednji celici. Npr., če je v programu NKA ukaz $\delta(q_i, 1) = \{q_j, q_k, q_\ell\}$, nastanejo 3 kopije (konfiguracije):



- Nato na podoben način vsaka kopija nadaljuje svoje izvajanje neodvisno od ostalih. Tako zamišljeno vzporedno računanje lahko predstavimo z **drevesom izvajanja**.

- Spet je koristno (nasl. prosojnjica), če razširimo δ tako, da je lahko njen drugi argument niz simbolov (ne le en simbol).
- Iščemo funkcijo $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$, da bo $\hat{\delta}(q, x)$ množica stanj, kamor NKA lahko pride iz stanja q , ko prebere besedo x .
Torej je $\hat{\delta}(q, x)$ množica stanj, do katerih iz q vodijo poti, označene z x .

- Razširjeno funkcijo prehodov definiramo rekurzivno tako:
 - $\hat{\delta}(q, \varepsilon) = \{q\}$
 - $\hat{\delta}(q, wa) = \{p \in Q \mid \exists r \in \hat{\delta}(q, w) : p \in \delta(r, a)\}$
 - $\hat{\delta}(q, a) = \delta(q, a)$
- Ker je δ (dokaži!) ni neujemanja med δ in $\hat{\delta}$.
Zato zaradi enostavnosti običajno pišemo kar namesto in si zapomnimo, da zdaj deluje nad nizi.
- Koristno je $\delta(S, x) = \bigcup_{q \in S} \hat{\delta}(q, x)$ razširiti tako, da je njen prvi argument lahko množica stanj:



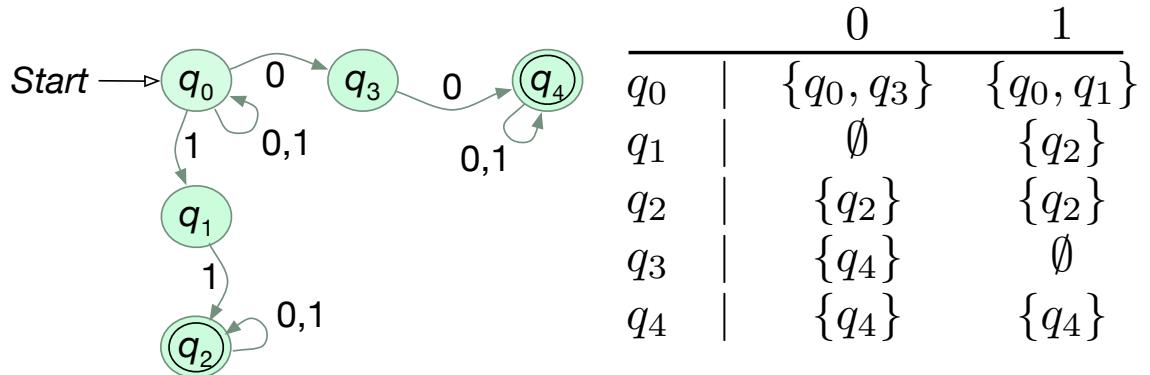
♦ Definicije.

- ♦ NKA $M = (Q, \Sigma, \delta, q_0, F)$ sprejme besedo (niz) x , če $\delta(q_0, x)$ vsebuje kako končno stanje $p \in F$ (tj. če $\delta(q_0, x) \cap F \neq \emptyset$).
- ♦ Jezik, sprejet z NKA M je množica $L(M)$ vseh besed, ki jih sprejme M :

$$L(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \text{ vsebuje neko končno stanje iz } F\}$$

- ♦ Kakšen je $L(M)$? Če bi bil M determinističen, torej DKA, bi bil $L(M)$ regularen jezik. Toda zdaj je M nedeterminističen, tj. NKA. Intuicija nam pravi, da je NKA močnejši od DKA (zaradi svoje magične sposobnosti) in zato sprejme več besed kot DKA.
Torej pričakujemo, da $L(M)$ ni regularen. (Toda videli bomo, da nas intuicija varata.)

Primer (nadaljevanje)



- Ali je beseda $x = 01001$ v jeziku $L(M)$?
 - Ugotoviti moramo, ali množica $\delta(q_0, x) = \delta(q_0, 01001)$ vsebuje kako končno stanje:

$$\begin{aligned}
 \delta(q_0, 01001) &= \delta(\{q_0, q_3\}, 1001) \\
 &= \delta(\{q_0, q_1\} \cup \emptyset, 001) = \delta(\{q_0, q_1\}, 001) \\
 &= \delta(\{q_0, q_3\} \cup \emptyset, 01) = \delta(\{q_0, q_3\}, 01) \\
 &= \delta(\{q_0, q_3\} \cup \{q_4\}, 1) = \delta(\{q_0, q_3, q_4\}, 1) \\
 &= \delta(\{q_0, q_1\} \cup \emptyset \cup \{q_4\}, \varepsilon) = \delta(\{q_0, q_1, q_4\}, \varepsilon), \text{ and } q_4 \in F
 \end{aligned}$$

2.4 Ekvivalentnost DFA in NFA

- Vsak DKA je tudi NKA.

Zakaj? Na DKA lahko gledamo kot na *posebni* NKA, pri katerem je funkcija prehodov δ taka, da velja $\delta(q, a) = \{r\}$, za vsak par $q \in Q$, $a \in \Sigma$ in neki $r \in Q$. Pri takem NKA *vedno* obstaja *natanko eno* možno naslednje stanje.

- Zato razred jezikov, sprejetih z NKA, *vsebuje vse jezike, sprejete z DKA* (tj. *regularne jezike*):

Razred jezikov sprejetih z NKA

Razred jezikov sprejetih z DKA

?

Vprašanje: Ali obstajajo tu kaki jeziki?

- ◆ **Vprašanje:** Ali obstaja *neregularni* jezik, ki ga sprejme NKA?
Odgovor: Ne! Razred jezikov, sprejetih z NKA, je enak razredu jezikov, sprejetih z DKA!
- ◆ **Vprašanje:** Kako to ugotovimo?
Odgovor: Dokažemo, da ima vsak NKA *ekvivalentni* DKA (tj. DKA, ki sprejme *isti* jezik kot NKA).
- ◆ **Izrek.** Naj bo L jezik, ki ga sprejme dani NKA M .
Potem obstaja DKA M' , ki sprejme L .
- ◆ **Ideja dokaza.** Konstruiramo DKA M' , ki je sposoben *simulirati* NKA M . Kako? Vsako stanje DKA M' bo predstavljalo neko *množico stanj* NKA M . Nadzorna enota DKA M' bo *beležila* vsa stanja, v katerih bi lahko bil NKA M , če bi prebral vhodno besedo, ki jo je dotlej prebral M' .

► Dokaz.

- Naj bo $M = (Q, \Sigma, \delta, q_0, F)$ NKA, ki sprejme jezik L . (Torej $L = L(M)$.)
- Definirajmo DKA $M' = (Q', \Sigma', \delta', q'_0, F')$ takole:
 - $Q' = 2^Q$. Stanje DKA M' naj predstavlja množico vseh stanj, v katerih bi bil lahko NKA M v tistem trenutku.
 - Notacija: Stanje DKA M' označimo s $[q_{i_1}, \dots, q_{i_k}]$ (kjer $q_{i_1}, \dots, q_{i_k} \in Q$ so stanja M). Torej stanje $[q_{i_1}, \dots, q_{i_k}]$ DKA M' predstavlja množico $\{q_{i_1}, \dots, q_{i_k}\}$ stanj NKA M .
 - $\Sigma' = \Sigma$ (DKA M' naj ima enako abecedo kot NKA M)
 - $q'_0 = [q_0]$ (Začetno stanje DKA M' mora biti $[q_0]$, ker je takrat NKA M v stanju q_0)
 - $\delta'([q_{i_1}, \dots, q_{i_k}], a) = [p_{j_1}, \dots, p_{j_\ell}]$ iff $\delta(\{q_{i_1}, \dots, q_{i_k}\}, a) = \{p_{j_1}, \dots, p_{j_\ell}\}$
Stanje, kamor preide M' iz stanja $[q_{i_1}, \dots, q_{i_k}]$ pri simbolu a , dobimo tako: (1) uporabimo δ nad vsakim stanjem v $\{q_{i_1}, \dots, q_{i_k}\}$ in (2) izračunamo unijo dobljenih množic. Če je unija $\{p_{j_1}, \dots, p_{j_\ell}\}$, je $[p_{j_1}, \dots, p_{j_\ell}]$ stanje, kamor preide M' .
 - F' naj bo množica stanj DKA M' , ki imajo v svojem zapisu vsaj eno končno stanje NKA M .

- ◆ **Dokaz** (nadaljevanje)

- ◆ Nato moramo pokazati, da velja

$$\delta'(q'_0, x) = [q_{i_1}, \dots, q_{i_k}] \text{ iff } \delta(q_0, x) = \{q_{i_1}, \dots, q_{i_k}\}, \text{ za poljuben } x \in \Sigma^*.$$

- ◆ To storimo z matematično indukcijo po dolžini $|x|$ vhodne besede x . (**Vaja.**)

- ◆ Obrazložimo, da velja tudi

$$\delta'(q'_0, x) \in F' \text{ iff } \delta(q_0, x) \text{ vsebuje končno stanje iz } F$$

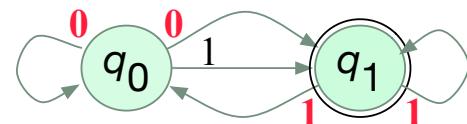
- ◆ Potem lahko zaključimo z ugotovitvijo, da je $L(M) = L(M')$. □

Dokazali smo, da DKA M' sprejme *isti* jezik kot NKA M . V tem smislu sta M' in M *ekvivalentna* (enako “močna”). Seveda pa se razlikujeta po številu svojih stanj in po ukazih v svojih programih.

Primer.

- Naj bo $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ NKA, kjer

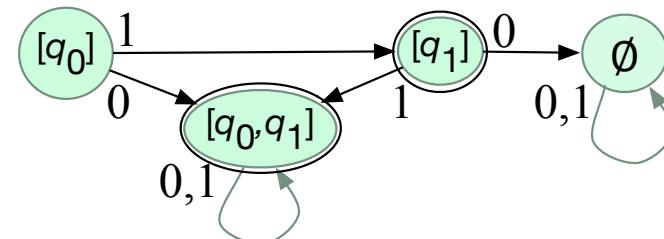
$$\begin{array}{ll} \delta(q_0, 0) = \{q_0, q_1\} & \delta(q_1, 0) = \emptyset \\ \delta(q_0, 1) = \{q_1\} & \delta(q_1, 1) = \{q_0, q_1\} \end{array}$$



- DKA $M' = (Q', \{0, 1\}, \delta', [q_0], F')$, ki sprejme $L(M)$, je:

$$\begin{array}{llll} \bullet & Q' = 2^Q = \{\emptyset, [q_0], [q_1], [q_0, q_1]\} & (\text{vse podmnožice množice } Q = \{q_0, q_1\}) & \\ \bullet & \delta'(\emptyset, 0) = \emptyset & \delta'([q_0], 0) = [q_0, q_1] & \delta'([q_1], 0) = \emptyset & \delta'([q_0, q_1], 0) = [q_0, q_1] \\ \bullet & \delta'(\emptyset, 1) = \emptyset & \delta'([q_0], 1) = [q_1] & \delta'([q_1], 1) = [q_0, q_1] & \delta'([q_0, q_1], 1) = [q_0, q_1] \end{array}$$

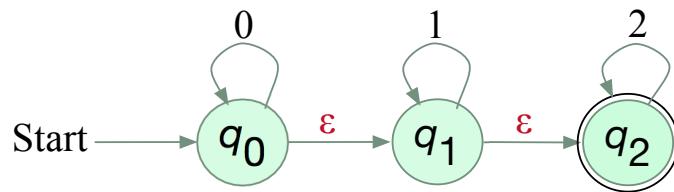
$$\bullet \quad F' = \{[q_1], [q_0, q_1]\}$$



2.5 Končni avtomati s tihimi prehodi

- Definicijo NKA lahko še razširimo tako, da ji dodamo sposobnost *spontanega prehajanja* med stanji, tj. prehajanja v nova stanja, ne da bi NKA pri tem prebral simbol v oknu. To je **tiki prehod (ϵ -prehod)**. Diagram takega NKA ima lahko na povezavah označke ϵ .

Primer.



- Ta NKA sprejme besede, ki se začnejo s poljubnim številom simbolov 0 (tudi ϵ), nadaljujejo s poljubnim številom simbolov 1 (tudi ϵ), in končajo s poljubnom številom simbolov 2 (tudi ϵ). Zakaj?
- Odgovor: NKA sprejme besedo x , če obstaja pot, označena z x iz q_0 v q_2 . Toda povezave, označene z ϵ , so lahko sestavni del te poti, čeprav ϵ ni eksplicitno naveden v x . (Npr. $x = ab = \epsilon a \epsilon b \epsilon$.)
- Npr., $x = 002$ je sprejeta, ker je pot $q_0 \rightarrow q_0 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_2$ označena z $00\epsilon\epsilon2$, hkrati pa vemo, da je $00\epsilon\epsilon2 = 002$.

◆ **Definicija.** NKA s tihimi prehodi (NKA_ε)

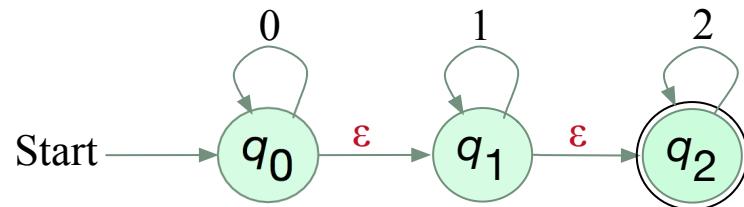
je peterka $(Q, \Sigma, \delta, q_0, F)$, kjer so:

- ◆ Q končna množica stanj,
- ◆ Σ vhodna abeceda,
- ◆ $q_0 \in Q$ začetno stanje,
- ◆ $F \subseteq Q$ množica končnih stanj, ¶
- ◆ δ **funkcija prehodov**, tj. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$

V množici $\delta(q, a)$ so natanko tista stanja, kamor NKA lahko preide iz stanja q po povezavah, ki so označene z a ali ε .

- ◆ *Opomba:* δ je program NKA_ε . Vsak NKA_ε ima svoj specifični program δ .

► **Primer (nadaljevanje)** NKA $_{\epsilon}$, ki ustreza diagramu prehodov

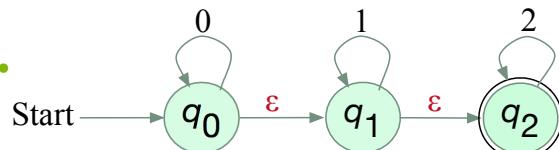


je $(Q, \Sigma, \delta, q_0, F)$, kjer so

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1, 2\}$
- $F = \{q_2\}$

		0	1	2	ϵ
►	q_0	$\{q_0\}$	\emptyset	\emptyset	$\{q_1\}$
►	q_1	\emptyset	$\{q_1\}$	\emptyset	$\{q_2\}$
►	q_2	\emptyset	\emptyset	$\{q_2\}$	\emptyset

- Spet želimo **razširiti** funkcijo δ avtomata NKA $_{\varepsilon}$ tako, da bo lahko njen drugi argument *niz simbolov* (morda celo prazen).
- Definirati želimo funkcijo $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ tako, da bo $\hat{\delta}(q, x)$ množica stanj p , do katerih iz q vodijo poti označene z x , na katerih so lahko povezave označene tudi z ε .
 - Definicija funkcije $\hat{\delta}$ bo jasnejša, če prej definiramo še množico vseh stanj, dosegljivih iz stanja q samo s tihimi prehodi : **ε -Closure(q)**.
- Primer.**



$$\varepsilon\text{-Closure}(q_0) = \{q_0, q_1, q_2\},$$

$$\varepsilon\text{-Closure}(q_1) = \{q_1, q_2\},$$

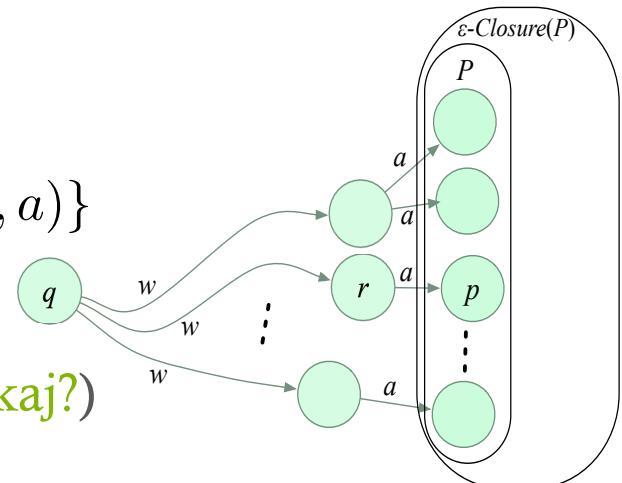
$$\varepsilon\text{-Closure}(q_2) = \{q_2\}.$$

- To razširimo še na množice: **ε -Closure(S)** = $\bigcup_{q \in S} \varepsilon\text{-Closure}(q)$

- ◆ Razširjeno funkcijo prehodov $\hat{\delta}$ definiramo rekurzivno:
 - ◆ $\hat{\delta}(q, \varepsilon) = \varepsilon\text{-Closure}(q)$
 - ◆ Pri $w \in \Sigma^*$ in $a \in \Sigma$ je
 $\hat{\delta}(q, wa) = \varepsilon\text{-Closure}(P)$
 kjer je $P = \{p \mid \exists r \in \hat{\delta}(q, w) : p \in \delta(r, a)\}$
- ◆ Zdaj je v splošnem $\hat{\delta}(q, a) \neq \delta(q, a)$. (Zakaj?)
- ◆ δ in $\hat{\delta}$ razširimo še na množice stanj: če je R množica stanj, potem sta

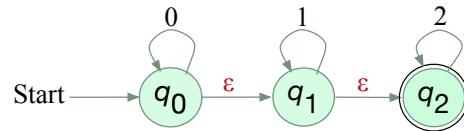
$$\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$$

$$\hat{\delta}(R, x) = \bigcup_{q \in R} \hat{\delta}(q, x)$$



► **Primer (nadaljevanje).**

NKA ε z diagramom prehodov (na levi) ima



	0	1	2	ϵ
q_0	{ q_0 }	\emptyset	\emptyset	{ q_1 }
q_1	\emptyset	{ q_1 }	\emptyset	{ q_2 }
q_2	\emptyset	\emptyset	{ q_2 }	\emptyset

Naj bo $x = 01$ vhodna beseda. Koliko je $\hat{\delta}(q_0, 01)$? (V katerih stanjih je lahko, ko jo prebere?)

- $\hat{\delta}(q_0, \varepsilon) = \varepsilon\text{-Closure}(q_0) = \{q_0, q_1, q_2\}$
- $\begin{aligned} \hat{\delta}(q_0, 0) &= \varepsilon\text{-Closure}(\delta(\hat{\delta}(q_0, \varepsilon), 0)) = \varepsilon\text{-Closure}(\delta(\{q_0, q_1, q_2\}, 0)) \\ &= \varepsilon\text{-Closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) = \varepsilon\text{-Closure}(\{q_0\} \cup \emptyset \cup \emptyset) \\ &= \varepsilon\text{-Closure}(\{q_0\}) = \{q_0, q_1, q_2\} \end{aligned}$
- $\begin{aligned} \hat{\delta}(q_0, 01) &= \varepsilon\text{-Closure}(\delta(\hat{\delta}(q_0, 0), 1)) = \varepsilon\text{-Closure}(\delta(\{q_0, q_1, q_2\}, 1)) \\ &= \varepsilon\text{-Closure}(\{q_1\}) = \{q_1, q_2\} \end{aligned}$

♦ Definicije.

- ♦ NKA_ε $M = (Q, \Sigma, \delta, q_0, F)$ sprejme besedo (niz) x , če $\hat{\delta}(q_0, x)$ vsebuje kako končno stanje $p \in F$.
- ♦ Jezik, sprejet z NKA_ε $M = (Q, \Sigma, \delta, q_0, F)$ je množica $L(M)$ vseh besed, ki jih sprejme M , torej

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \text{ vsebuje neko končno stanje iz } F\}$$

2.6 Ekvivalentnost NKA in NKA_ϵ

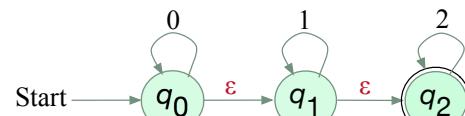
- ◆ *Intuicija* nam pravi, da je NKA_ϵ močnejši (sposobnejši) od NKA, saj lahko tiho prehaja med stanji (česar NKA ne more).
- ◆ Če nas intuicija ne vara, bi moral biti model NKA_ϵ sposoben sprejemati tudi jezike, ki *niso regularni*.

- ◆ **Vprašanje:** Ali obstaja *neregularni* jezik, ki ga sprejme NKA $_{\varepsilon}$?
- Odgovor:** Ne! Razred jezikov, sprejetih z NKA $_{\varepsilon}$, je *enak* razredu jezikov, sprejetih z NKA!
- ◆ **Vprašanje:** Kako to ugotovimo?
- Odgovor:** Dokažemo, da ima vsak NKA $_{\varepsilon}$ *ekvivalentni* NKA (tj. NKA, ki sprejme *isti* jezik kot NKA $_{\varepsilon}$).
- ◆ **Izrek.** Naj bo L jezik, ki ga sprejme dani NKA $_{\varepsilon} M$.
Potem obstaja NKA M' , ki sprejme L .
- ◆ **Ideja dokaza.** Dokazati želimo, da bo M' lahko *simuliral* potezo avtomata M za *vsak* par (q, a) . Ker pa M lahko izvaja tudi *tihe* prehode, bo moral biti M' sposoben preiti iz stanja q v stanje p , če v diagramu prehodov avtomata M obstaja pot iz q v p , označena z a , na kateri so lahko tudi ε -prehodi. Torej bomo morali poiskati tako funkcijo prehodov δ' avtomata M' , da bo veljalo $\delta'(q, a) = \hat{\delta}(q, a)$.

◆ Dokaz.

- ◆ Naj bo $M = (Q, \Sigma, \delta, q_0, F)$ NKA $_{\varepsilon}$, ki sprejme jezik L .
- ◆ Definirajmo NKA $M' = (Q, \Sigma, \delta', q_0, F')$ takole:
 - ◆ $\delta' = \hat{\delta}$, torej naj bo $\delta'(q, a) = \hat{\delta}(q, a)$ za vsak par $q \in Q$ in $a \in \Sigma$.
 - ◆ $F' = \begin{cases} F \cup \{q_0\} & \text{if } \varepsilon\text{-Closure}(q_0) \text{ contains a state in } F, \\ F & \text{otherwise.} \end{cases}$
- ◆ Pozor: M' nima ε -prehodov (torej je NKA), zato smo njegovo funkcijo prehodov označili z δ' (ne pa z $\hat{\delta}'$). Razlikovati pa moramo δ in $\hat{\delta}$ (ker obe opisujeta NFA $_{\varepsilon}$).
- ◆ Lema. $\delta'(q_0, x) = \hat{\delta}(q_0, x)$ za $|x| \geq 1$.
Dokaz: Indukcija po $|x|$. (Vaja.) □
- ◆ Nato dokažemo še naslednje: $\delta'(q_0, x)$ vsebuje stanje iz F' natanko tedaj, ko $\hat{\delta}(q_0, x)$ vsebuje stanje iz F . Dokaz. (Vaja.) □
□

♣ **Primer (nadaljevanje).** NKA _{ϵ} M z diagramom prehodov



ima: $Q = \{q_0, q_1, q_2\}$ $\delta =$
 $F = \{q_2\}$

	0	1	2	ϵ
q_0	$\{q_0\}$	\emptyset	\emptyset	$\{q_1\}$
q_1	\emptyset	$\{q_1\}$	\emptyset	$\{q_2\}$
q_2	\emptyset	\emptyset	$\{q_2\}$	\emptyset

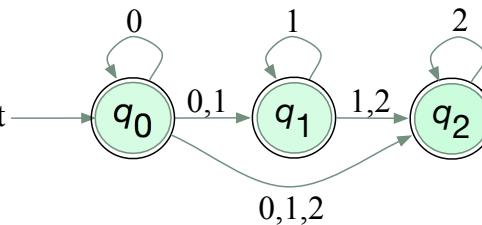
Ekvivalentni NKA je $M' = (Q, \Sigma, \delta', q_0, F')$, kjer so:

♣ $\delta'(q, a) = \hat{\delta}(q, a) =$

	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	\emptyset	$\{q_1, q_2\}$	$\{q_2\}$
q_2	\emptyset	\emptyset	$\{q_2\}$

♣ $F' = \{q_0, q_1, q_2\}$ (ker je $q_2 \in F$ dosegljivo iz vsakega stanja v Q)

Diagram prehodov za NKA M' je torej:



2.7 Regularni izrazi

- ◆ Jezike, ki jih *sprejemajo* končni avtomati (tj. regularne jezike), lahko *predstavimo* z enostavnimi izrazi, t.i. **regularnimi izrazi**.
- ◆ V nadaljevanju bomo
 - ◆ vpeljali operaciji **stik** in **zaprtje** jezikov, da bomo lahko
 - ◆ strogo definirali pojem **regularnega izraza**, in nato
 - ◆ dokazali, da je razred vseh jezikov, *sprejetih s končnimi avtomati*, *enak* razredu vseh jezikov, *predstavljenih z regularnimi izrazi*.

- ◆ **Definicija.** Naj bo Σ abeceda ter L_1 in L_2 množici besed iz Σ^* .
Stik L_1L_2 jezikov L_1 in L_2 je jezik, definiran z

$$L_1L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$$

Besede jezika L_1L_2 dobimo tako, da besedi x iz L_1 pripnemo besedo y iz L_2 , po vseh možnih x, y .

- ◆ **Definicija.** Naj bo $L \subseteq \Sigma^*$. Definirajmo $L^0 = \{\varepsilon\}$ in $L^i = LL^{i-1}$ za $i \geq 1$.

- ◆ **Kleenejevo zaprtje** L^* jezika L je jezik $L^* = \bigcup_{i=0}^{\infty} L^i$

tranzitivno zaprtje L^+ jezika L pa jezik $L^+ = \bigcup_{i=1}^{\infty} L^i$

L^* je množica besed, nastalih s stikom končno mnogo (tudi nič) besed iz L .

L^+ je množica besed, nastalih s stikom končno mnogo (toda ne nič) besed iz L .

- ◆ **Trditev:** L^+ vsebuje ε če in samo če L vsebuje ε . (Dokažite za vajo.)

♦ Primer.

Bodi $L_1 = \{10, 1\}$ in $L_2 = \{011, 11\}$.

♦ Tedaj: $L_1 L_2 = \{10, 1\} \{011, 11\} = \{10011, 1011, 111\}$. (Pozor: $1011 = 1011$.)

♦ In še: $L_1^* = \{10, 1\}^*$

$$= L_1^0 \cup L_1^1 \cup L_1^2 \cup \dots$$

$$= \{10, 1\}^0 \cup \{10, 1\}^1 \cup \{10, 1\}^2 \cup \dots$$

$$= \{\varepsilon\} \cup \{10, 1\} \cup \{1010, 101, 110, 11\} \cup \dots$$

$$= \{\varepsilon, 10, 1, 1010, 101, 110, 11, \dots\}$$

♦ In še: $L_1^+ = \{10, 1\}^+ = \{10, 1, 1010, 101, 110, 11, \dots\}$

♦ Primer.

Naj bo Σ abeceda. Σ^* je množica vseh končnih nizov simbolov iz Σ .

♦ Let $\Sigma = \{1\}$. Then $\Sigma^* = \{\varepsilon, 1, 11, 111, 1111, \dots\}$

♦ Let $\Sigma = \{0, 1\}$. Then $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \dots\}$

◆ **Definicija.** Naj bo Σ abeceda. **Regularni izraz (r.i.)** nad Σ in jezik, ki ga r.i. predstavlja, sta definirana induktivno takole:

- 1) \emptyset je r.i., ki predstavlja jezik \emptyset
- 2) ε je r.i., ki predstavlja jezik $\{\varepsilon\}$
- 3) Za vsak $a \in \Sigma$, je a r.i., ki predstavlja jezik $\{a\}$
- 4) Če sta r in s r.i. in predstavljata jezika R in S , potem:
 - a) $(r + s)$ je r.i., ki predstavlja jezik $R \cup S$ (unija jezikov R in S)
 - b) (rs) je r.i., ki predstavlja jezik RS (stik jezikov R in S)
 - c) (r^*) je r.i., ki predstavlja jezik R^* (Kleeneovo zaprtje jezika R)

Opomba. Osnovni r.i. so definirani *explicitno* (1,2,3), vsi ostali pa *induktivno* (4). Pravimo, da so take definicije *induktivne*. V nadaljevanju jih bomo še srečali. Lastnosti tako definiranih objektov lahko dokazujemo z matem. indukcijo.

Dogovori

- Veliko oklepajev v r.i. postane nepotrebnih, če se dogovorimo, da ima operacija $*$ prednost pred operacijo stik, ta pa prednost pred $+$.

Primer. $((0(1^*)) + 0)$ se poenostavi v ekvivalenten r.i. $01^* + 0$.

- Regularne izraze oblike rr^* krajše zapišemo z r^+ .
- Regularne izraz oblike $rrr\dots r$, kjer je staknjenih k r.i. r , okrajšamo z r^k .
- Običajno bomo *jasno razlikovali* med r.i. in jezikom, ki ga r.i. predstavlja. Zato bomo z $L(r)$ označili **jezik, ki ga predstavlja** r.i. r .
 - Včasih -- če ni nevarnosti za pomoto –označijo z r tudi jezik, ki ga predstavlja r . Mi se bomo temu izogibali.

♣ Primeri.

- ♦ $L(\mathbf{00}) = \{\mathbf{00}\}$; tj. jezik z eno besedo, 00.
- ♦ $L(\mathbf{0}^*) = \{\varepsilon, 0, 00, 000, \dots\}$; vsaka beseda je niz končno mnogo ničel.
- ♦ $L(\mathbf{0}^+) = \{0, 00, 000, \dots\}$; vsaka beseda je niz pozitivno mnogo ničel.
- ♦ $L(\mathbf{0}^*\mathbf{1}^*)$; vsaka beseda se začne s končno mnogo ničlami, ki jim sledi končno mnogo enic.
- ♦ $L(\mathbf{0}^+\mathbf{1}^+)$; vsaka beseda se začne s pozitivno mnogo ničlami, čemur sledi pozitivno mnogo enic.
- ♦ $L((\mathbf{0}+\mathbf{1})^*)$; vsaka beseda je sestavljena iz ničel in enic.
- ♦ $L((\mathbf{0}+\mathbf{1})^*\mathbf{1}\mathbf{1})$; vsaka beseda je sestavljena iz ničel in enic in se konča z dvema enicama.
- ♦ $L((\mathbf{0}+\mathbf{1})^*\mathbf{00}(\mathbf{0}+\mathbf{1})^*)$; vsaka beseda je sestavljena iz ničel in enic in vsebuje vsaj dve zaporedni ničli.
- ♦ $L(\mathbf{0}2^6\mathbf{1}^+)$; vsaka beseda se začne z eno ničlo, nadaljuje s šestimi dvojkami in konča s pozitivno mnogo enkami.
- ♦ $(\mathbf{1}+\mathbf{1}\mathbf{0})^*$; vsaka beseda je sestavljena iz ničel in enic, se začne z enico in ne vsebuje dveh zaporednih ničel.
(Dokaži: Z indukcijo po i dokaži, da se $(\mathbf{1}+\mathbf{1}\mathbf{0})^i$ začne z enko in ne vsebuje dveh zaporednih ničel.)
- ♦ $(\mathbf{0} + \varepsilon)(\mathbf{1}+\mathbf{1}\mathbf{0})$; vsaka beseda je sestavljena iz ničel in enic in ne vsebuje dveh zaporednih ničel.
- ♦ $\mathbf{0}^*\mathbf{1}^*\mathbf{2}^*$; vsaka beseda je niz končno mnogo ničel, ki jim sledi končno mnogo enic, tem pa končno mnogo dvojk.
- ♦ $\mathbf{00}^*\mathbf{11}^*\mathbf{22}^*$ vsaka beseda je niz pozitivno mnogo ničel, ki jim sledi pozitivno mnogo enic, tem pa sledi pozitivno mnogo dvojk. (Enakovreden zapis za $\mathbf{00}^*\mathbf{11}^*\mathbf{22}^*$ je $\mathbf{0}^+\mathbf{1}^+\mathbf{2}^+$.)

2.8 Ekvivalentnost končnih avtomatov in regularnih izrazov

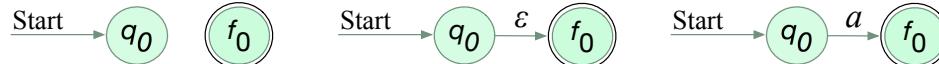
- ◆ **Dokazali bomo:** *Razred jezikov, sprejetih s končnimi avtomati, je identičen razredu jezikov, predstavljenih z regularnimi izrazi.*
- ◆ **Kako?** V dveh korakih:
 1. Za vsak r.i. r obstaja NKA $_{\varepsilon}$, ki *sprejme* jezik $L(r)$.
(Ker je model NKA $_{\varepsilon}$ ekvivalenten modelu NKA, ta pa modelu DKA, vsak od teh modelov sprejema vse jezike, ki so predstavljeni z regularnimi izrazi.)
 2. Za vsak DKA M obstaja r.i., ki *predstavlja* jezik $L(M)$.
- ◆ Vsi štirje načini (DKA, NKA, NKA $_{\varepsilon}$, r.i.) definirajo *isti razred* formalnih jezikov, t.i. *razred regularnih jezikov*.

- ◆ **Izrek.** Naj bo r poljuben regularni izraz.
Potem obstaja $\text{NKA}_\varepsilon M$, ki sprejme jezik $L(r)$.
- ◆ **Ideja dokaza.** Z *indukcijo* po številu operatorjev v r.i. bomo dokazali, da lahko za poljuben r sestavimo $\text{NKA}_\varepsilon M = (Q, \Sigma, \delta, q_0, \{f_0\})$ z *enim končnim stanjem* f_0 (in brez prehodov iz njega), da velja $L(M) = L(r)$.

Opomba. NKA_ε z enim samim končnim stanjem nam bodo omogočili *enostavno sestavljanje* manjših NKA_ε v večje. Zahteva po enem končnem stanju pa ne zmanjša splošnosti izreka. (Zakaj? **Vaja:** Pokažite, da lahko *poljuben* NKA pretvorimo v *ekvivalentni* NKA_ε z *enim* končim stanjem.)

Dokaz

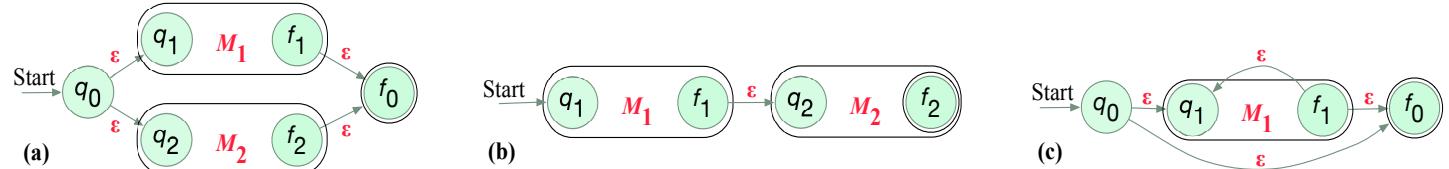
- Naj bo $P(n) \equiv$ ‘Če je r r.i. z n operatorji, potem obstaja NKA $_{\varepsilon}$ M , da je $L(M) = L(r)$.’
- Trditev $P(n)$ dokažemo z indukcijo po n .
- Osnova [preverimo $P(0)$]. Pri $n=0$ je r bodisi \emptyset , ε , ali a ($a \in \Sigma$). Ustrezni NKA $_{\varepsilon}$ so:



- Induktivna hipoteza [predpostavimo, da $P(n)$ velja za vse $n \leq k-1$ (torej je $k \geq 1$)].
- Induktivni korak [dokažemo, da potem velja $P(n)$ za vse $n \leq k$].

Naj ima r k operatorjev. Glede na obliko r obstajajo naslednje tri možnosti:

- $r = r_1 + r_2$. Vsak od r_1, r_2 ima $\leq k-1$ operatorjev. Po ind.hipotezi obstajata NKA $_{\varepsilon}$ M_1, M_2 , da je $L(M_1) = L(r_1)$ in $L(M_2) = L(r_2)$. Tedaj je NKA $_{\varepsilon}$ M , ki ustreza r , narisan na Sliki (a).
- $r = r_1 r_2$. Vsak od r_1, r_2 ima $\leq k-1$ operatorjev. Po ind.hipotezi obstajata NKA $_{\varepsilon}$ M_1, M_2 , da je $L(M_1) = L(r_1)$ in $L(M_2) = L(r_2)$. Tedaj je NKA $_{\varepsilon}$ M , ki ustreza r , narisan na Sliki (b).
- $r = r_1^*$. Zdaj r_1 vsebuje $\leq k-1$ operatorjev. Po ind.hipotezi obstaja NKA $_{\varepsilon}$ M_1 , da je $L(M_1) = L(r_1)$. Tedaj je NKA $_{\varepsilon}$ M , ki ustreza r , narisan na Sliki (c).



❖ **Primer.**

❖ Vaje.

- ◆ **Izrek.** Naj bo M poljuben DKA.
Tedaj obstaja regularen izraz r , ki predstavlja jezik $L(M)$.
- ◆ **Ideja dokaza.**
 - ◆ Jezik $L(M)$, ki ga sprejema M , zapišemo kot unijo končno mnogo množic.
 - ◆ Vsaka množica ustreza nekemu *končnemu stanju* avtomata M in vsebuje natanko tiste besede, ki privedejo M iz začetnega stanja v to končno stanje.
 - ◆ Te množice sestavimo na *induktiven* način (tj. večje izrazimo z manjšimi), za vsako pa tudi induktivno sestavljamo *regularen izraz*, ki jo predstavlja.
 - ◆ Regularni izraz, ki predstavlja jezik $L(M)$, je potem *vsota* regularnih izrazov, ki predstavljajo induktivno definirane množice.

Dokaz

- ◆ Naj bo dan DKA $M = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F)$.
- ◆ Po definiciji je $L(M) = \text{'množica besed, ki privedejo } M \text{ iz } q_1 \text{ v katerokoli končno stanje'}$.
- ◆ Pišimo $R^n_{1j} \equiv \text{"množica besed, ki privedejo } M \text{ iz } q_1 \text{ v } q_j"$. Potem $\bigcup_{q_j \in F} R^n_{1j}$ je $L(M) = \bigcup_{q_j \in F} R^n_{1j}$.
- ◆ Če bi poznali r.i. r^n_{1j} , ki predstavlja R^n_{1j} , bi $L(M)$ predstavili z r.i. $\sum_{j=1}^n r^n_{1j}$.
- ◆ Bodи $R^k_{ij} \equiv \text{"množica besed, ki privedejo } M \text{ iz } q_i \text{ v } q_j, \text{ ne da bi } M \text{ šel čez stanje } >k"$
- ◆ Opazimo: R^k_{ij} lahko sestavimo induktivno: $R^k_{ij} = R^{k-1}_{ik}(R^{k-1}_{kk})^* R^{k-1}_{kj} \cup R^{k-1}_{ij}$ (*)

$$R^0_{i,j} = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & \text{if } i = j \end{cases}$$
 (**)
- ◆ Vprašanje. Ali lahko sestavimo r.i. r^k_{ij} (ki predstavlja R^k_{ij}), ko sestavimo R^k_{ij} ?
- ◆ Odgovor. Da; konstruktivni dokaz naslednje trditve opisuje, kako to naredimo.
- ◆ Trditev: $P(k) \equiv \text{"Za poljubne } i, j, k \text{ obstaja r.i. } r^k_{ij}, \text{ ki predstavlja } R^k_{ij}"$.

Dokaz (indukcija po k).

Osnova [preverimo $P(0)$]. (**) pove, da je R^0_{ij} predstavljen z $r^0_{ij} = a_1 + \dots + a_p$ ali $r^0_{ij} = a_1 + \dots + a_p + \varepsilon$.
 Ind.hip.[denimo, da $P(k-1)$ velja]. Torej, za poljubne i, j, k obstaja r.i. r^{k-1}_{ij} , ki predstavlja R^{k-1}_{ij} .

Ind.korak [Ali potem velja $P(k-1) \Rightarrow P(k)$?]

(*) in ind.hip. Nam povesta, da je R^k_{ij} predstavljen z r.i. $r^k_{ij} = r^{k-1}_{ik}(r^{k-1}_{kk})^* r^{k-1}_{kj} + r^{k-1}_{ij}$



Primer.

- Vaje.

2.9 Uporaba končnih avtomatov

- ◆ Nekatere probleme pri razvoju programske opreme lahko rešimo tako, da pretvorimo ustreerne regularne izraze v pripadajoče DKA (njihove simulatorje).
- ◆ Nekaj takih primerov:
 - ◆ *Leksikalni analizatorji*
 - ◆ *Urejevalniki*
 - ◆ *Kompresorji podatkov*
 - ◆ Glejte še Google: *Application of Finite Automata*

◆ Leksikalni analizator.

- ◆ *Leksikalni analizator* izvede leksikalno analizo vhodnega besedila.

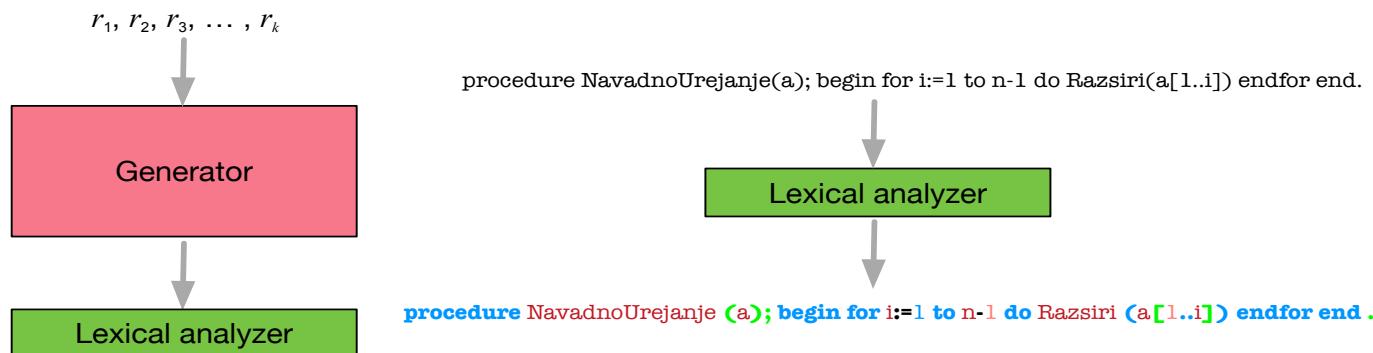
Leksikalna analiza je process, kjer se besedilo (npr. izvorna koda programa), ki je zaporedje simbolov dane abecede, preslika v zaporedje jezikovnih simbolov. *Jezikovni simbol* ima definiran pomen; ta pove, ali je beseda, ki ji jezikovni simbol pripada, ključna beseda, identifikator, num. konstanta, operator, ...

- ◆ Obliko besed, ki jih jezikovni simbol označuje, opisuje neki *regularni izraz*.

◆ Primeri.

- ◆ V nekem programirnem jeziku (ALGOL) je jezikovni simbol *identifikator* označeval vsako besedo izvornega programa, ki se je začela z malo ali veliko črko, čemur je sledil poljubno dolg niz števk ali veliki ali malih črk. Obliko vseh besed, ki so identifikatorji, je opisoval r.i. $(\text{črka})(\text{črka}+\text{števka})^*$, kjer črka = $(A+B+\dots+Z+a+b+\dots+z)$ in števka = $(0+1+\dots+9)$.
- ◆ V nekem drugem starem programirnem jeziku (FORTRAN) je bila *identifikator* vsaka beseda, ki se je začela z veliko črko, in nadaljevala z nizom kvečjemu petih velikih črk, števk ali simbola \$. Obliko teh identifikatorjev opisuje r.i. $(\text{črka})(\epsilon+\text{črka}+\text{števka})^5$, kjer črka = $(\$+A+B+\dots+Z)$.
- ◆ Vaja. Poglejte, kako je definiran identifikator v Javi, Python, C, ...

- Program, imenovan *generator leksikalnih analizatorjev* dobi na vhod zaporedje $r_1, r_2, r_3, \dots, r_k$ regularnih izrazov, kjer r_i opisuje jezikovni simbol S_i (npr. ključno besedo, identifikator, literal, numerično konstanto, operator, relacijo, ločilo, ...). Generator vrne programsko kodo *leksikalnega analizatorja*, ki je sposoben svoje vhodno besedilo (programska koda uporabnikovega programa) razdeliti v zaporedje jezikovnih simbolov.



- Kako deluje generator?

- Najprej izvede preslikavo $\{r_1, r_2, r_3, \dots, r_k\} \rightarrow \text{NKA}_\epsilon \rightarrow \text{DKA}$ (neposredno v DKA).
- Končna stanja* $q_1, q_2, q_3, \dots, q_k$ DKA ustrezajo regularnim izrazom $r_1, r_2, r_3, \dots, r_k$.
- DKA se znajde v stanju q_i , če je bila v vhodnem besedilu pravkar razpoznanata beseda, ki sodi v $L(r_i)$.
- Tej besedi zato pripredi jezikovni simbol S_i .
- Leksikalni analizator torej simulira izvajanje DKA. Leksikalni analizator je del sprednjega dela prevajalnika).

♦ Urejevalnik.

- ♦ Nekateri urejevalniki omogočajo, da uporabnik vpiše (kot parameter) poljuben regularni izraz r ; urejevalnik pa potem v uporabnikovem besedilu poišče vse besede, ki sodijo v jezik $L(r)$.
- ♦ Delovanje.
 - ♦ Urejevalnik iz vpisanega r.i. r konstruira pripadajoči DKA.
 - ♦ Simulira delovanje DKA nad uporabnikovim besedilom (= vhodno "besedo" DFA).

- ❖ Stiskanje (kompresija) podatkov.
 - ❖
 - ❖ Primeri.
 - ❖ ...

2.10 Slovar

token jezikovni simbol finite automaton končni avtomat regular expression regularni izraz finite state system končni sistem state stanje switching circuit preklopno vezje Turing machine Turingov stroj deterministic finite automaton deterministični končni avtomat state transition prehod stanja input symbol vhodni simbol input alphabet vhodna abeceda initial state začetno stanje final state končno stanje accepting state sprejemajoče stanje to accept sprejeti transition diagram diagram prehodov transition function funkcija prehodov control unit nadzorna enota tape trak move poteza window okno extended transition function razširjena funkcija prehodov regular set regularna množica nondeterministic finite automaton nedeterministični končni avtomat execution tree drevo izvajanja ϵ -move tihi prehod concatenation stik closure zaprtje Kleene closure Kleenovo zaprtje positive closure pozitivno zaprtje lexical analysis leksikalna analiza lexical analyzer leksikalni analizator language token jezikovni simbol lexical-analyzer generator generator leksikalnih analizatorjev text editor urejevalnik data compressor

3

Lastnosti regularnih jezikov

Vsebina

- ◆ Lema o napihovanju (za regularne jezike)
- ◆ Latnosti zaprtosti (za razred regularnih jezikov)
- ◆ Odločitveni problemi in algoritmi (za regularne jezike)
- ◆ Myhill-Nerodejev izrek in najmanjši ekvivalentni končni avtomati

- ❖ **Vprašanja o regularnih jezikih**
- ❖ O regularnih jezikih se lahko *vprašamo* marsikaj. Na primer:
 - ❖ Ali je dani jezik L (opisan na nek način) regularen?
 - ❖ Ali sta jezika $L(r_1)$, $L(r_2)$, ki ju predstavljata dana r.i. r_1 , r_2 , enaka? (tj. Ali je $L(r_1) = L(r_2)$?)
 - ❖ Končnemu avtomatu M najdi minimalni (z najmanj stanji) equivalentni končni avtomat.
 - ❖ ...
- ❖ Opisali bomo orodja za iskanje odgovorov na ta vprašanja. Orodja so:
 - ❖ *Lema o napihovanju* (koristna, če hočemo dokazati, da dani jezik *ni regularen*)
 - ❖ *Lastnosti zaprtosti* (koristne, če hočemo dokazati, da dani jezik *je regularen*)
 - ❖ *Odločitveni problemi* (koristni pri vprašanjih o regularnih izrazih in končnih avtomatih)
 - ❖ *Myhill-Nerodejev izrek* (koristen pri ugotavljanju regularnosti jezika; iskanju min. ekv. k.a.)

3.1 Lema o napihovanju za regularne jezike

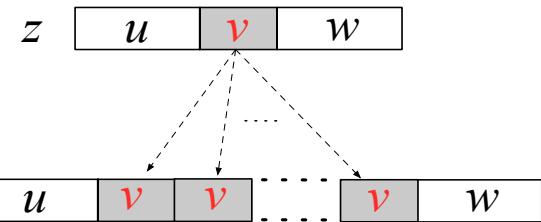
- ◆ Lema o napihovanju je močno orodje za dokazovanje
 - ◆ da nekateri jeziki *niso* regularni
 - ◆ da nekateri regularni jeziki *so/niso končni*

- ◆ **Lema o napihovanju** (za regularne jezike). Naj bo L regularni jezik. Potem obstaja konstanta n (odvisna samo od L), da velja naslednje : če je z poljubna beseda z lastnostjo

$$z \in L \wedge |z| \geq n,$$

potem obstajajo besede u, v, w , da velja

$$\begin{aligned} z &= uvw \quad \wedge \\ \wedge \quad |uv| &\leq n \quad \wedge \\ \wedge \quad |v| &\geq 1 \quad \wedge \\ \wedge \quad \forall i \geq 0: &uv^i w \in L. \end{aligned}$$



Kakšen je n ? Dokaz leme bo razkril, da je n lahko enak številu stanj v DFA, ki sprejme L .

Kakšna je beseda v ? Iz zgornjih lastnosti u, v, w enostavno sledi, da je $1 \leq |v| \leq n$. (Dokaži za vajo).

- ◆ **Intuitivno.** Če je dana poljubna zadosti dolga beseda z , ki jo sprejme neki končni avtomat, je v z blizu njenega začetka neka neprazna in ne zelo dolga podbeseda v , ki jo znotraj z lahko ponovimo poljubno (končno) mnogokrat, a bo končni avtomat napihnjeno besedo $z' = uvv\ldots vw$ tudi sprejel!
- ◆ **Formalno.** **Lema o napihovanju (za regularne jezike) pravi naslednje:** (Pozor, ta strogi zapis leme bomo rabili !)

$$L \text{ regular} \implies (\exists n)(\forall z) [z \in L \wedge |z| \geq n \Rightarrow (\exists u, v, w)[z = uvw \wedge |uv| \leq n \wedge |v| \geq 1 \wedge (\forall i \geq 0)uv^i w \in L]]$$

◆ Dokaz.

- ◆ Naj bo L regularen jezik.

Torej obstaja DKA $M = (Q, \Sigma, \delta, q_0, F)$, ki sprejme L .

Naj bo $n := |Q|$.

- ◆ Naj bo $z = a_1 \dots a_m$ ($m \geq n$) poljubna beseda iz L , ki je dolga vsaj n .
- ◆ Zaženimo M nad vhodno besedo z . Med branjem besede z avtomat M prihaja v razna stanja. Označimo s q_{ℓ_i} stanje, katerem je M , ko je prebral predpono $a_1 \dots a_i$ besede z . Ko prebere celo $z = a_1 \dots a_m$, je M do takrat vstopil v $m+1$ stanj $q_0, q_{\ell_1}, \dots, q_{\ell_m}$.
- ◆ Toda: vsaj dve od njih, npr. q_{ℓ_j} in q_{ℓ_k} , ($0 \leq j < k \leq n$), morata biti enaki, saj je $|Q| < m+1$. Torej ima pot $q_0 \rightarrow q_{\ell_1} \rightarrow \dots \rightarrow q_{\ell_m}$ cikel $q_{\ell_j} \rightarrow \dots \rightarrow q_{\ell_k}$, označen z $a_{j+1} \dots a_k$.
- ◆ Zdaj pa postavimo $u := a_1 \dots a_j$; $v := a_{j+1} \dots a_k$ in $w := a_{k+1} \dots a_m$. Potem lahko dokažemo (vaja!), da za u, v, w velja tole:
 - ◆ $z = uvw$ in
 - ◆ $|uv| \leq n$ in
 - ◆ $1 \leq |v|$ in
 - ◆ za vse $i \geq 0$, $uv^i w \in L$.



◆ Uporaba leme o napihovanju

- ◆ Lema je uporabna, če hočemo dokazati, da neki jezik *ni regularen*. *Metodo*, ki nas pri tem vodi, bomo razvili iz *formalnega* zapisa leme. Kako?

- ◆ Formalno se lema o napihovanju glasi takole:

$$L \text{ regular} \implies (\exists n)(\forall z) \left[\underbrace{z \in L \wedge |z| \geq n}_P \Rightarrow (\exists u, v, w) \left[\underbrace{z = uvw \wedge |uv| \leq n \wedge |v| \geq 1}_Q \wedge (\forall i \geq 0) uv^i w \in L \right] \right]$$

- ◆ Na bo n konstanta iz leme. Opazujmo besede z, u, v, w , pri katerih sta izjavi P in Q *resnični*. Rekli bomo, da so taki n, z, u, v, w 'dobri'. Za dobre n, z, u, v, w se zgornji izraz poenostavi v izraz

$$L \text{ regular} \implies (\forall z)(\exists u, v, w)(\forall i \geq 0) uv^i w \in L \quad (\text{where } n, z, u, v, w \text{ are 'good'})$$

- ◆ Spomnimo se iz logike: $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$ in $\neg(\forall x)F(x) \Leftrightarrow (\exists x)\neg F(x)$ ter $\neg(\exists x)F(x) \Leftrightarrow (\forall x)\neg F(x)$. Ko to uporabimo nad zgornjim izrazom (tj. $A := L$ regular; $B := (\forall z)(\exists u, v, w)(\forall i \geq 0) uv^i w \in L$), dobimo

$$(\exists z)(\forall u, v, w)(\exists i \geq 0) uv^i w \notin L \implies L \text{ ni regularen jezik} \quad (\text{tu so } n, z, u, v, w \text{ 'dobri')}$$

- ◆ Zdaj se vidi: **Če za dani L dokažemo, da leva stran ' \implies ' velja, potem L ni regularen jezik.**

To je osnova metode, po kateri se ravnamo, ko skušamo dokazati, da dani jezik ni regularen.

- ◆ **Metoda** (ko za dani jezik L želimo dokazati, da ni regularen)
- ◆ Kot smo razložili (prejšnja prosojnica), bomo **poskušali dokazati**, da za L velja tole:
 $(\exists z)(\forall u, v, w)(\exists i \geq 0)uv^iw \notin L \implies L$ ni regularen jezik (tu so n, z, u, v, w 'dobri')
 Če bomo uspeli, bo avtomatično sledilo (prejšnja prosojnica), da L *ni* regularen jezik.
- ◆ Pri dokazovanju se držimo naslednjega postopka:
 - a) Predpostavimo, da je n 'dobra' konstanta (odvisna od L) iz leme.
 - b) Izberemo 'dobro' besedo z (tj. besedo z , za katero velja $z \in L, |z| \geq n$)
 - c) Poiščemo vse možne razdelitve z v 'dobre' podbesede u, v, w (tj. take, da je $z = uvw, |uv| \leq n, |v| \geq 1$)
 - d) Poskušamo dokazati naslednjo trditev:
 za vsako 'dobro' razdelitev u, v, w
 obstaja $i \geq 0$
 za katerega je $uv^iw \notin L$.
 - e) Če nam korak d uspe, potem L *ni* regularen jezik. Če nam korak d ne uspe, je razlog za to bodisi ta, da je L v resnici regularen, ali pa ta, da njegove neregularnosti ne zmoremo dokazati s to metodo.

► Primer.

- ◆ Naj bo $L = \{0^{i^2} \mid i \in \mathbb{N}\}$. Dokazati želimo, da L ni regularen jezik.
- ◆ Uporabimo opisano metodo.
 - ◆ Naj bo n konstanta iz leme o napihovanju.
 - ◆ Vzemimo besedo $z = 0^{n^2}$. (z je ‘dobra’, ker je $z \in L$ in $|z| = n^2 \geq n$.)
 - ◆ Obstaja več možnih razdelitev besede z na ‘dobre’ u, v, w (tj. $z = uvw$, $|uv| \leq n$, $|v| \geq 1$).
 - ◆ Naj bo u, v, w poljubna ‘dobra’ razdelitev besede z . Dokazali bomo, da $uv^2w \notin L$.
 - ◆ Računamo: $|uv^2w| = |u| + 2|v| + |w| = |z| + |v| = n^2 + |v|$.
 - ◆ Vemo, da je $1 \leq |v| \leq n$. (To ste dokazali za vajo takoj po zapisu leme.)
 - ◆ Če prištejemo n^2 , dobimo $n^2 + 1 \leq |uv^2w| \leq n^2 + n$. Velja tudi $n^2 + n < (n+1)^2$, ker $n > 0$.
 - ◆ Torej je $n^2 < |uv^2w| < (n+1)^2$.
To pomeni, da $|uv^2w|$ ni popoln kvadrat; to pa pomeni, da $\underline{uv^2w \notin L}$.
 - ◆ Dokazali smo, da za poljubne ‘dobre’ u, v, w obstaja $i (=2)$, da $uv^i w \notin L$.
 - ◆ To pa po opisani metodi zadošča za sklep, da L ni regularen jezik.

S tem primerom smo spoznali, da *obstaja formalni jezik, ki ni regularen*.

To seveda pomeni, da *tega jezika ne sprejme noben končni avtomat!*

Zato bomo potrebovali *močnejši model računanja, ki bo sposoben sprejeti ta jezik!*

► Primer.

- ◆ Bodi $L = \{0^p \mid p \text{ je praštevilo}\}$. Dokazati želimo, da L ni regularen jezik.
- ◆ Uporabimo opisano metodo.
 - ◆ Naj bo n konstanta iz leme o napihovanju.
 - ◆ Vzemimo besedo $z = 0^p$, kjer je p najmanjše praštevilo, večje od n . (z je ‘dobra’.)
 - ◆ Bodi u, v, w poljubna ‘dobra’ razdelitev besede z . Dokazali bomo, da $uv^{p+1}w \notin L$.
 - ◆ Računamo: $|uv^{p+1}w| = |u| + (p+1)|v| + |w| = |z| + p|v| = p + p|v| = p(1 + |v|)$.
 - ◆ Vidimo: $p(1 + |v|)$ ni praštevilo (saj je $1 + |v| \geq 2$).
 - ◆ Torej $|uv^{p+1}w|$ ni praštevilo, in zato $uv^{p+1}w \notin L$.
 - ◆ Dokazali smo, da za poljubne ‘dobre’ u, v, w obstaja $i (=p+1)$, da $uv^i w \notin L$.
 - ◆ To po opisani metodi zadošča za sklep, da L ni regularen jezik.
- ◆ Tudi za ta jezik ne obstaja končni avtomat, ki bi ga sprejel.

3.2 Lastnosti zaprtosti za regularne jezike

- ◆ Nekatere operacije na regularnih jezikih *ohranjajo* regularnost teh jezikov (tj., delovanje teh operacij na regularnem jeziku vrne spet regularni jezik).
- ◆ *Defiramo:* razred regularnih jezikov je **zaprt za dano operacijo**, če je rezultat te operacije pri argumentih, ki so regularni jeziki, spet regularni jezik.
 - ◆ *Splošno:* Če je neki razred (ne nujno regularnih) jezikov zaprt za kako operacijo, rečemo tudi, da ima ta razred lastnost **zaprtosti** (za to operacijo)
- ◆ *Definiramo:* Zaprtost za dano operacijo je **efektivna**, če obstaja *algoritem*, ki na podlagi končnih opisov regularnih jezikov (ki so argumenti operacije) vrne *končni opis* regularnega jezika, ki je rezultat operacije. Zanimale nas bodo efektivne zaprtosti razreda regularnih jezikov.
 - ◆ Doslej smo spoznali naslednje končne opise regularnega jezika: DKA, NKA, NKA_{ϵ} in r.i.

- ◆ **Zaprtost za operacije unija, stik, Kleenejevo zaprtje.**
- ◆ **Izrek.** Razred regularnih jezikov je zaprt za operacije unija, stik in Kleenejevo zaprtje.

Intuitivno. Unija $L_1 \cup L_2$ in stik L_1L_2 poljubnih regularnih jezikov L_1, L_2 sta tudi regularna jezika, in Kleenejevo zaprtje L^* poljubnega regularnega jezika L je tudi regularni jezik.

- ◆ **Dokaz.** Izrek sledi neposredno iz definicije regularnih jezikov.
 - ◆ Naj bosta L_1, L_2 poljubna regularna jezika. Ali je jezik $L_1 \cup L_2$ tudi regularen?
Ker sta L_1, L_2 regularna, obstajata regularna izraza r_1, r_2 , za katera je $L_1 = L(r_1)$ in $L_2 = L(r_2)$.
(Če sta bila L_1, L_2 opisana z ustreznima končnima avtomatoma M_1, M_2 , lahko r_1 in r_2 konstruiramo iz M_1, M_2 .)
Iz r_1, r_2 sestavimo r.i. $r_1 + r_2$. Po definiciji ta predstavlja jezik $L_1 \cup L_2$. Zato je $L_1 \cup L_2$ regularen.
 - ◆ Podobno dokažemo zaprtost za operacije stik in Kleenovo zaprtje. (Poskusite za vajo).

□

- ◆ **Zaprtost za operaciji komplement in presek.**
- ◆ **Izrek.** Razred regularnih jezikov je zaprt za operacije komplement in presek.

Intuitivno. Komplement $\Sigma^* - L$ poljubnega regularnega jezika L je tudi regularen jezik.

Presek $L_1 \cap L_2$ poljubnih regularnih jezikov L_1, L_2 je tudi regularen jezik.

◆ Dokaz.

- ◆ (komplement) Bodи L poljuben regularen jezik. Ali je jezik $\Sigma^* - L$ tudi regularen? Ker je L regularen, obstaja DKA $M = (Q, \Sigma, \delta, q_0, F)$, da je $L = L(M)$. Iz M lahko sestavimo DKA M' , ki sprejeme $\Sigma^* - L$. Kako? M' ima komplementirano množico končnih stanj; natančneje, $M' = (Q', \Sigma, \delta', q_0', F')$, kjer so $Q' := Q$, $\Sigma' := \Sigma$, $\delta' := \delta$, $q_0' := q_0$, in $F' := Q - F$. To pomeni, da M' sprejme x natanko tedaj, ko M zavrne x . To pa pomeni, da M' sprejme jezik $\Sigma^* - L(M) = \Sigma^* - L$. Zato je $\Sigma^* - L$ regularen jezik.
- ◆ (presek) Naj bosta L_1, L_2 poljubna regularna jezika. Ali je $L_1 \cap L_2$ tudi regularen jezik? Vemo, da velja

$$L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2}$$

kjer prečne črte pomenijo operacijo komplement (glede na abecedo, ki vsebuje abecedi jezikov L_1, L_2). Vemo že, da je razred regularnih jezikov zaprt za operacije komplement in unija. Iz tega in iz zgornje enačbe sledi, da je zaprt tudi za operacijo presek.



◆ Zaprtost za operaciji *substitucija* in *inverzni homomorfizem*.

- ◆ **Definition.** Let Σ, Δ be alphabets. A **substitution** is a function f that maps each symbol of Σ to a language over Δ ; i.e. $f(a) \subseteq \Delta^*$ for each $a \in \Sigma$. We extend f to *words* in Σ^* by defining $f(\varepsilon) = \varepsilon$ and $f(wa) = f(w)f(a)$; and then to *languages* by defining $f(L) = \bigcup_{x \in L} f(x)$

◆ **Question.** The definition of substitution says nothing about the kind of the set L and the sets $f(a)$, $a \in \Sigma$. What if we additionally require that L and all $f(a)$, $a \in \Sigma$ are *regular*? Is then $f(L)$ regular too?

◆ **Example.** Let $\Sigma = \{0,1\}$, $\Delta = \{a,b\}$ and f a substitution defined by $f(0) = a$, $f(1) = b^*$. Both $f(a), f(b)$ are *regular*. Let $x = 010$. Then $f(x) = f(010) = f(0)f(1)f(0) = ab^*a$.

Let L be regular set denoted by $0^*(0+1)1^*$; then $f(L) = a^*(a+b^*)(b^*)^*$. This is a regular set. (Prove.)

- ◆ **Definition.** A **homomorphism** is a substitution h such that $h(a)$ contains a *single word* for each $a \in \Sigma$. We extend h to words and languages as in the case of the substitution. The **inverse homomorphic image** of a word w is the set $h^{-1}(w) = \{x \mid h(x) = w\}$ and of a language L is the set $h^{-1}(L) = \{x \mid h(x) \in L\}$.

◆ **Example.** Let h be a homomorphism defined by $h(0) = aa$ and $h(1) = aba$.

Let $x = 010$. Then $h(x) = h(010) = aaabaaa$; and $h^{-1}(aaabaaa) = \{010\}$. (Why? Only 010 maps to aaabaaa.)

Let $L_1 = (01)^*$. Then $h(L_1) = (aaaba)^*$. Let $L_2 = (ab+ba)^*a$. Then $h^{-1}(L_2) = \{x \mid h(x) \in (ab+ba)^*a\} = \{1\}$. (Why?)

- ◆ **Theorem.** The class of regular sets is closed under substitution, homomorphism and inverse homomorphism.

Remark. Let f be a substitution and h a homomorphism. The theorem states that if L and all $f(a)$ are regular, then also $f(L)$ is regular; and if L is regular, $h(L)$ and $h^{-1}(L)$ are regular too.

- ◆ **Proof idea.**

- ◆ (*substitution*) Let L and all $f(a)$, $a \in \Sigma$ be regular sets. Let L be denoted by r.e. r and $f(a)$ by r_a .
Idea: replace each occurrence of a in r by r_a . Then prove that the resulting r.e. r' denotes $f(L)$.
(Use induction on the number of operators in r' .)
- ◆ (*homomorphism*) Closure under homomorphism follows directly from closure under substitution
(because every homomorphism is by definition a (special) substitution).
- ◆ (*inverse homomorphism*) Let L be regular and h a homomorphism. We want to prove that $h^{-1}(L)$ is regular. Let M be DFA accepting L . We want to construct a DFA M' such that M' accepts $h^{-1}(L)$ iff M accepts L . *Idea:* construct M' so that when M' reads $a \in \Delta$, it simulates M on $h^{-1}(L)$.

□

- ◆ Homomorphisms and inverse homomorphisms often simplify proofs.

- ◆ **Zaprtost za operacijo kvocient.**
- ◆ **Definicija.** Kvocient jezikov L_1 in L_2 je jezik L_1/L_2 , definiran takole:

$$L_1/L_2 = \{x \mid \exists y \in L_2 : xy \in L_1\}.$$

Intuitivno. L_1/L_2 vsebuje predpone tistih besed jezika L_1 , katerih pripadajoče pripone so besede v jeziku L_2 .

 - ◆ **Vaja.** Dokaži ali ovrži: $(L_1/L_2)L_2 = L_1$.
 - ◆ **Primer.** To do.
 - ◆ **Vprašanje.** Definitcija L_1/L_2 ne predpostavlja ničesar glede jezikov L_1 , L_2 .
Kaj če sta L_1 , L_2 regularna jezika? Ali je tedaj regularen tudi kvocient L_1/L_2 ?
Kaj če je deljenec L_1 regularen, delitelj L_2 pa poljuben? Kakšen je tedaj kvocient L_1/L_2 ?
 Naslednji izrek odgovarja na obe vprašanji.
- ◆ **Izrek.** Razred regularnih jezikov je zaprt za operacijo kvocient s poljubnim jezikom (deliteljem).
- ◆ **Ideja dokaza.** Izpustimo. □

3.3 Odločitveni problemi in algoritmi za regularne jezike

- ◆ Potrebujemo **algoritme**, ki bodo odgovarjali na razna vprašanja o regularnih jezikih. Primeri takih vprašanj so:
 - ◆ Ali je dani regularni jezik L *neprazen* ?
 - ◆ Ali je dani regularni jezik L *neskončen* ?
 - ◆ Ali sta dva dana končna avtomata M_1 in M_2 *ekvivalentna* ?
- ◆ Zgornja vprašanja sprašujejo po odgovoru, ki je bodisi DA ali NE. Računski problemi, ki sprašujejo po odgovoru DA/NE, so **odločitveni problemi**, algoritmi, ki rešujejo odločitvene probleme, pa **odločitveni algoritmi**.
- ◆ Vhodni podatki odločitvenih algoritmov bodo *končni opisi* regularnih jezikov, tj. DKA, NKA, NKA_ε ali r.i. (Ker znamo efektivno pretvoriti en opis v drugega, lahko za vhod algoritma izberamo tistega, ki se nam zdi najustreznejši.)

◆ Praznот in končnost regularnih jezikov.

Odločitvena algoritma za odločitvena problema “Ali je regularni jezik L neprazen?” ter “Ali je regularni jezik L neskončen?” sta zasnovana na naslednjem izreku.

- ◆ **Izrek.** Jezik $L(M)$, ki ga sprejme končni avtomat M z n stanji, je:
 - 1) neprazen iff M sprejme neko besedo dolžine ℓ , kjer je $\ell < n$.
 - 2) neskončen iff M sprejme neko besedo dolžine ℓ , kjer je $n \leq \ell < 2n$.

- ◆ **Odločitvena algoritma.** Odločitvena algoritma za zgornja problema sta enostavna:
 - ◆ „Ali je $L(M)$ neprazen?“ : ”Preveri, ali v $L(M)$ obstaja beseda dolžine $\ell < n$.“
 - ◆ ”Ali je $L(M)$ neskončen?“ : ”Preveri, ali v $L(M)$ obstaja beseda dolžine $n \leq \ell < 2n$.“

Algoritem sistematično generira besede dolžin ℓ v intervalu $[0, n-1]$ oz. $[n, 2n-1]$; po vsaki generirani besedi preveri, ali bi jo M sprejel. Če bi jo sprejel, algoritem vrne odgovor DA in se ustavi, sicer pa generira naslednjo besedo. Če ne more več generirati besed, algoritem vrne odgovor NE in se ustavi. Za poljuben vhod M se algoritma ustavita in vrneta odgovor DA ali NE. (poskusite dokazati)

Vaja. Koliko besed (v odvisnosti od n in ℓ) morata v najslabšem primeru generirati in preveriti?

◆ Ekvivalentnost končnih avtomatov.

- ◆ **Definicija.** Končna avtomata M_1, M_2 sta **ekvivalentna**, če sprejmeta isti jezik, torej če je $L(M_1) = L(M_2)$.
- ◆ **Izrek.** Obstaja algoritem, ki odloči, ali sta končna avtomata M_1 in M_2 *ekvivalentna*.
 - ◆ **Dokaz.** Naj bosta M_1 in M_2 končna avtomata in $L_1 = L(M_1)$ ter $L_2 = L(M_2)$. Definirajmo jezik L_3 takole:

$$L_3 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2).$$

L_3 je *regularen jezik* (zaradi lastnosti zaprtosti), zato ga sprejme neki končni avtomat M_3 . M_3 lahko sestavimo iz končnih avtomatov M_1 in M_2 na podlagi zgornjega izraza (**vaja**). Za M_3 pa lahko dokažemo naslednje (**vaja**):

$$M_3 \text{ sprejme besedo iff } L_1 \neq L_2.$$

Torej moramo samo še preveriti, ali je L_3 *neprazen regularen jezik* (glej prejšnjo prosojnicu).

□

3.4 Myhill-Nerodejev izrek in najmanjši ekvivalentni končni avtomati

- ◆ Naj bo L regularen jezik in M DKA, ki sprejme L . Obstaja (števno) neskončno končnih avtomatov, ki so ekvivalentni M . Ti končni avtomati se razlikujejo v komponentah Q , δ in F .
- ◆ Vprašanja:
 - ◆ Ali med končnimi avtomati, ki so ekvivalentni danemu končnemu avtomatu M , obstaja **najmanjši DKA** (tj. DKA z *najmanjšim številom stanj*)?
 - ◆ Če tak DKA obstaja, ali ga lahko (efektivno) konstruiramo iz M ?
- ◆ Odgovora na obe vprašanji sta DA. Za dokaz potrebujemo t.i. **Myhill-Nerodejev izrek**.

- Preden zapišemo *Myhill-Nerodejev izrek* moramo uvesti nekaj novih pojmov.
- Definicija.** Bodи $L \subseteq \Sigma^*$ poljuben jezik. Definirajmo **relacijo R_L** na Σ^* takole:

$$xR_Ly \text{ iff } \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L.$$

- Besedi $x, y \in \Sigma^*$ sta v relaciji R_L natanko tedaj, ko sta njuni *poljubni razširitvi* xz, yz bodisi obe v L bodisi izven L . Trditev: R_L je *ekvivalenčna* relacija (Dokaz: [vaja](#)). Torej R_L razdeli L v *ekvivalenčne razrede*. Njihovemu številu pravimo **indeks relacije R_L** . Ta je lahko *končen* ali *neskončen*. (Primer. Če za vsako besedo $x \in \Sigma^*$ velja, da ni v relaciji R_L z nobeno drugo besedo y , je indeks neskončen.)

- Definicija.** Bodи $M = (Q, \Sigma, \delta, q_0, F)$ DKA. Definirajmo **relacijo R_M** na Σ^* takole:

$$xR_My \text{ iff } \delta(q_0, x) = \delta(q_0, y).$$

- Besedi $x, y \in \Sigma^*$ sta v relaciji R_M natanko tedaj, ko privedeta M iz začetnega stanja q_0 v *isto* stanje q . Trditev: R_M je *ekvivalenčna* relacija (Dokaz: [vaja](#)). R_M razdeli Σ^* v *ekvivalenčne razrede*, po en razred za vsako stanje q , ki je dosegljivo iz q_0 . **Index relacije R_M** je *končen* (ker je Q končna množica). Jezik $L(M)$ je (očitno) unija ekvivalenčnih razredov relacije R_M , ki ustrezajo (dosegljivim) *končnim* stanjem $q \in F$. Trditev: R_M je **desno invariantna**, ker zanjo velja tole: (Dokaz: [vaja](#))

$$xR_My \Rightarrow \forall z \in \Sigma^* : xzR_Myz$$

- ◆ *Intuitivno:* Myhill-Nerodejev izrek pove, da je lastnost „biti regularen jezik L “ tesno povezana z lastnostmi relacij R_L in R_M .

- ◆ **Izrek. (Myhill-Nerode)** Naslednje izjave so *ekvivalentne*:

- 1) $L \subseteq \Sigma^*$ je *regularen jezik*.
- 2) R_L ima *končen indeks*.
- 3) L je unija (nekaterih) *ekvivalenčnih razredov* (neke) *desno invariantne ekvivalenčne relacije s končnim indeksom*.

Dokaz: *Opustimo; glej navedeno literaturo.*

- ◆ **Uporaba.** Izrek je koristen, ko za dani L uspemo dokazati eno (vseeno katero) od zgornjih izjav. Potem *takoj (brez dokaza!)* veljata tudi ostali dve izjavi -- to pa razkrije nove, dotlej nepoznane lastnosti jezika L .

- ◆ Posledica Myhill-Nerodejevega izreka je, da ima *vsak regularni jezik* („v bistvu en sam“) *najmanjši deterministični končni avtomat, ki sprejema ta jezik.*
- ◆ **Izrek. (najmanjši DKA)** Najmanjši DKA, ki sprejme dani regularni jezik L , je enolično določen do izomorfnosti (tj. do preimenovanja stanj) **natančno**.
- ◆ **Ideja dokaza.**
 - ◆ Bodи L regularen jezik. Po *Myhill-Nerodejevem izreku* ima R_L končno mnogo ekvivalentnih razredov. Označimo z $[x]$ ekv.razred, kjer je $x \in \Sigma^*$. Množica vseh ekv.razredov rel. R_L je tedaj $\{[x] \mid x \in \Sigma^*\}$.
 - ◆ Konstruirajmo DKA $M = (Q, \Sigma, \delta, q_0, F)$ na sledeči način:
 - ◆ $Q := \{[x] \mid x \in \Sigma^*\};$ (stanja predstavljajo ekv. razrede relacije R_L)
 - ◆ $\delta([x], a) := [xa],$ for $a \in \Sigma;$
 - ◆ $q_0 := [\varepsilon];$ (začetno stanje predstavlja ekv. razred $[\varepsilon]$, kjer je prazna beseda)
 - ◆ $F := \{[x] \mid x \in L\}.$ (končna stanje predstavlja ekv. razredi, kjer je sprejete besede)
 - ◆ **Opombe:** $\delta(q_0, w) = \delta(q_0, a_1a_2\dots a_n) = [a_1a_2\dots a_n] = [w].$ Torej M sprejme w natanko tedaj, ko $[w] \in F.$ To pa pomeni, da M sprejme $L.$ Iz (opuščenega) dokaza Myhill-Nerodejevega izreka pa sledi, da je tako konstruirani M najmanjši končni avtomat, ki sprejme $L.$

□

3.5 Slovar

regular set regularna množica **pumping lemma** lema o napihovanju closure property zaprtost **closed under an operation** zaprt za operacijo effective efektiven substitution substitucija homomorphism homomorfizem inverse homomorphic image inverzna homomorfna slika quotient kvocient decision problem odločitveni problem **decision algorithm** odločitveni algoritmom, odločevalnik **descriptor** opis, predstavitev **minimum state FA** najmanjši končni avtomat **right invariant relation** desno invariantna relacija

4

Kontekstno-neodvisne gramatike in jeziki

Vsebina

- ◆ Uvod
- ◆ Kontekstno-neodvisne gramatike in jeziki
 - ◆ Drevesa izpeljav
 - ◆ Poenostavitve kontekstno-neodvisnih gramatik
 - ◆ Normalna oblika Chomskega
 - ◆ Normalna oblika Greibachove
- ◆ Bistveno dvoumni kontekstno-neodvisni jeziki

4.1 Uvod

- ◆ Definirali bomo pojem **kontekstno-neodvisne gramatike (KNG)** in **kontekstno-neodvisnega jezika (KNJ)**, ki ga taka gramatika opisuje.
- ◆ KNG in KNJ so zelo pomembni v praksi. Pojavljajo se npr.
 - ◆ pri *definiranju* programskih jezikov,
 - ◆ pri formalizaciji in implementaciji *sintaksnih analizatorjev* v prevajalnikih (*parsing*)
 - ◆ omogočajo časovno *učinkovito prevajanje* programov
 - ◆ ... (glejte Google)
- ◆ **Primer.** KNG so uporabne pri opisovanju
 - ◆ *aritmetičnih izrazov* (poljubne dolžine in poljubne globine gnezdenja),
 - ◆ *bločne zgradbe programov* v razih programskih jezikih (npr. ujemanje znakov { in } v Javi). Teh vidikov (programov in programskih jezikov) *se ne da predstaviti z regularnimi izrazi*.

- ◆ Intuitivno povedano je KNG sestavljena iz končne množice **vmesnih simbolov**, med katerimi je eden **začetni simbol**, končne množice **končnih simbolov** in končne množice **produkcijskih oblik** $\dots \rightarrow \dots$. Končni simbol je nespremenljiv, vmesni simbol pa *predstavlja* jezik, ki ga KNG razvije (*generira*) iz tega simbola s produkcijskimi pravili razvijanja).

- ◆ **Primer.** KNG, ki definira zgradbo **aritmetičnih izrazov**, sestavljenih iz simbolov $+$, $*$, $($, $)$ in besede **id**, ki predstavlja numerično konstanto, ima štiri produkcijske pravile:
 - ◆ (1) $\langle \text{aritmetični izraz} \rangle \rightarrow \langle \text{aritmetični izraz} \rangle + \langle \text{aritmetični izraz} \rangle$
 - ◆ (2) $\langle \text{aritmetični izraz} \rangle \rightarrow \langle \text{aritmetični izraz} \rangle * \langle \text{aritmetični izraz} \rangle$
 - ◆ (3) $\langle \text{aritmetični izraz} \rangle \rightarrow (\langle \text{aritmetični izraz} \rangle)$
 - ◆ (4) $\langle \text{aritmetični izraz} \rangle \rightarrow \text{id}$

- ◆ Tu je en sam vmesni simbol, $\langle \text{aritmetični izraz} \rangle$; zato je to tudi začetni simbol:
- ◆ Končni simboli so $+$, $*$, $($, $)$, **id**; (**id** predstavlja poljuben numerični operand)
- ◆ Produkcijske pravile pomenijo naslednje:
 - ◆ (1) pravi, da je aritm.izraz lahko sestavljen iz dveh aritm.izrazov, med katerima je $+$;
 - ◆ (2) pravi, da je aritm.izraz lahko sestavljen iz dveh aritm.izrazov, med katerima je $*$;
 - ◆ (3) pravi, da je aritm.izraz lahko aritm.izraz, ki je med oklepajem in zaklepajem ;
 - ◆ (4) pravi, da je aritm.izraz lahko en sam operand **id**.
- ◆ Začetni simbol $\langle \text{aritmetični izraz} \rangle$ predstavlja *jezik*, v katerem so vsi aritmetični izrazi, zgrajeni s produkcijskimi 1,2,3,4.

- Z večkratno uporabo produkcij lahko **izpeljemo** iz začetnega simbola poljubno zapletene aritmetične izraze.
- S simbolom \Rightarrow bomo označili *neposredno izpeljavo*, tj. zamenjavo enega vmesnega simbola z desno stranjo kake od produkcij, ki lahko *razvijejo* (tj. imajo na levi strani) ta simbol.
- **Primer (nadaljevanje).** Izpeljimo aritmetični izraz $(\mathbf{id} + \mathbf{id}) * \mathbf{id}$ v prejšnji KNG:

$$\begin{aligned}
 <\text{aritmetični izraz}> &\Rightarrow <\text{aritmetični izraz}> * <\text{aritmetični izraz}> && \dots \text{prod. (2)} \\
 &\Rightarrow (\langle \text{aritmetični izraz} \rangle) * <\text{aritmetični izraz}> && \dots \text{prod. (3)} \\
 &\Rightarrow (\langle \text{aritmetični izraz} \rangle) * \mathbf{id} && \dots \text{prod. (4)} \\
 &\Rightarrow (\langle \text{aritmetični izraz} \rangle + \langle \text{aritmetični izraz} \rangle) * \mathbf{id} && \dots \text{prod. (1)} \\
 &\Rightarrow (\langle \text{aritmetični izraz} \rangle + \mathbf{id}) * \mathbf{id} && \dots \text{prod. (4)} \\
 &\Rightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{id} && \dots \text{prod. (4)}
 \end{aligned}$$

Končni niz (tj. niz končnih simbolov) $(\mathbf{id} + \mathbf{id}) * \mathbf{id}$ smo *izpeljali* iz začetnega simbola $\langle \text{aritmetični izraz} \rangle$. Zato je končni niz $(\mathbf{id} + \mathbf{id}) * \mathbf{id}$ v jeziku, ki ga *generira dana KNG*. Zaporedje produkcij 2,3,4,1,4,4, uporabljenih med tem razvojem, je *sled izpeljave* tega končnega niza.

4.2 Kontekstno-neodvisne gramatike in jeziki

- ◆ Definirajmo *kontekstno-neodvisno gramatiko*, KNG.
- ◆ **Definicija.** Kontekstno-neodvisna gramatika (KNG) je četverka $G = (V, T, P, S)$, kjer so:
 - ◆ V končna množica vmesnih simbolov,
 - ◆ T končna množica končnih simbolov,
 - ◆ P končna množica produkcij oblike $\dots \rightarrow \dots$, kjer je `` \dots '' na levi strani znaka `` \rightarrow '' poljuben vmesni simbol iz V in `` \dots '' na desni strani znaka `` \rightarrow '' poljubna beseda iz jezika $(V \cup T)^*$;
 - ◆ S je poseben vmesni simbol, imenovan začetni simbol.

- ◆ **Dogovor.** Za lažje razumevanje bomo simbole uporabljali takole:
 - ◆ A, B, C, D, E, \dots, S ... za vmesne simbole;
 - ◆ $a, b, c, d, e, \dots, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, **poudarjene besede** ... za končne simbole;
 - ◆ X, Y, Z ... za nekaj, kar je vmesni ali končni simbol;
 - ◆ u, v, w, x, y, z ... za nize končnih simbolov;
 - ◆ α, β, γ ... za nize, sestavljene iz končnih in vmesnih simbolov.
- ◆ Skupino produkcij $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_k$, ki lahko razvijejo vmesni simbol A , lahko okrajšamo z $A \rightarrow | \alpha_1 | \alpha_2 | \dots | \alpha_k$ ('|' pomeni 'ali').
- ◆ **Primer.** Producije iz prejšnjega primera bi lahko zapisali kot $S \rightarrow S + S \mid S * S \mid (S) \mid \text{id}$. Vmesni simbol (aritmetični izraz) (edini v tej gramatiki in zato služi tudi kot začetni simbol) smo nadomestili z začetnim simbolum S .

- Preden definiramo jezik, ki ga generira KNG $G = (V, T, P, S)$, pojasnimo še nekaj pojmov in izrazov, ki jih bomo rabili.

Definicije. Naj bo $A \rightarrow \beta$ produkcija in $\alpha, \gamma \in (V \cup T)^*$ poljubni besedi.

- Producijo $A \rightarrow \beta$ **uporabimo** na besedi $\alpha A \gamma$ tako, da v tej besedi *zamenjamo* A z β . Takrat rečemo, da smo iz $\alpha A \gamma$ **neposredno izpeljali** $\alpha \beta \gamma$ z uporabo produkcije $A \rightarrow \beta$ (oziroma, da smo $\alpha \beta \gamma$ **razvili neposredno** iz $\alpha A \gamma$ z uporabo produkcije $A \rightarrow \beta$).
- Besedi α_i in α_j sta v relaciji $\alpha_i \xrightarrow{G} \alpha_j$, če je α_j *neposredno izpeljiv* iz α_i z uporabo neke produkcije gramatike G .
- Naj bodo $\alpha_1, \alpha_2, \dots, \alpha_m \in (V \cup T)^*$, $m \geq 1$, nizi.
Če velja $\alpha_1 \xrightarrow{G} \alpha_2 \wedge \alpha_2 \xrightarrow{G} \alpha_3 \wedge \dots \wedge \alpha_{m-1} \xrightarrow{G} \alpha_m$,
potem to zapišemo z $\alpha_1 \xrightarrow{G}^* \alpha_m$ in rečemo,
da iz α_1 **izpeljemo** α_m po gramatiki G (oz. da je α_m *izpeljiv* iz α_1 po gramatiki G).

Opomba: Relacija \xrightarrow{G}^* je *refleksivno in tranzitivno zaprtje* relacije \xrightarrow{G} .

► **Definicija.** Jezik kontekstno-neodvisne gramatike $G = (V, T, P, S)$ je

$$L(G) = \{w \mid w \in T^* \wedge S \xrightarrow{G}^* w\}.$$

$L(G)$ je množica vseh končnih nizov, ki so *izpeljivi* iz začetnega simbola S po gramatiki G .

► V nadaljevanju bomo rabili še tri pojme.

Definicije.

- Jezik L je **kontekstno-neodvisen** (KNJ), če je $L = L(G)$ za neko (katerokoli) KNG G .
- Niz $\alpha \in (V \cup T)^*$ imenujemo **stavčna oblika**, če velja $S \xrightarrow{G}^* \alpha$.
- Gramatiki G_1 in G_2 sta **ekvivalentni**, če velja $L(G_1) = L(G_2)$.

Opomba. Končni niz je tudi stavčna oblika; tj., končni niz je stavčna oblika, v kateri nastopajo samo končni simboli.

- **Primer.** Dana je KNG $G = (V, T, P, S)$, kjer so

- $V = \{S\}$,
- $T = \{a, b\}$,
- $P = \{S \rightarrow aSb, S \rightarrow ab\}$.

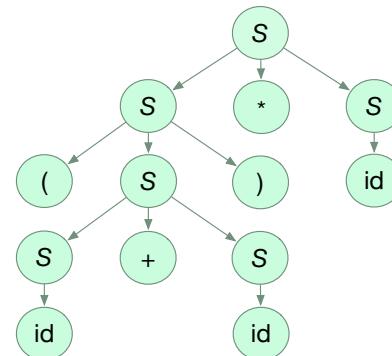
S je edini vmesni symbol (zato je začetni simbol), a, b pa končna simbola. Produciji sta $S \rightarrow aSb \mid ab$.

Kaj je $L(G)$, jezik gramatike G ?

- Če uporabimo produkcijo $S \rightarrow aSb$ $n-1$ -krat in nato produkcijo $S \rightarrow ab$, dobimo izpeljavo $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow a^3Sb^3 \Rightarrow \dots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n$. Torej iz S lahko izpeljemo končne nize oblike $a^n b^n$, $n \geq 1$. Povedano krajše: $S_G \Rightarrow^* a^n b^n$, $n \geq 1$. Dokazali smo: $\{a^n b^n \mid n \geq 1\} \subseteq L(G)$.
- *Ali lahko iz S izpeljemo še kaj drugega?* Ne. Edini končni nizi, ki se dajo izpeljati iz S , so nizi $a^n b^n$, $n \geq 1$. Zakaj? Vsakokrat, ko uporabimo produkcijo $S \rightarrow aSb$, je v novi stavčni obliki samo en vmesni simbol, po uporabi produkcije $S \rightarrow ab$ pa takega ni več. Ker obe produkciji razvijeta S , imajo vse možne sledi izpeljav končnih nizov eno samo obliko: končno število (npr. $n-1$) uporab $S \rightarrow aSb$ in nato ena uporaba $S \rightarrow ab$. To pa pomeni, da je $L(G) \subseteq \{a^n b^n \mid n \geq 1\}$.
- **Posledica:** $L(G) = \{a^n b^n \mid n \geq 1\}$.

4.3 Drevesa izpeljav

- Za boljšo ponazoritev izpeljave lahko vpeljemo pojem **drevesa izpeljav**.
Intuitivno:
 - Točke drevesa so označene z vmesnimi ali končnimi simboli (lahko tudi z ϵ).
 - Če je *notranja točka* označena z vmesnim simbolom, npr. A , potem so njeni *sinovi* (*od leve na desno*) označeni z X_1, X_2, \dots, X_k če in samo če $A \rightarrow X_1X_2\dots X_k$ je produkcija.
- Primer (nadaljevanje).** Drevo izpeljav končnega niza $(\text{id} + \text{id}) * \text{id}$ je



Možna izpeljava po gramatiki:

$$\begin{aligned} S &\Rightarrow S * S \\ &\Rightarrow (S) * S \\ &\Rightarrow (S) * \text{id} \\ &\Rightarrow (S + S) * \text{id} \\ &\Rightarrow (S + \text{id}) * \text{id} \\ &\Rightarrow (\text{id} + \text{id}) * \text{id} \end{aligned}$$

- ♣ **Natančneje:**
- ♣ **Definicija.** Bodि $G = (V, T, P, S)$ poljubna KNG. Drevo D_G je **drevo izpeljav po gramatiki** G , če velja naslednje:
 - 1) Vsaka točka $v \in D_G$ je označena z vmesnim ali končnim simbolom ($\in V \cup T \cup \{\varepsilon\}$).
 - 2) Koren drevesa D_G je označen z začetnim simbolom S .
 - 3) Če je $v \in D_G$ notranja točka, je označena z vmesnim simbolom ($\in V$).
 - 4) Če je $v \in D_G$ označena z vmesnim simbolom, npr. A , in so vsi njeni sinovi od leve v desno označeni z X_1, X_2, \dots, X_k , potem je $A \rightarrow X_1 X_2 \dots X_k$ produkcija ($\in P$).
 - 5) Če je $v \in D_G$ označena z ε , potem je v list v D_G in je edini sin svojega očeta.

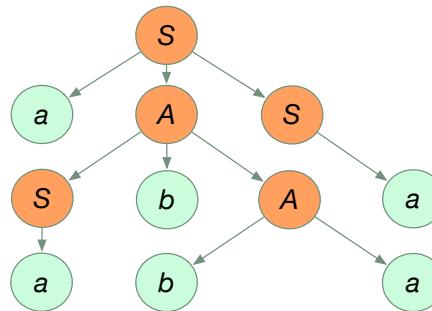
Opomba. Če je $v \in D_G$ označena s končnim simbolom ($\in T$), je v list v D_G (vendar ne nujno edinček).

- ◆ **Primer.** Dana je gramatika $G = (\{S,A\}, \{a,b\}, P, S)$, kjer so v P produkije

$$S \rightarrow aAS \mid a$$

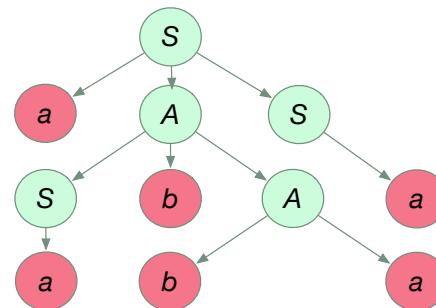
$$A \rightarrow SbA \mid SS \mid ba$$

Vprašanje: Ali je spodnje drevo drevo izpeljav po G ?



Preveriti moramo, ali to drevo izpolnjuje vse zahteve iz prejšnje definicije. Notranja vozlišča smo obarvali oranžno. Koren je označen s S ; njegovi sinovi so od leve proti desni označeni z a , A , S ; vidimo, da je $S \rightarrow aAS$ res produkcia v G . (Podobno preverimo vsako notranjo točko v .) Izkaže se, da je zgornje drevo res drevo izpeljav po gramatiki G .

- Drevesa izpeljav implicitno opisujejo vse izpeljave *stavčnih oblik* po gramatiki G .
 - **Definicija.** Oznake *listov*, izpisane med *premim obhodom* drevesa izpeljav D_G , tvorijo **rob** drevesa D_G .
 - **Primer (nadaljevanje).** Rob spodnjega drevesa izpeljav je *aabbaa* .

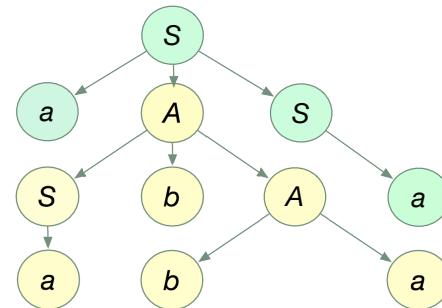


Tu je rob drevesa končni niz (sestavljen je le iz končnih simbolov). V splošnem rob drevesa D_G vsebuje tudi vmesne simbole. (**Primer:** Rob zelenega drevesa je *SAAS*.)

Opomba. Kmalu bomo dokazali: α je rob drevesa D_G če in samo če je α stavčna oblika (tj. ko velja $S \Rightarrow^* \alpha$).

- ◆ Za lažje sporazumevanje uvedimo še dva enostavna pojma.
 - ◆ **Definicija. Poddrevo** drevesa izpeljav D_G v točki ν je drevo s korenom v točki ν ter vsemi točkami in povezavami, ki v D_G sledijo točki ν . Če je ν označena z $A \in V$, rečemo, da je poddrevo v tej točki **A -drevo**.

Poddrevo ima lastnosti drevesa izpeljav, le da oznaka v njegovem korenju *ni nujno* začetni simbol S gramatike.
- ◆ **Primer (nadaljevanje).** V spodnjem drevesu izpeljav sta dve A -drevesi. Rob rumenega A -drevesa je $abba$, rob drugega A -drevesa pa ba .



- ◆ **Zveza med izpeljivostjo (po G) in robovi (v D_G)**
 - ◆ **Izrek.** Naj bo $G = (V, T, P, S)$ poljubna KNG. Tedaj velja:
 $S \xrightarrow{G}^* \alpha$ če in samo če obstaja drevo izpeljav D_G z robom α
(krajše: α ima drevo izpeljav D_G).
- Ekvivalentno:* Stavčna oblika α je *izpeljiva* iz S po G natanko tedaj, ko je α rob nekega drevesa izpeljav D_G .
- ◆ **Ideja dokaza.** Indukcija po številu notranjih točk drevesa D_G . (**Poskusite za vajo.**) □

- ◆ Skrajno leva (desna) izpeljava (po G)
- ◆ **Definicija.** Izpeljava stavčne oblike po gramatiki G je **skrajno leva**, če na *vsakem* koraku uporabimo produkcijo, ki razvije *skrajno levi vmesni simbol trenutne stavčne oblike*. Podobno je izpeljava stavčne oblike **skrajno desna**, če na *vsakem* koraku uporabimo produkcijo, ki razvije *skrajno desni vmesni simbol trenutne stavčne oblike*.
- ◆ **Primer.** Skrajno leva izpeljava končnega niza $aabbba$ je

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbba$$

```

graph TD
    S((S)) --> a((a))
    S --> A1((A))
    S --> S1((S))
    a --> a1((a))
    A1 --> b1((b))
    A1 --> A2((A))
    S1 --> a2((a))
    b1 --> a3((a))
    A2 --> b2((b))
    A2 --> a4((a))
    a3 --> a5((a))
    b2 --> a6((a))
  
```

skrajno desna izpeljava istega niza pa

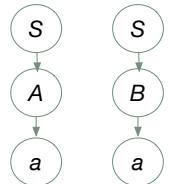
$$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbba \Rightarrow aabbba$$

```

graph TD
    S((S)) --> a((a))
    S --> A1((A))
    S --> S1((S))
    a --> a1((a))
    A1 --> b1((b))
    A1 --> A2((A))
    S1 --> a2((a))
    b1 --> a3((a))
    A2 --> b2((b))
    A2 --> a4((a))
    a3 --> a5((a))
    b2 --> a6((a))
  
```

◆ Dvoumnost

- ◆ Bodи G KNG. Če je $w \in L(G)$, ima w drevo izpeljav D_G (po prejšnjem izreku). Temu drevesu izpeljav besede w pripadata *natančno določena skrajno leva izpeljava* besede w in *natančno določena skrajno desna izpeljava* besede w .
- ◆ **Vprašanje.** Kaj pa, če je w rob več kot enega drevesa izpeljav? Je to možno? Je!
 - ◆ **Dokaz (s primerom).** Naj bo $G = (\{S,A,B\}, \{a\}, \{S \rightarrow A \mid B, A \rightarrow a, B \rightarrow a\})$.
Opazimo: končni niz je $a \in L(G)$, toda zanj obstajata *dve* drevesi izpeljav.
(Prvo drevo dá skrajno levo izpeljavo $S \xrightarrow{G} A \xrightarrow{G} a$, drugo pa $S \xrightarrow{G} B \xrightarrow{G} a$.)
- ◆ **Definicija.** KNG G je **dvoumna**, če obstaja $w \in L(G)$, ki ima *več kot eno* drevo izpeljav.
Ekvivalenta definicija: KNG G je *dvoumna*, če ima kaka beseda $w \in L(G)$ *več kot eno* skrajno levo (desno) izpeljavo po G .
- ◆ **Definicija.** KNJ L je **bistveno dvoumen**, če je vsaka KNG za jezik L *dvoumna*.
Opomba. Kmalu bomo videli, da taki jeziki obstajajo !



4.4 Normalne oblike kontekstno-neodvisnih gramatik

- ◆ Obstajajo načini, s katerimi lahko *omejimo obliko produkciј KNG*, ne da bi pri tem zmanjšali "moč" KNJ. Natančneje: če je L neprazen KNJ, potem je L jezik (neke) gramatike G ($L = L(G)$), ki ima naslednje lastnosti:
 - ◆ Vsak vmesni in končni simbol gramatike G nastopa v izpeljavi vsaj ene besede $w \in L$.
 - ◆ V gramatiki G ni produkциј oblike $A \rightarrow B$ (kjer sta A, B poljubna vmesna simbola).
 - ◆ Če $\varepsilon \notin L$, potem v G ni produkциј oblike $A \rightarrow \varepsilon$ (kjer je A poljuben vmesni simbol).
 - ◆ Če $\varepsilon \notin L$, potem je v G
 - ◆ vsaka produkacija oblike $A \rightarrow BC$ ali oblike $A \rightarrow b$ (*normalna oblika Chomskega*) (kjer sta A, B poljubna vmesna simbola in b poljuben končni simbol).
- ali
 - ◆ vsaka produkacija oblike $A \rightarrow b\gamma$ (*normalna oblika Greibachove*) (kjer je γ poljuben niz vmesnih simbolov ($\gamma \in V^*$) in b poljuben končni simbol ($b \in T$)).

► Odstranitev odvečnih simbolov

Simbol gramatike G je *potreben*, če nastopa v izpeljavi kake besede $w \in L(G)$.

- **Definicija.** Bodi $G = (V, T, P, S)$ KNG. Simbol X je *potreben*, če obstaja izpeljava $S \xrightarrow{G^*} \alpha X \beta \xrightarrow{G^*} w$, kjer sta $\alpha, \beta \in (V \cup T)^*$ in $w \in T^*$. Če X ni potreben, je *odvečen*.
 - **Lema 1.** Poljubni KNG $G = (V, T, P, S)$, kjer $L(G) \neq \emptyset$ lahko *efektivno* priredimo *ekvivalentno* KNG $G' = (V', T, P', S)$, kjer za vsak vmesni simbol $A \in V'$ obstaja $w \in T^*$, da velja $A \xrightarrow{G'} \Rightarrow^* w$. (tj., iz vsakega vmesnega simbola A se dá izpeljati po G' nek končni niz w).
 - **Lema 2.** Poljubni KNG $G' = (V', T, P', S)$ lahko *efektivno* priredimo *ekvivalentno* KNG $G'' = (V'', T', P'', S)$, kjer za vsak (vmesni ali končni) simbol $X \in V'' \cup T'$ obstajata $\alpha, \beta \in (V'' \cup T')^*$, da velja $S \xrightarrow{G''} \Rightarrow^* \alpha X \beta$. (tj., vsak vmesni ali končni simbol X se pojavi v neki izpeljavi po G'' iz S).
 - Uporaba Leme 1 in *nato* Leme 2 priredi KNG G ekvivalentno G'' brez odvečnih simbolov.
(Uporaba najprej leme 2 in nato 1 v nekaterih primerih ne odstrani vseh odvečnih simbolov.)
- **Izrek.** Vsak neprazen KNJ generira neka KNG, ki je brez odvečnih simbolov.
- **Opomba.** Odslej bomo predpostavljali, da so vse KNG, ki jih bomo rabili, brez odvečnih simbolov.

◆ Odstranitev ε -produkciј

◆ **Definicija.** ε -produkciј je produkciј oblike $A \rightarrow \varepsilon$. (A je poljuben vmesni simbol.)

Če je $\varepsilon \in L(G)$, očitno ne moremo iz G odstraniti vseh ε -produkciј (ker potem ε ne bi bil več v $L(S)$). Če pa $\varepsilon \notin L(G)$, lahko odstranimo vse ε -produkciјe iz G .

◆ **Izrek.** Če je $L = L(G)$ za neko KNG $G = (V, T, P, S)$, potem jezik $L - \{\varepsilon\}$ generira neka KNG G' , ki je brez ε -produkciј. (in brez odvečnih simbolov).

Ideja dokaza.

- ◆ Za vsak vmesni symbol $A \in V$ ugotovi, ali $A \xrightarrow{G}^* \varepsilon$. Če velja, rečemo, da je simbol A **uničljiv**. Naj bo U množica vseh uničljivih vmesnih simbolov gramatike G .
- ◆ Vsako produkciјo $B \rightarrow X_1X_2\dots X_n$ nadomesti z novimi produkciјami, ki nastanejo, ko iz $X_1X_2\dots X_n$ izločamo simbole iz podmnožic množice U ; med nove produkciјe pa ne štej $B \rightarrow \varepsilon$ (tudi če so vsi $X_1 X_2\dots X_n$ uničljivi).

□

- ◆ Odstranitev enotskih produkcijs
- ◆ **Definicija.** Enotska produkcija je produkcija oblike $A \rightarrow B$. (A,B sta poljubna vm.simb.)

Pozor: $A \rightarrow a$, kjer je a končni symbol, ni enotska produkcija, prav tako tudi $A \rightarrow \epsilon$ ni enotska produkcija

Izrek. Vsak KNJ brez ϵ lahko generira KNG, ki je brez enotskih produkcijs.
(in brez ϵ -produkcijs ter odvečnih simbolov).

Ideja dokaza. Letos izpustim. □

4.5 Normalna oblika Chomskega

- ◆ Izreki o normalnih oblikah kontekstno-neodvisnih gramatik pravijo, da so te gramatike ekvivalentne kontekstno-neodvisnim gramatikam, katerih produkcijs imajo posebno obliko. To se izkaže za zelo koristno.
- ◆ Prvi tak izrek je odkril Noam Chomsky. Izrek pravi, da imajo vse take produkcijs na desni strani bodisi *dva vmesna simbola* bodisi *en končni simbol*.

Izrek (Normalna oblika Chomskega). Vsak KNJ, ki ne vsebuje ϵ , se dá generirati s KNG, katere produkcijs so oblike

$$A \rightarrow BC$$

ali $A \rightarrow a$

Tu so $A, B, C \in V$ poljubni vmesni simboli in $a \in T$ poljuben končni simbol gramatike.

► Ideja dokaza (konstruktivnega).

- ◆ Naj bo $L(G)$ KNJ brez ε .
- ◆ Sestavimo eqvivalentno KNG $G_1 = (V, T, P, S)$, ki je brez odvečnih simbolov, enotskih produkcij in ε -produkcijs.
- ◆ Če ima produkcija iz P na desni strani en sam simbol, je to končni simbol, zato je produkcija že v želeni obliki.
- ◆ Če produkcija na desni strani nima enega samega simbola, mora biti oblike $A \rightarrow X_1 X_2 \dots X_m$ ($m \geq 2$). (X_i je vmesni ali končni simbol.)

◆ Če je X_i končni simbol, npr. a , potem

- ◆ uvedemo nov vmesni symbol C_a
- ◆ uvedemo novo produkcijo $X_i \rightarrow a$ (ta je v želeni obliki)
- ◆ nadomestimo X_i s C_a .

Ko to storimo z vsemi X_i , ki so končni simboli, dobimo novo množico V' vmesnih simbolov in novo množico P' produkcij.

Naj bo $G_2 = (V', T, P', S)$. Dokažemo lahko, da velja $L(G_1) = L(G_2)$. (Poskusite za vajo.)

- ◆ Torej $L(G)$ generira KNG G_2 , katere produkcije so bodisi oblike $A \rightarrow a$ bodisi oblike $A \rightarrow B_1 B_2 \dots B_m$ ($m \geq 2$). (B_i je vmesni simbol, a pa končni simbol.)

◆ Če je produkcija oblike $A \rightarrow B_1 B_2 \dots B_m$ in $m \geq 3$, potem

- ◆ uvedemo nove vmesne simbole D_1, D_2, \dots, D_{m-2}
- ◆ nadomestimo produkcijo s produkcijami $A \rightarrow B_1 D_1$

$$D_1 \rightarrow B_2 D_2 \\ \ddots$$

$$D_{m-3} \rightarrow B_{m-2} D_{m-2} \\ D_{m-2} \rightarrow B_{m-1} B_m$$

Ko to storimo z vsemi produkcijami $A \rightarrow B_1 B_2 \dots B_m$, ki imajo $m \geq 3$, dobimo novo množico V'' vmesnih spremenljivk in novo množico P'' produkcij, ki imajo obliko $A \rightarrow a$ ali $A \rightarrow BC$.

Naj bo $G_3 = (V'', T, P'', S)$. Dokažemo lahko, da velja $L(G_2) = L(G_3)$. Potem sledi še $L(G) = L(G_3)$. (Poskusite za vajo.)



4.6 Normalna oblika Greibachove

- Drug tak izrek je odkrila *Sheila Greibach*. Njen izrek pravi, da se take produkcijs na desni strani začnejo s končnim simbolom, ki mu sledijo samo vmesni simboli.

Izrek (Normalna oblika Greibachove). Vsak KNJ, ki ne vsebuje ϵ , se dá generirati s KNG, katere produkcijs so oblike

$$A \rightarrow b\gamma$$

Tu je $A \in V$ poljuben vmesni simbol, $b \in T$ poljuben končni simbol in $\gamma \in V^*$ poljuben niz vmesnih simbolov.

◆ Ideja dokaza (konstruktivnega).

- ◆ Naj bo $L(G)$ KNJ brez ε , kjer je gramatika $G = (V, T, P, S)$ v normalni obliki Chomskega in $V = \{A_1, A_2, \dots, A_m\}$.
- ◆ Sestavimo eqvivalentno KNG $G_1 = (V, T, P, S)$, ki je brez odvečnih simbolov, enotskih produkcij in ε -produkcijs.
- ◆ Popravimo produkcije tako, da bo veljalo naslednje: če je $A_i \rightarrow A_j \gamma$ produkcija, potem $j > i$.
Kako?

Uvedemo nove vmesne simbole B_1, B_2, \dots, B_m . Produkcije so zdaj oblik

$$\begin{aligned} A_i &\rightarrow A_j \gamma, \text{ kjer } j > i \\ A_i &\rightarrow a \gamma, \text{ kjer } a \in T \\ A_i &\rightarrow \gamma, \text{ kjer } \gamma \in (V \cup \{B_1, B_2, \dots, B_{i-1}\})^*. \end{aligned}$$

- ◆ Spremeni vse A_m -produkcijs, potem vse A_{m-1} -produkcijs, potem vse A_{m-2} -produkcijs, potem vse A_1 -produkcijs.
Kako?

Spreminjanje A_k -produkcijs ($m \geq k \geq 1$) poteka na naslednji način:

Za vsako A_k -produkcijs:

poišči v njenem desnem delu skrajno levi vmesni symbol, npr. X ;
zamenjaj ta simbol X s stikom desnih strani vseh X -produkcijs.

Zdaj imajo vse A -produkcijs desne strani, ki se začnejo s končnim simbolum.

- ◆ Toda desne strani B -produkcijs se še vedno lahko začnejo s kakim vmesnim simbolum A_i . Zato popravimo še to.
Kako?

Modify the productions for the new variables B_1, B_2, \dots, B_m .

V vsaki B_k -produkcijs ($k = 1, 2, \dots, m$), katere desna stran se začne z nekim vmesnim simbolum, npr. A_i :
zamenjaj A_i s stikom desnih strani vseh A_i -produkcijs.

□

4.7 Bistveno dvoumni kontekstno-neodvisni jeziki

- ◆ Dvoumne KNG ni težko najti: KNG s produkcijami $S \rightarrow A \mid B$, $A \rightarrow a$, $B \rightarrow a$ je dvoumna. (Glej eno od prejšnjih prosojnic.)
 - ◆ Dosti težje je najti KNJ, za katerega bi bila *vsaka KNG dvoumna*.
- Definicija.** KNJ L je **bistveno dvoumen**, če: $L = L(G) \wedge G$ je KNG $\Rightarrow G$ je dvoumna.
- ◆ Toda, ali tak jezik sploh obstaja? Obstaja!
- Izrek.** KNJ $L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$ je bistveno dvoumen.
- Dokaz.** S protislovjem (dolg) \square

4.8 Slovar

context-free grammar kontekstno-neodvisna gramatika context-free language kontekstno-neodvisen jezik variable vmesni simbol terminal končni simbol production produkcija to derive izpeljati start symbol začetni simbol to apply (a production) uporabiti (produkциjo) to directly derive neposredno izpeljati language generated generiran (izpeljan) jezik sentential form stavčna oblika derivation tree drevo izpeljave yield (of a derivation tree) rob (drevesa izpeljave) subtree poddrevo leftmost/rightmost derivation skrano leva/desna izpeljava ambiguous dvoumen inherently ambiguous bistveno dvoumen format of a production oblika produkcije useful/useless symbol koristen/odvečen simbol ϵ -production ϵ -produkacija nullable variable uničljiv vmesni simbol unit production enotska produkcija Chomsky normal form normalna oblika Chomskega Greibach normal form normalna oblika Greibachove

5

Skladovni avtomati

Vsebina

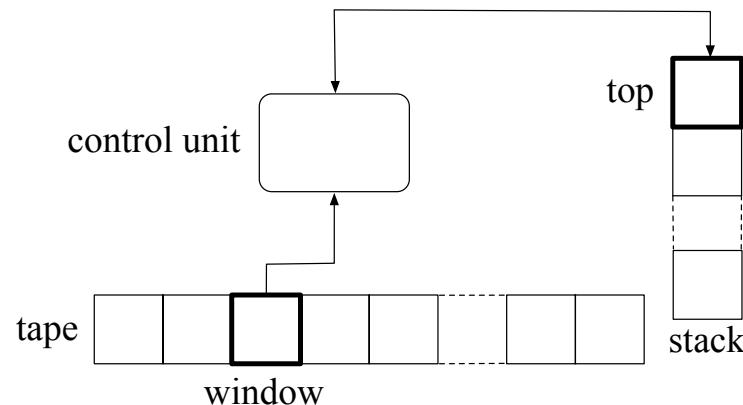
- ◆ Uvod
- ◆ Definicije
- ◆ Skladovni avtomati in kontekstno-neodvisni jeziki

5.1 Uvod

- ◆ Tako kot so regularni izrazi in regularni jeziki tesno povezani s končnimi avtomati (KA), so tudi kontekstno-neodvisne gramatike (KNG) in kontekstno-odvisni jeziki (KNJ) tesno povezani s takoimenovanimi **skladovnimi avtomati (SA)**.
- ◆ SA je v bistvu KA, ki poleg traka uporablja še a **sklad**.
- ◆ Vednar se razlikujeta: SA je *po definiciji nedeterministični model računanja*, njegova *deterministična različica (DSA)* pa sprejema *pravi podrazred razreda vseh KNJ*.
- ◆ Na srečo ta podrazred zajema *večino programskih jezikov*.

Intuitivno.

- SA ima **vhodni trak**, **nadzorno enoto** in **sklad**.
 - Sklad je niz simbolov iz neke abecede.
Skrajno levi simbol tega niza je na vrhu sklada.*



- SA je po definiciji **nedeterminističen** (v dani situaciji ima končno število $(0, 1, 2, \dots)$) alternativ za naslednjo potezo.

- ◆ Poteze so dveh vrst: *navadne* in *tihe* (ε -poteze).
 - ◆ Pri **navadni potezi** se vhodni simbol *konzumira*. Poglejmo kako.

V odvisnosti od

- ◆ stanja q nadzorne enote,
- ◆ vhodnega simbola a v oknu in
- ◆ simbola Z na vrhu sklada,

obstaja končno mnogo alternativ

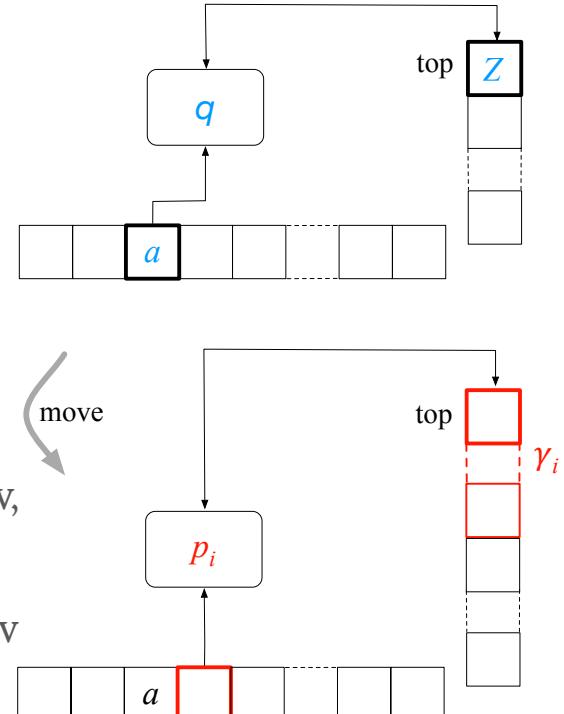
$$(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m),$$

kjer i -ta alternativa (p_i, γ_i) vsebuje

- ◆ naslednje stanje p_i nadzorne enote,
- ◆ niz γ_i (lahko tudi prazen) skladovnih simbолов, ki bodo nadomestili Z .

SA *nedeterministično* izbere in izvede eno od alternativ

in *premakne okno na naslednjo celico* (konzumira vh. simbol).



- (nadaljevanje)

- Pri **tihi potezi** se vhodni simbol *ne konzumira*.

V odvisnosti od

- stanja q nadzorne enote in
- simbola Z na vrhu sklada,
- ter *neodvisno od vhodnega simbola* ●,

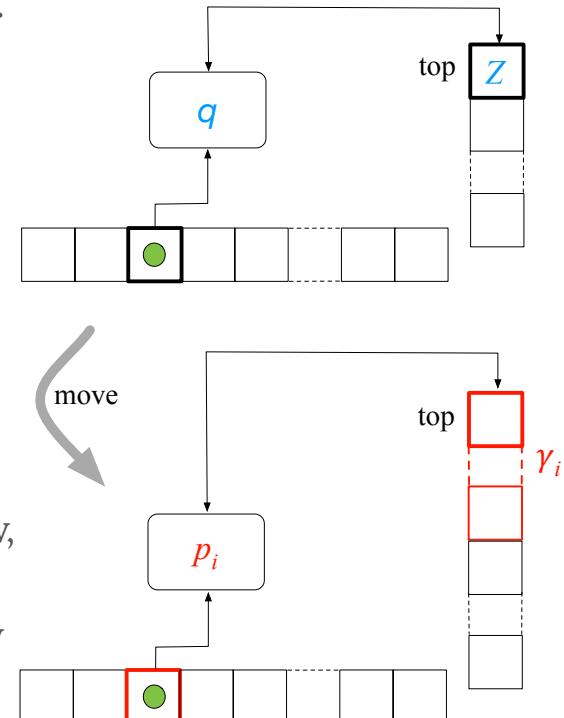
obstaja končno mnogo alternativ

$(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)$,
kjer i -ta alternativa (p_i, γ_i) vsebuje

- naslednje stanje p_i nadzorne enote,
- niz γ_i (lahko tudi prazen) skladovnih simbolov,
ki bodo nadomestili Z .

SA *nedeterministično izbere* in izvede eno od alternativ
in *pusti okno na isti celici* (ne konzumira vh. simbola).

- *Opomba:* Tihe poteze omogočajo SA manipulira s skladom, ne da bi bral vhodne simbole s traku.



- ◆ Jezik, ki ga sprejme SA, lahko definiramo na dva načina.
Poglejmo množico vseh besed w z lastnostjo
 - ① Ko SA prebere w , se znajde v *končnem stanju*.
Pravimo, da je ta množica *jezik*, ki ga SA **sprejme s končnim stanjem**.
 - ② Ko SA prebere w , *izprazni svoj sklad*.
Pravimo, da je ta množica *jezik*, ki ga SA **sprejme s praznim skladom..**
- ◆ Videli bomo, da sta definiciji ekvivalentni v naslednjem smislu:
 L sprejme SA M_1 s *končnim stanjem* *natanko tedaj, ko* L sprejme SA M_2 s *praznim skladom*
- ◆ Prva definicija je bolj običajna, z drugo pa je lažje dokazati **osnovni izrek o skladovnih avtomatih**, ki pravi tole:
 L sprejme SA *natanko tedaj, ko* L je KNJ.

5.2 Definicije

- ◆ **Definicija.** Skladovni avtomat (**SA**) je sedmerka $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kjer so
 - ◆ Q končna množica **stanj**,
 - ◆ Σ **vhodna abeceda**,
 - ◆ Γ **skladovna abeceda**,
 - ◆ $q_0 \in Q$ **začetno stanje**,
 - ◆ $Z_0 \in \Gamma$ **začetni skladovni simbol**,
 - ◆ $F \subseteq Q$ množica **končnih stanj**, in
 - ◆ δ **funkcija prehodov**,
tj. preslikava iz $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ v podmnožice množice $Q \times \Gamma^*$.
- ◆ **Opomba:** δ je program SA; vsak SA ima svoj specifični δ .

◆ Poteze skladovnega avtomata.

◆ Prehod

- ◆ $\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$ pomeni, da SA v stanju q , z vhodnim simbolom a v oknu in skladovnim simbolom Z na vrhu sklada, nedeterministično izbere i ($1 \leq i \leq m$) in stori naslednje: preide v stanje p_i , zamenja Z z nizom γ_i in premakne okno na naslednjo celico traku.
To je **navadna poveza**.
- ◆ $\delta(q, \varepsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$ pomeni, da SA v stanju q , s skladovnim simbolom Z na vrhu sklada in neodvisno od vhodnega simbola v oknu, nedeterministično izbere i ($1 \leq i \leq m$) in stori naslednje: preide v stanje p_i in zamenja Z z nizom γ_i (okno pusti pri miru)
To je **tiha poteza**.

Dogovori: skrajno levi simbol v γ_i je na vrhu sklada. Vhodne simbole bomo označevali z a, b, c, \dots , nize vhodnih simbolov z u, v, w, \dots , skladovne simbole z A, B, C, \dots , nize skladovnih simbolov pa z $\alpha, \beta, \gamma \dots$

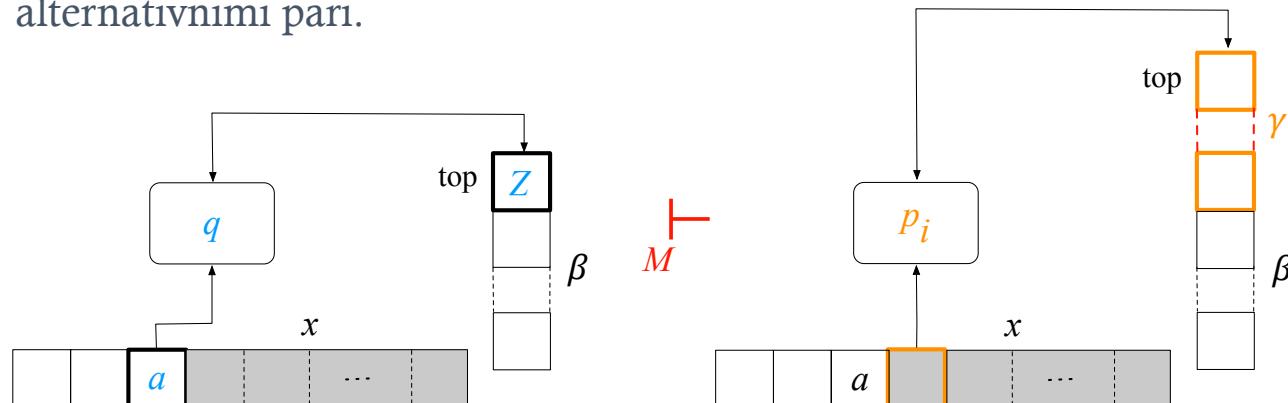
◆ Trenutni opisi.

- ◆ *Trenutni opis* (konfiguracijo) SA v nekem trenutku sestavljajo *trenutno stanje* SA, *dotlej neprebrani del vhodne besede* in *trenutna vsebina sklada*. Trenutni opisi nam bodo omogočili, da natančno opišemo *računanje*, ki ga izvaja SA.
- ◆ **Definicije.** *Trenutni opis* (TO) je trojka (q, w, γ) , kjer je q stanje, w niz vhodnih simbolov in γ niz skladovnih simbolov. Med delovanjem SA je vsak TO v zvezi s prejšnjim TO.
 - ◆ Če je $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ SA, rečemo, da TO $(q, ax, Z\beta)$ **neposredno preide v** TO $(p_i, x, \gamma_i\beta)$ --- kar zapišemo $(q, ax, Z\beta) \xrightarrow{M} (p_i, x, \gamma_i\beta)$ --- če $\delta(q, a, Z)$ vsebuje (p_i, γ_i) . Tu je namesto vhodnega simbola a lahko tudi ε (zaradih tihih prehodov).
 - ◆ *Refleksivno tranzitivno zaprtje* relacije \xrightarrow{M} je relacija \xrightarrow{M}^* . Če velja $I \xrightarrow{M}^* J$, rečemo, da TO I **preide v** TO J . Če I prede v J v k neposrednih prehodih, to pišemo kot $I \xrightarrow{M}^k J$.

Opomba. Kadar bo jasno, kateri SA M je mišljen, bomo indeks M opustili.

(nadaljevanje)

- Intuitivno: situacija na levi strani slike lahko *neposredno preide* v situacijo na desni samo če program δ skladovnega avtomata M vsebuje ukaz $\delta(q, a, Z) = \{\dots, (p_i, \gamma_i), \dots\}$. Se bo ta prehod tudi izvedel? To bo odločil M , ki bo *nedeterministično izbiral* med alternativnimi pari.



ID $(q, ax, Z\beta)$ directly becomes ID $(p_i, x, \gamma_i \beta)$
if $\delta(q, a, Z)$ contains (p_i, γ_i)

- ◆ **Jeziki, ki jih sprejemajo SA.**
 - ◆ **Definicija.** Vsak SA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ sprejme *dva* jezika:
 - ◆ **$L(M)$, jezik sprejet s končnim stanjem**, ki je definiran kot
 

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \gamma), \text{ kjer je } p \in F \text{ in } \gamma \in \Gamma^*\}$$
 - ◆ **$N(M)$, jezik sprejet s praznim skladom**, ki je definiran kot
 

$$N(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon), \text{ kjer je } p \in Q\}.$$
- $L(M)$ vsebuje besedo w , če se M po branju w lahko (nedeterminizem!) znajde v kakem *končnem stanju*. $N(M)$ vsebuje besedo w , če ima M po branju w lahko (nedeterminizem!) *prazen sklad*.
- Opomba.* Pri sprejemanju s *praznim skladom* so končna stanja nepomembna, zato je takrat *lahko* $F = \emptyset$.

- ◆ **Primer.** Sestavimo SA M , ki sprejme jezik $N(M) = \{wcw^R \mid w \in (0+1)^*\}$ s praznim skladom.
- ◆ **Zamisel.** Beri vh.besedo v stanju q_1 in za vsak prebrani vh.simbol *porini* njegovega predstavnika na sklad. (Predstavnik 0 je B, predstavnik 1 pa G.) Ko prebereš c , *preidi* v drugo stanje, q_2 . Nadaljuj z branjem vhodne besede v tem stanju in za vsak prebrani vh.simbol *odstrani* simbol na vrhu sklada, če je predstavnik prebranega vh.simbola (če ni, se ustavi). Če ni več vh.simbolov in je bil pravkar odstranjen symbol Z_0 (oznaka dna sklada), je bila vh.beseda oblike wcw^R . Ker je sklad prazen, to besedo sprejmi.
- ◆ SA je $M = (\{q_1, q_2\}, \{0,1,c\}, \{R,B,G\}, \delta, q_1, Z_0, \emptyset)$, kjer je program δ definiran takole:
 1. $\delta(q_1, 0, Z_0) = \{(q_1, BZ_0)\}$
 2. $\delta(q_1, 1, Z_0) = \{(q_1, GZ_0)\}$
 3. $\delta(q_1, 0, B) = \{(q_1, BB)\}$
 4. $\delta(q_1, 1, B) = \{(q_1, GB)\}$
 5. $\delta(q_1, 0, G) = \{(q_1, BG)\}$
 6. $\delta(q_1, 1, G) = \{(q_1, GG)\}$
 7. $\delta(q_1, c, Z_0) = \{(q_2, Z_0)\}$
 8. $\delta(q_1, c, B) = \{(q_2, B)\}$
 9. $\delta(q_1, c, G) = \{(q_2, G)\}$
 10. $\delta(q_2, 0, B) = \{(q_2, \varepsilon)\}$
 11. $\delta(q_2, 1, G) = \{(q_2, \varepsilon)\}$
 12. $\delta(q_2, \varepsilon, Z_0) = \{(q_2, \varepsilon)\}$

Opomba. Čeprav so SA po definiciji *nedeterministični*, zgornji M deluje *deterministično*, saj ima vedno na izbiro le eno možnost.

- **Primer.** Sestavimo SA M' , ki sprejme $N(M') = \{ww^R \mid w \in (0+1)^*\}$ s praznim skladom.
- **Pozor.** Za razliko od prejšnjega primera tu ni več simbola c , ki bi označeval sredino vh.besede. Torej bo moral M' uganiti, ali je med branjem dosegel sredino vh.besede. Ker bo M' nedeterminističen, bo to pravilno ugani in takrat prešel iz stanja q_1 v stanje q_2 . Torej bomo v prejšnji program dodali možnost izbiranja (ali je sredina dosežena ali ne).
- **Zamisel.** Beri vh.besedo v stanju q_1 in za vsak prebrani vh.simbol *porini* njegovega predstavnika na sklad. (Predstavnik 0 je B, predstavnik 1 pa G.). Vsakokrat, ko bo prebrani vh.simbol “ustrezal” vrhnjemu simbolu sklada, je morda dosežena sredina vh.besede. Nedeterministično ugani, ali je sredina dosežena ali ni; če je, preidi v stanje q_2 , sicer pa *porini* predstavnika prebranega vh.simbla na sklad. Ko uganeš (zaznaš) sredino vh.besede, nadaljuj z branjem v tem stanju in za vsak prebrani vh.simbol *odstrani* simbol z vrha sklada, če je predstavnik prebranega vh.simbla (če ni, se ustavi). Če ni več vh.simbolov, je bila vh.beseda oblike ww^R . Ker je na skladu le še Z_0 , ga odstrani. Tako bo vh.beseda sprejeta. Če pa nikoli ne zaznaš sredine vh.besede, je morala biti vh.beseda bodisi ε ali en sam simbol, zato jo sprejmi.
- $M' = (\{q_1, q_2\}, \{0,1\}, \{R,B,G\}, \delta, q_1, Z_0, \emptyset)$ deluje po naslednjem programu δ :

- | | |
|--|--|
| 1. $\delta(q_1, 0, R) = \{(q_1, BR)\}$ | 2. $\delta(q_1, 0, G) = \{(q_1, BG)\}$ |
| 3. $\delta(q_1, 1, R) = \{(q_1, GR)\}$ | 4. $\delta(q_1, 1, B) = \{(q_1, GB)\}$ |
| 5. $\delta(q_1, 0, B) = \{(q_1, BB), (q_2, \varepsilon)\}$ | 6. $\delta(q_1, 1, G) = \{(q_1, GG), (q_2, \varepsilon)\}$ |
| 7. $\delta(q_2, 0, B) = \{(q_2, \varepsilon)\}$ | 8. $\delta(q_2, 1, G) = \{(q_2, \varepsilon)\}$ |
| 9. $\delta(q_2, \varepsilon, R) = \{(q_2, \varepsilon)\}$ | 10. $\delta(q_1, \varepsilon, R) = \{(q_2, \varepsilon)\}$ |

- ◆ Intuitivno gledano, je bil SA M , ki je sprejemal $N(M) = \{wcw^R \mid w \in (0+1)^*\}$, “determinističen” v smislu, da je imel v *vsakem* TO na voljo le *eno* potezo. *Stroga definicija determinističnega* SA je bolj natančna.
- ◆ **Definicja.** SA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je **determinističen**, če za δ velja:
Za vsak par $q \in Q$ and $Z \in \Gamma$:
 1. $\delta(q, \varepsilon, Z) \neq \emptyset \implies \forall a \in \Sigma : \delta(q, a, Z) = \emptyset$
 2. $\forall a \in \Sigma \cup \{\varepsilon\} : |\delta(q, a, Z)| \leq 1$

Kaj to pomeni? Pogoj 1 preprečuje izbiranje med *navadnimi* in *tihimi* potezami.
Pogoj 2 preprečuje izbiranje pri navadnih potezah in izbiranje pri tihih potezah .

- ◆ **Opomba.** Ker je SA po definiciji nedeterminističen, bomo deterministične SA označevali z **DSA** (**deterministični skladovni avtomati**).

5.3 Skladovni avtomati in kontekstno-neodvisni jeziki

- ◆ Vemo, da *deterministični* KA sprejmejo *isti* razred jezikov kot *nedeterministični* KA (regularne jezike). Ali je tako tudi pri *determinističnih* in *nedeterminističnih* SA?
- ◆ Ali razred jezikov, sprejetih s SA, vsebuje kake že *znane* jezike? (regularne? KNJ? ...?)
- ◆ Kako se razlikujejo jeziki $L(M)$ (sprejeti s končnim stanjem) od jezikov $N(M)$ (sprejetih s praznim skladom)?

Ekvivalenca sprejemanja s končnim stanjem in praznim skladom.

- Ali se sprejemanje s *končnim stanjem* in sprejemaje s *praznim skladom* razlikujeta v „moči“? Ne! Da to vidimo, moramo dokazati, da je *razred jezikov* $L(M)$ (sprejetih s končnim stanjem) enak *razredu jezikov* $N(M)$ (sprejetih s končnim stanjem). Slednje pa dokažemo, če dokažemo: *L sprejme neki SA s končnim stanjem natanko tedaj, ko L sprejeme neki SA s praznim skladom.* To izražata naslednja dva izreka.

Izrek. Če je $L=L(M_2)$ za nek SA M_2 , potem je $L=N(M_1)$ za nek (drugi) SA M_1 .

- Ideja dokaza.** Naj bo $L = L(M_2)$ poljuben jezik. Konstruiramo SA M_1 , ki simulira M_2 , pri tem pa še zбриše svoj sklad vsakokrat, ko bi M_2 vstopil v kako končno stanje. Zato je $L = N(M_1)$. \square

Izrek. Če je $L=N(M_1)$ za nek SA M_1 , potem je $L=L(M_2)$ za nek (drugi) SA M_2 .

- Zamisel dokaza.** Naj bo $L = N(M_1)$ poljuben jezik. Konstruiramo SA M_2 , ki simulira M_1 , pri tem pa še vstopi v končno stanje vsakokrat, ko bi M_1 izpranil svoj sklad. Zato je $L = L(M_2)$. \square

- Posledica:** Razred jezikov, ki jih sprejmejo SA s končnim stanjem, je enak razredu jezikov, ki jih sprejemejo SA s prazni skladom.

♣ Zveza med skladovnimi avtomati in kontekstno-neodvisnimi jeziki.

- ♣ Ali obstaja kakšna zveza med jeziki, sprejetimi s SA, in regularnimi in/ali kontekstno-neodvisnimi jeziki? *Slutimo*, da so SA „močnejši“ od KA, tj. da sprejemejo več kot le razred regularnih jezikov.

Ali morda SA sprejemajo vse KNJ? (Da bi to dokazali, moramo dokazati tole: *Če je L KNJ, potem L sprejme neki SA.*) Če to drži, *ali morda SA sprejemajo samo KNJ?* (Da bi to dokazali, moramo dokazati: *Če L sprejeme neki SA, potem je L KNJ.*) Oboje drži -- to povesta spodnja izreka.

♣ Izrek. **Če je L KNJ, potem** obstaja SA M , da je $L = N(M)$.

- ♣ **Ideja dokaza.** Naj bo L poljubien KNJ. L lahko generira KNG G v Greibachovi normalni obliki. Konstruiramo SA M , ki simulira skrajno leve izpeljave po G . (M naj sprejema s praznim skladom) Sledi, da je $L = N(M)$. □

♣ Izrek. **Če je $L = N(M)$ za neki SA M , potem** je L KNJ..

- ♣ **Ideja dokaza.** Naj bo M poljuben SA. Konstruiramo KNG G tako, da skrajni leva izpeljava po v besede x simulira (opisuje) delovanje SA M na vhodni besedi x . Sledi, da je $L = L(G)$, torej KNJ. □

♣ **Posledica:** Razred vseh jezikov, sprejetih s SA, je enak razredu vseh KNJ.

♣ Deterministični in nedeterministični SA.

- ♣ Videli smo, da SA (nedeterministični po definiciji) sprejemajo natanko KNJ. Katere jezike pa sprejemajo DSA (deterministične različice SA)? Ker so DSA le omejeni SA, se vprašamo, ali so dovolj "močni, da bi sprejemali vse KNJ?

Vprašanje: Ali je razred jezikov, ki jih sprejemajo DSA, isti kot razred vseh KNJ?

Odgovor: Ne. Obstaja jezik, ki je KNJ, a ga ne sprejme noben DSA.

- ♣ **Izrek.** Jezik $\{ww^R \mid w \in (0+1)^*\}$ sprejme neki SA in noben DSA.
 - ♣ **Ideja dokaza.** Opustim. □

♣ Sklep: DSA je po sposobnosti sprejemanja šibkejši od SA.

Opomba. Pri skladovnih avtomatih nedeterminizem vpliva na njihovo sposobnost sprejemanja. To je drugače kot pri končnih avtomatih, kjer nedeterminizem nima vpliva na njihovo sposobnost sprejemanja. Nauk. *Pri različnih modelih računanja nedeterminizem različno vpliva na njihovo sposobnost sprejemanja.*

5.4 Slovar

pushdown automaton skladovni avtomat **stack** sklad **regular move** navaden prehod, navadna poteza **ϵ -move** tih prehod, tih poteza **language accepted by empty stack (final state)** jezik sprejet s praznim skladom (končnim stanjem) **basic theorem of PDA** osnovni izrek o skladovnih avtomatih **stack alphabet** skladovna abeceda **instantaneous description** trenutni opis **directly becomes** neposredno preide v **becomes** preide v

6

Lastnosti kontekstno-neodvisnih jezikov

Vsebina

- ◆ Lema o napihovanju za KNJ
- ◆ Lastnosti zaprtosti KNJ
- ◆ Odločitveni problemi in algoritmi za KNJ

6.1 Uvod

- ◆ V tem poglavju bomo opisali:
 - ◆ **Lemo o napihovanju** za KNJ. Uporabna bo npr. pri dokazovanju, da nekateri jeziki niso KNJ.
 - ◆ Nekatere operacije, ki ohranjajo KNJ. Te lastnosti **zaprtosti** so uporabne pri ugotavljanju, da nekateri jeziki *so* KNJ, pa tudi pri ugotavljanju, da nekateri jeziki *niso* KNJ.
 - ◆ **Odločitvene algoritme** za iskanje odgovorov na nekatera vprašanja o KNJ. npr. ali je dani KNJ *prazen*, *končen*, ali *vsebuje* dano besedo.
 - ◆ Vprašanja o KNJ, na katera *ne more (vedno) odgovoriti noben algoritem!*

6.2 Lema o napihovanju za KNJ

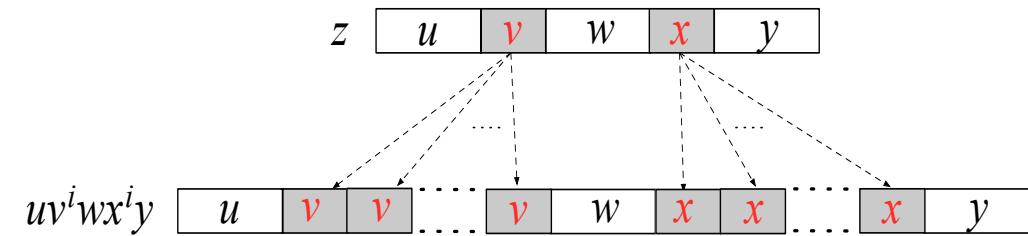
- ◆ **Spomnimo se:** Lema o napihovanju za regularne jezike pravi, da vsaka *zadosti dolga* beseda iz regularnega jezika vsebuje *blizu* svojega začetka *kratko* podbesedo, ki jo lahko ponovimo *končno mnogokrat*, a bo nova beseda še vedno v tem jeziku.
- ◆ **Lema o napihovanju za KNJ** pravi, da vsaka *zadosti dolga* beseda iz KNJ jezika L vsebuje *dve kratki* podbesedi, ki sta *si blizu* in ki ju lahko *končno mnogokrat* ponovimo (obe enako mnogokrat), a bo nova beseda še vedno v jeziku L .

- ◆ **Lema o napihovanju** (za kontekstno-neodvisne jezike). Naj bo L KNJ.
Potem obstaja konstanta n (odvisna samo od L), da velja naslednje :
če je z poljubna beseda z lastnostjo

$$z \in L \wedge |z| \geq n,$$

potem obstajajo besede u, v, w, x, y , da velja

$$\begin{aligned} z &= uvwxy \quad \wedge \\ \wedge |vwx| &\leq n \quad \wedge \\ \wedge |vx| &\geq 1 \quad \wedge \\ \wedge \forall i &\geq 0: uv^iwx^i y \in L \end{aligned}$$



- ◆ **Intuitivno.** Če je L KNJ in $z \in L$ zadosti dolga beseda ($|z| \geq n$), ima z dve podbesedi v in x , ki sta si blizu ($|vwx| \leq n$) in ne obe prazni ($|vx| \geq 1$), ki ju lahko (obe enakokrat) končno mnogokrat ponovimo, a bo dobljena napihnjena beseda še vedno v jeziku L .
- ◆ **Ideja dokaza.** Naj bo G KNG v normalni obliki Chomskega. Z indukcijo po i dokažemo: Če drevo izpeljav besede $z \in L(G)$ nima poti, ki bi bila dalšja od i , potem je $|z| \leq 2^{i-1}$. Naj bo $z \in L(G)$ in $|z| \geq n = 2^k$, kjer je k število vmesnih simbolov v G . Ker je $|z| > 2^{k-1}$, ima vsako drevo izpeljav besede z neko pot dolžine $\geq k+1$. Na tej poti se mora nek vmesni simbol ponoviti.
- ◆ Podobno kot Lemo o napihovanju za regularne jezike, lahko tudi Lemo o napihovanju za KNJ zapišemo s formulo v *predikatnem računu* in iz nje izpeljemo **Metodo za ugotavljanje, ali L ni v KNJ.** (Poskusite za vajo.)

♣ Primer.

- ♣ Naj bo $L = \{a^i b^i c^i \mid i \geq 1\}$. Dokazati želimo, da L ni KNJ.
- ♣ Metoda je podobna kot pri regularnih jezikih.
 - ♣ Naj bo n konstanta iz leme.
 - ♣ Poglejmo besedo $z = a^n b^n c^n$. (Ta beseda z je ‘dobra’, ker je $z \in L$ in $|z| = 3n \geq n$.)
 - ♣ Obstaja več ‘dobrih’ razdelitev z na u, v, w, x, y (tako da je $z = uvwxy$, $|vwx| \leq n$, $|vx| \geq 1$).
 - ♣ Poglejmo, kakšni sta podbesedi v in x v besedi $a^n b^n c^n$.
 - ♣ Ker je $|vwx| \leq n$, opazimo, da vx ne more vsebovati hrati a -jev in c -jev. (Razloži zakaj.)
 - ♣ V vx so **bodisi** samo a -ji **bodisi** samo a -ji in b -ji **bodisi** samo b -ji **bodisi** samo b -ji in c -ji **bodisi** samo c -ji.
 - ♣ Analizirajmo vsako od zgornjih petih možnosti:
 - ♣ Če imata v in x samo a -je, potem ima $uv^0wx^0y = uw$ n b -jev in n c -jev ter manj kot n a -jev (Zakaj? Ker je $|vx| \geq 1$). To pa pomeni, da uv^0wx^0y ni v L !
 - ♣ Če imata v in x samo a -je in b -je, potem ima $uv^0wx^0y = uw$ več c -jev kot a -jev ali b -jev, zato ni v L !
 - ♣ Ostale tri možnosti analiziramo podobno; vsaka do ugotovitve, da uv^0wx^0y ni v L ! (Poskusite za vajo.)
 - ♣ Skladno z metodo, ki izvira iz leme o napihovanju (za KNJ), sledi, da L ni KNJ!
- ♣ S tem primerom smo doazali, da *obstaja formalni jezik, ki ni KNJ!* To pomeni, da *tega jezika ne sprejme noben skladovni avtomat SA!* Zato bomo potrebovali *močnejši model računanja, ki bo sposoben sprejeti ta jezik!*

- ◆ *Ogdenova lema. (Ta prosojnjica ni obvezna.)
- ◆ There are certain non-CFLs for which the pumping lemma is of *no help* (e.g. $L = \{a^i b^j c^i d^j \mid i, j \geq 1\}$). We need a **stronger version of the pumping lemma for CFLs** that will allow us to *focus on some small number of positions* in the word and pump them. (Such an extension is easy for regular languages. The result for CFLs is much harder to obtain.) Here is a *weak* version of the so-called *Ogden's lemma*. Using this lemma we can prove that the above L is *not* CFL.
- ◆ **Ogden's Lemma.** Let L be a CFL. Then there is a constant n (which may be the same as in the pumping lemma for CFL) such that the following holds:
 - if z is any word such that
 - $z \in L \wedge$ we mark any n or more places in z ,
 - then there are words u, v, w, x, y such that
 - $z = uvwxy \wedge$
 - $\wedge vx$ has *at least one* marked place \wedge
 - $\wedge vwx$ has *at most n* marked places \wedge
 - $\wedge \forall i \geq 0: uv^i wx^i y \in L.$
- ◆ **Proof.** Omitted. \square

3.2 Lastnosti zaprtosti za kontekstno-neodvisne jezike

- ◆ Nekatere operacije na KNJ *ohranjajo* kontekstno neodvisnot jezikov.
(tj., delovanje teh operacij na KNJ vrne spet KNJ).
- ◆ *Defiramo*: razred kontekstno-neodvisnih jezikov je **zaprt za dano operacijo**, če je rezultat te operacije pri argumentih, ki so KNJ, spet KNJ.
 - ◆ *Opomba*: Rečemo tudi, da ima ta razred **lastnost zaprtosti** (za dano operacijo)
- ◆ *Definiramo*: Lastnost zaprtosti za dano operacijo je **efektivna**, če obstaja *algoritem*, ki na podlagi končnih opisov KNJ (ki so argumenti operacije) vrne *končni opis* KNJ, ki je rezultat operacije. Zanimale nas bodo le efektivne lastnosti zaprtosti razreda KNJ.
 - ◆ Končni opisi KNJ so npr. konkretni SA, KNG ali ustreznega normalnega oblika Chomskega ali Greiachove.

- ◆ **Izrek.** Razred kontekstno-neodvisnih jezikov je **zaprt** za operacije
 - ◆ unija,
 - ◆ stik,
 - ◆ Kleeneovo zaprtje,
 - ◆ substitucija (in homomorizem),
 - ◆ inverzni homomorfizem.

Dokaz. Opustim. □

- ◆ **Izrek.** Razred kontekstno-neodvisnih jezikov **ni zaprt** za operaciji
 - ◆ presek,
 - ◆ komplement.

Dokaz. Opustim. □

- ◆ **Toda:** Razred kontekstno-neodvisnih jezikov *je zaprt za presek z regularnim jezikom.*
Izrek. Če je L KNJ in R regularen jezik, potem je $L \cap R$ KNJ.

Dokaz. Opustim. □

6.4 Odločitveni problemi in algoritmi za KNJ

- ◆ Zanimajo nas **odločitveni algoritmi** za razne **odločitvene probleme** o KNJ; npr. “Ali je KNJ L prazen? Ali je KNJ L končen? Ali KNJ L vsebuje besedo x ?“ Za naštete primere odločitvenih problemov bomo našli odločitvene algoritme.
- ◆ Seveda obstaja še mnogo drugih odločitvenih problemov o KNJ; na primer, “Ali je *komplement* KNJ L tudi KNJ? Ali je KNJ L *kofiniten*? Ali sta KNG G_1 in G_2 *ekvivalentni*? Ali je KNG G *dvoumna*?“
Videli bomo (pogl.8), da obstajajo odločitveni problemi, ki nimajo odločitvenih algoritmov. Zgornji primeri odločitvenih problemov o KNJ nimajo odločitvenih algoritmov.
- ◆ Vhodni podatki odločitvenih algoritmov so *končni opisi* KNJ (KNG, SA s praznim skl. /končnim st.) Ker lahko efektivno pretvorimo enega v drugega, lahko izberemo najustreznejšega.

◆ Praznот in končnost KNJ.

◆ **Izrek.** Obstajata odločitvena algoritma za ugotavljanje, ali je KNJ

- 1) prazen;
- 2) končen.

◆ **Ideja dokaza.** Naj bo $G = (V, T, P, S)$ KNG.

- ◆ Za ugotavljanje praznosti jezika $L(G)$ moramo ugotoviti, ali S generira vsaj en niz končnih simbolov.
- ◆ Za ugotavljanje končnosti jezika $L(G)$ konstruiramo KNG $G' = (V', T, P', S)$ v normalni obliki Chomskega brez odvečnih simbolov, ki generira jezik $L(G) - \{\varepsilon\}$. (Seveda: $L(G')$ je končen natanko tedaj, ko je $L(G)$ končen.) Nato narišemo usmerjen graf, kjer točke predstavljajo vmesne simbole iz V' , iz točke A v točko B pa obstaja usmerjena povezava, če v P' obstaja produkcija oblike $A \rightarrow BC$ ali $A \rightarrow CB$ (za poljuben vmesni simbol C). Potem velja: $L(G')$ je končen natanko tedaj, ko je graf *acikličen*. Nato preverimo, ali je graf acikličen (za kar obstajajo znani algoritmi (npr. topološka ureditev grafa)).

□

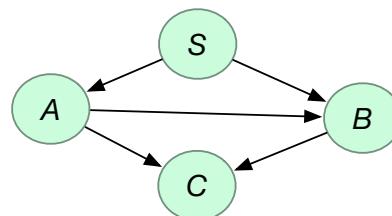
- Primer. Dana je KNG $G_1 = (V, T, P, S) = (\{A, B, C, S\}, \{a, b\}, P, S)$, kjer so v P produkcije

$$S \rightarrow AB$$

$$A \rightarrow BC \mid a$$

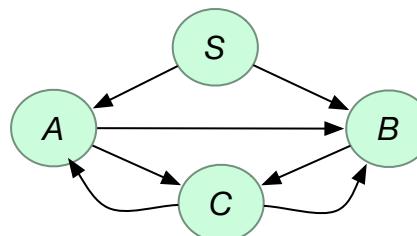
$$B \rightarrow CC \mid b$$

$$C \rightarrow a$$



G_1 je v normalni obliki Chomskega in brez odvečnih vmesnih simbolov. Pripadajoči usmerjeni graf (glej zgoraj) je *acikličen* (nima ciklov), zato je $L(G_1)$ končen.

- Primer. Dodajmo zgornji KNG produkcijo $C \rightarrow AB$. Nova gramatika G_2 je še vedno v normalni obliki Chomskega in brez odvečnih vmesnih simbolov. Pridruženi usmerjeni graf (glej spodaj) je *cikličen* (ciklov je več, npr. $A \rightarrow C \rightarrow A$, $B \rightarrow C \rightarrow B$, $A \rightarrow B \rightarrow C \rightarrow A$), zato je $L(G_2)$ neskončen.



□

◆ **Pripadnost KNJ.**

- ◆ **Definicija.** Problem pripadnosti (za KNG) je vprašanje ``Ali je $x \in L(G)$, kjer je $G = (V, T, P, S)$ dana KNG in $x \in T^*$ dana beseda?''
- ◆ **Vprašanje.** Ali obstaja odločitveni algoritem, ki za poljubno dano KNG G in poljubno dano besedo $x \in T^*$ odgovori na vprašanje ``Ali je $x \in L(G)$?''
- ◆ **Odgovor.** Da, obstaja. Takšen je naslednji (naivni) algoritem:
 1. Pretvori G v normalno obliko Greibachove, G' . /* Opomba: $L(G') = L(G) - \{\varepsilon\}$ */
 2. Če je $x = \varepsilon$, potem preveri ali $S \xrightarrow{G'} * \varepsilon$
sicer /* Zdaj velja: $x \in L(G')$ natanko tedaj, ko $x \in L(G)$. Zato se osredotoči na KNG G' .
Pozor: vsaka produkcija gramatike G' doda nizu, ki se generira, natanko en končni simbol.
Če za x obstaja izpeljava po G' , potem ta izpeljava mora imeti natanko $|x|$ korakov. Nadalje, če ima vsak vmesni simbol (gramatike G') $\leq k$ produkcij, potem obstaja $\leq k^{|x|}$ skrajno levih izpeljav besed dolžine $|x|$. Ali je x med njimi? */

Sistematicno pregleduj eno za drugo skrajno leve izpeljave besed dolžine $|x|$, dokler bodisi ne najdeš izpeljave besede x ali pa si neuspešno pregledal vse možne izpeljave.

- ◆ (nadaljevanje)
 - ◆ Opisani algoritem odgovori z DA/NE za poljubne vhodne podatke G, x .
 - ◆ S stališča izračunljivosti je problem propadnosti za KNJ rešen.
 - ◆ S stališča praktične uporabnosti pa je opisani algoritem časovno neučinkovit, ker lahko zahteva eksponentno mnogo časa glede na dani x (saj v najslabšem primeru, tj. ko $x \notin L(G)$, pregleda vseh $k^{|x|}$ skrajno levih izpeljav besed dolžine $|x|$).
 - ◆ Obstaja učinkovitejši algoritem, t.i. **algoritem CYK** (Cocke-Younger-Kasami), ki
 - ◆ je razvit z metodo *dinamičnega programiranja*,
 - ◆ zahteva $O(|x|^3)$ časa za odgovor na vprašanje $x \in? L(G)$.

◆ *Algoritem CYK (Cocke-Younger-Kasami) (Neobvezno)

- ◆ Nas bo x poljubna beseda dolžine $n \geq 1$ in G poljubna KNG v normalni obliki Chomskega.
- ◆ Označimo z x_{ij} podbesedo besede x dolžine j in začetkom na mestu i . Velja: $1 \leq i \leq n$ in $1 \leq j \leq n-i+1$.
- ◆ Za vsak par i,j in vsak vmesni simbol A želimo ugotoviti, ali velja $A \xrightarrow{G}^* x_{ij}$. Kako? Opazimo naslednje:
 - ◆ [Če $j=1$] V tem primeru je x_{ij} en sam končni simbol. Posledica: $A \xrightarrow{G}^* x_{ij}$ natanko tedaj, ko $A \rightarrow x_{ij}$ je produkcija.
 - ◆ [Če $j > 1$] Zdaj ima x_{ij} vsaj 2 končna simbola. Posledica: $A \xrightarrow{G}^* x_{ij}$ natanko tedaj, ko obstaja produkcija $A \rightarrow BC$ in število k ($1 \leq k \leq j$), tako da B generira prvih k simbolov besede x_{ij} (tj. $B \xrightarrow{G}^* x_{ik}$) in C generira preostalih $j-k$ simbolov besede x_{ij} (tj. $C \xrightarrow{G}^* x_{i+k,j-k}$).
 - ◆ [Če $j = n$] V tem primeru mora to biti podbeseda x_{1n} , kar je x . Posledica: Ugotoviti bo treba, ali $S \xrightarrow{G}^* x_{1n}$.
- ◆ V splošnem je lahko več vmesnih simbolov, ki generirajo x_{ij} ; naj bodo v množici $V_{ij} = \{A \mid A \xrightarrow{G}^* x_{ij}\}$.
Opomba: Pri danem j je lahko $i = 1, 2, \dots, n-j+1$.
- ◆ **Ideja algorima.** Po naraščajočem $j = 1, \dots, n$ računaj množice V_{ij} z upoštevanjem zgornjih treh primerov [$j=1$], [$j>1$], [$j=n$].

- ♦ (nadaljevanje)

begin /* Algoritem CYK

- 1) **for** $i := 1$ to n **do**
 - 2) $V_{i1} := \{A \mid A \rightarrow a \text{ je produkacija } \wedge \text{ } i\text{-ti simbol v } x \text{ je } a\};$
 - 3) **for** $j := 2$ to n **do**
 - 4) **for** $i := 1$ to $n-j+1$ **do**
 - 5) $V_{ij} := \emptyset;$
 - 6) **for** $k := 1$ to $j-1$ **do**
 - 7) $V_{ij} := V_{ij} \cup \{A \mid A \rightarrow BC \text{ je produkacija } \wedge B \in V_{ik} \wedge C \in V_{i+k, j-k}\}$
- endfor**
- endfor**
- end.**

6.5 Slovar

pumping lemma for CFL lema o napihovanju za KNJ Ogden's lemma Ogdenova lemma cofinite kofiniten CYK algorithm algoritem CYK membership problem (for KNG) problem pripadnosti (za KNG)

7

Turingov stroj

Vsebina

- ◆ Uvod
- ◆ Turingov stroj
- ◆ Uporaba Turingovega stroja
- ◆ Različice Turingovega stroja
- ◆ Univerzalni Turingov stroj
- ◆ Prvi osnovni rezultati

7.1 Uvod

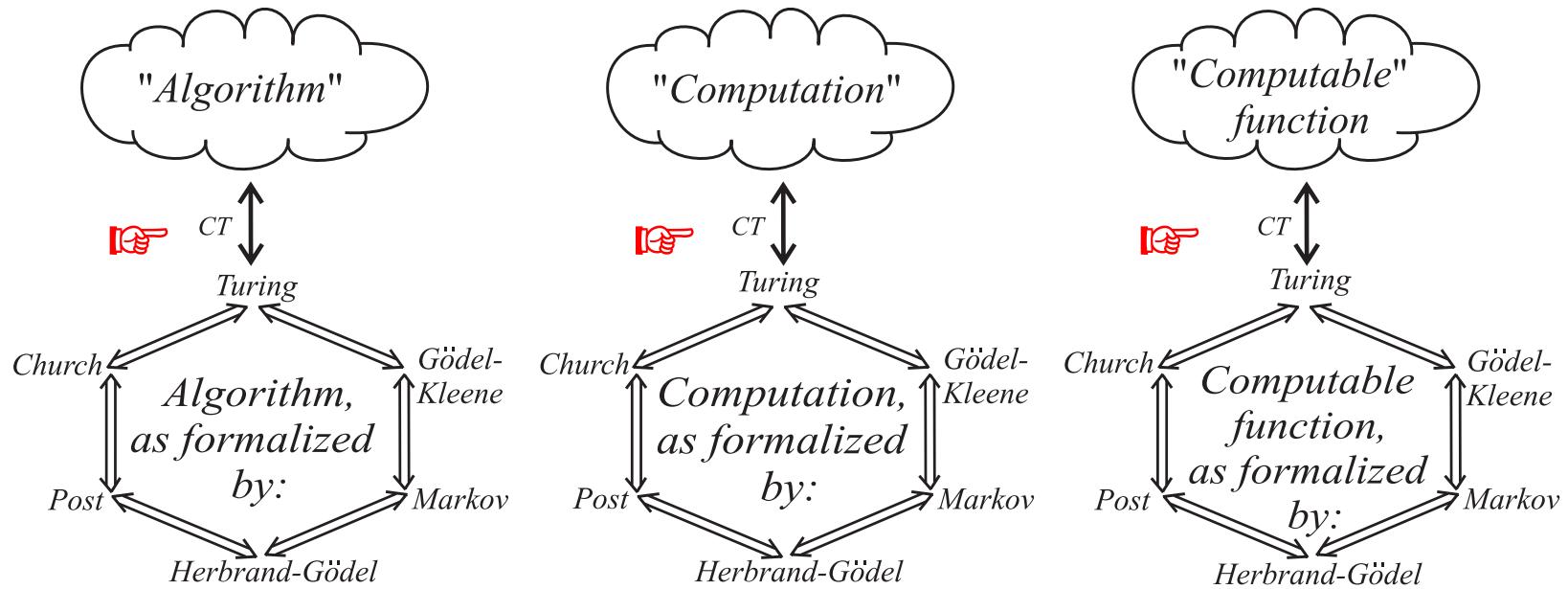
- ◆ **Kaj je algoritem? Kaj je računanje?**
- ◆ Že od Evklida dalje je bil algoritem razumljen *intuitivno*. Znanstveniki so ga razumeli kot *recept, končen spisek ukazov, ki opisuje, kako rešimo dani problem; moramo mu le slepo, mehanično slediti in izvajati njegove ukaze, pa nas bo slejkoprej privедel do rešitve problema.*
- ◆ **Definicija** (algoritem intuitivno) **Algoritem** za reševanje problema je končna množica ukazov, ki vodijo izvajalca, da v končnem številu korakov dobi iz vhodnih podatkov rešitev problema.
- ◆ Dolgo ni bilo potrebe, da bi pojem algoritma definirali strogo, formalno. Zato je ostal *intuitivno, neformalno* definiran vse do 30-ih let prejšnjega stoletja.

- ◆ Potreba po *formalni* definiciji pojma “algoritem” se je pojavila v prvih desetletjih 20. stoletja kot posledica dogodkov v matematiki (matematični logiki). **Kaj se je zgodilo?**
 - ◆ Okrog 1900 je Cantor zasnoval **teorijo množic**. Teorija je bila zelo obetavna, saj se je kazala kot dobra *podlaga* za veliko področij matematike (aritmetika, analiza, ...). Toda Cantor je v svoji teoriji uporabljal matematično *neskončnost* neprevidno. To se je maščevalo: teoriji množic so bila odkrita razna **protislovja**.
 - ◆ Ker jih Cantor s popravki svoje (naivne) teorije ni uspel odpraviti, so to poskušali drugi s **formalno logiko**. Nastale so tri *smeri* poskusov—**intuicionizem**, **logicizem**, **formalizem**—ki so prispevale nove zamisli in orodja, s katerimi so v matematiko vpeljali *jedrnato* in *natančno* izražanje ter *strogost* pri raziskovanju.
 - ◆ **Hilbertov program** je bil obetaven *formalistični* poskus, da bi matematiko rešil protislovij. Žal so ta poskus leta 1933 izničila *Gödelova odkritja o teorijah*, ki so razvite v *formalnih axiomatskih sistemih*.
 - ◆ Toda padec Hilbertovega programa je *preživel* in *ostalo aktualno* vprašanje o **obstoju algoritma** za reševanje nekega zelo **pomembnega problema** iz logike, t.i. *Entscheidungsproblem*. Pri iskanju odgovora pa so se pojavile nove hude težave.

- ◆ **Težava**, je bila tale: Kako naj najdemo odgovor na vprašanje “*Ali obstaja algoritem za reševanje Entscheidungsproblema?*”, če pa sploh ni povsem jasno, kaj je algoritem. Intuitivna definicija algoritma ni zadoščala. Poglejmo zakaj:
 - ◆ Da bi dokazali, da algoritem za Entscheidungsproblem obstaja, bi zadoščalo, če bi konstruirali en sam recept (in dokazali, da *reši* naš problem).
 - ◆ Da bi dokazali, da algoritem za Entscheidungsproblem ne obstaja, bi morali zavrniti vsak recept (tako, da bi dokazali, da *ne reši* našega problema).
- Toda receptov je *neskončno* mnogo! Kako dokazati, da *nobeden* ne reši našega problema?
- Odgovor:** Da bi to dokazali, bi morali imeti na voljo nekaj, kar bi vsebovalo:
- 1) *formalno karakterizacijo pojma "algoritem"*, tj. natančno definirano *lastnost*, ki bi jo imeli *vsi algoritmi* in nič drugega kot algoritmi;
 - 2) *formalno definicijo realističnega okolja*, ki bi bilo sposobno izvesti vsak (tako karakteriziran) algoritem;
 - 3) *formalno definicijo izvajanja* (tako karakteriziranih) algoritmov v tem okolju.
- Temu ”nekaj” rečemo **model računanja**.
- Če bi imeli model računanja, bi z njim lahko izločili neskončno mnogo kandidatov za iskani algoritem.*
- ◆ Potreba po modelu računanja je postala jasna.
- Definicija.** (model računanja) **Model računanja** formalno karakterizira osnovne pojme algoritmičnega računanja, tj. algoritem, njegovo okolje in njegovo izvajanje v tem okolju.

- ◆ V 30-ih 20.stoletja se je pričelo *iskanje* modela računanja. Potekalo je v raznih smereh. Sčasoma je bilo predlaganih kar nekaj modelov:
 - ◆ μ -*rekurzivne funkcije* (Kurt Gödel, Stephen Kleene)
 - ◆ *splošne rekurzivne funkcije* (Jacques Herbrand, Kurt Gödel)
 - ◆ λ -*račun* (Alonzo Church)
 - ◆ *Turingov stroj* (Alan Turing)
 - ◆ *Postov stroj* (Emil Post)
 - ◆ *Markovski algoritmi* (Andrej Markov Jr.)
- ◆ Modeli se *povsem razlikujejo*, zato se je postavilo vprašanje:
Kateri model (če sploh kateri) je “pravi” ali vsaj najbolj “ustrezen” ?
- ◆ Večina raziskovalcev se je ogrela za *Turingov stroj*, ker je z najbolj popolno in razumljivo *utemeljitvijo* definiral osnovne pojme algoritmičnega računanja.
- ◆ Presenetljivo so raziskovalci kmalu *dokazali*, da so našteti modeli *ekvivalentni* :
Kar zmore izračunati eden, zmorejo tudi ostali.

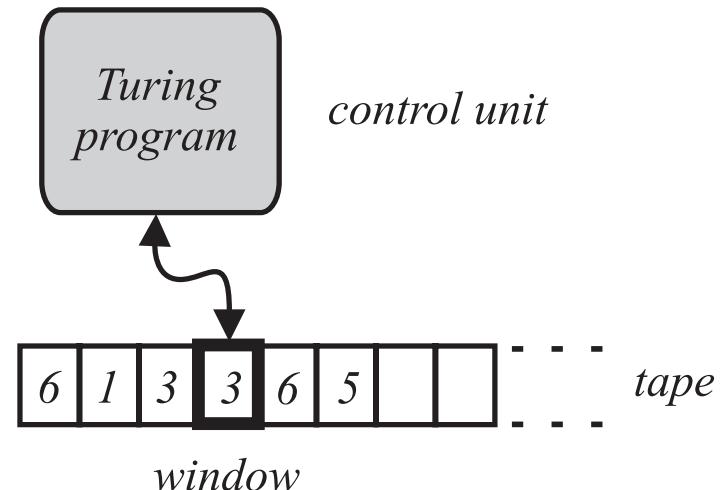
- ◆ Kaj pa naše **intuitivno** razumevanje osnovnih pojmov računanja?
Ali obstaja kaka povezava med človekovim *intuitivnim razumevanjem osnovnih pojmov algoritmičnega računanja* (“algoritem”, “računanje”, “izračunljiva funkcija”) in formalnimi *modeli računanja*?
- ◆ Da. Ker so dokazali, da so znani modeli računanja *ekvivalentni* (čeprav *povsem različni*), je bila postavljena naslednja trditev (*teza*):
Teza o izračunljivosti (tudi **Church-Turingova teza**). *Intuitivni osnovni pojmi algoritmičnega računanja so ustrezno formalizirani* (strogo definirani) takole:
 - ◆ intuitivni pojem “*algoritma*” formalizira *Turing program*;
 - ◆ intuitivni pojem “*računanja*” formalizira *izvajanje Turingovega programa*;
 - ◆ intuitivni pojem “*izračunljive funkcije*” formalizira *funkcija, izračunljiva s Turingovim programom*.
- ◆ Tezo je sprejela večina raziskovalcev, danes jo sprejemajo vsi (doslej je nihče ni uspel ovreči).



Teza o izračunljivosti je vzpostavila most (zvezo) med intuitivnim razumevanjem osnovnih pojmov "algoritmom," "računanje" in "izračunljivost" in matematičnim, formalno definiranim razumevanjem teh pojmov s pomočjo modelov računanja.
 Teza je odprla vrata *matematični obravnavi* intuitivnih osnovnih pojmov računanja.

7.2 Turingov stroj

- Modela KA in SA sta nekako *omejena*: oba lahko le *bereta* vhodne simbole *zaporedoma* od začetka vhodne besede do njenega konca.
- Tudi **Turingov stroj (TS)** ima trak z oknom in nadzorno enoto s programom.
- Toda Turingov stroj lahko bere in **piše** simbole v celice **kjerkoli** na **potencialno neomejenem** traku.



- ◆ **Definicija.** (Turingov stroj) **Turingov stroj** (*osnovna različica*) ima naslednje enote: *nadzorno enoto*, v kateri je *Turingov program*; *trak*, ki ga sestavljajo *celice*; premično *okno* na traku, ki je povezano z nadzorno enoto.
- ◆ Podrobnosti so:
 - ◆ **Trak** služi *pisanju* in *branju* vhodnih podatkov ter vmesnih in končnih rezultatov. Razdeljen je na enake **celice** in je *potencialno neskončen* v eno (desno) smer; tj. vsakokrat, ko je to potrebno, se (samodejno) *podaljša* s *končno mnogo celicami*. Vsaka celica vsebuje nek **tračni simbol** iz **tračne abecede** $\Gamma = \{z_1, \dots, z_t\}$, $t \geq 3$. Simbol z_t je poseben: označuje, da je celica *prazna*. Običajno ga pišemo kot \sqcup in imenujemo simbol za **prazen prostor**. Poleg \sqcup sta v tračni abecedi še vsaj dva simbola: 0 in 1. (Ne da bi se s tem omejili, se dogovorimo, da je $z_1 = 0$ in $z_2 = 1$.) *Vhodni podatki* so zapisani v **vhodni besedi**; ta je sestavljena iz simbolov **vhodne abecede** Σ , za katero velja $\{0,1\} \subseteq \Sigma \subseteq \Gamma - \{\sqcup\}$. Torej Σ zagotovo vsebuje vsaj 0 in 1, in zagotovo ne vsebuje simbola \sqcup . Sprva je vhodna beseda vpisana v skrajno levih celicah (tj. na začetku traku), vse ostale celice traku pa so prazne (v vsaki je \sqcup).

- Nadzorna enota je vedno v kakem **stanju** iz končne množice stanj $Q = \{q_1, \dots, q_s\}$, $s \geq 1$. q_1 je **začetno stanje**. Nekatera stanja so **končna**; zbrana so v množici $F \subseteq Q$; ostala stanja so **nekončna**. Kadar indeks stanja ni pomemben, včasih uporabimo tudi oznaki q_{da} oz. q_{ne} za označevanje končnega oz. nekončnega stanja.

V nadzorni enoti je **Turingov program (TP)**, ki usmerja delovanje enot TS. Turingov program P je *značilen* za izbrani TS, tj. različni TS imajo različne TP. TP je *parcialna funkcija* $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$, imenovana **funkcija prehodov**. **Pozor.** TS je po definiciji **determinističen**: za vsak par (stanje, tračni simbol) ima na voljo kvečjemu eno potezo.

TP δ lahko predstavimo s *tabelo* $\Delta = Q \times \Gamma$, kjer velja:

- $\Delta[q_i, z_r] = (q_j, z_w, D)$ če $\delta(q_i, z_r) = (q_j, z_w, D)$ je ukaz v δ ;
- $\Delta[q_i, z_r] = 0$ če $\delta(q_i, z_r) \uparrow$ (*nedefinirano*).

Brez zmanjšanja splošnosti lahko predpostavimo, da iz q_{ne} vedno obstaja prehod, iz q_{da} pa ne.

Δ	z_1	z_2	\cdots	z_r	\cdots	z_t
q_1	•	•	...	•	...	•
q_2	•	•	...	•	...	•
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots
q_i	•	•	...	(q_j, z_w, D)	...	
\vdots	\vdots	\vdots		\vdots	\ddots	\vdots
q_s	•	•	...	•	...	•

- **Okno** se lahko premakne (korakoma, preko vmesnih celic) na *poljubno* celico. Nadzorna enota lahko iz celice pod oknom *prebere* simbol, v celico pod oknom pa lahko tudi *vpiše* simbol (tj. *zamenja* prejšnjega z novim). V *enem* koraku se okno lahko premakne le na *sosednjo* celico.

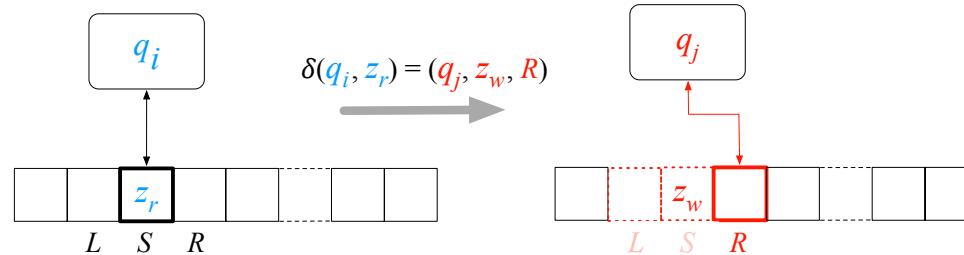
- ◆ Pred zagonom TS velja naslednje:
 - ◆ a. vhodna beseda je zapisana na začetku traku (tj. v skrajno levih celicah);
 - ◆ b. okno je premaknjeno na začetek traku (tj. nad skrajno levo celico);
 - ◆ c. nadzorna enota se nahaja v začetnem stanju.
- ◆ Odslej naprej TS deluje samostojno, mehanično in korakoma kot mu narekuje TP. Če je TS v stanju $q_i \in Q$ in vidi v celici pod oknom simbol $z_r \in \Gamma$, potem:

if q_i je končno stanje, **then** TS se ustavi;

else, if $\delta(q_i, z_r) \uparrow$ (tj. TP nima ukaza za par q_i, z_r) **then** TS se ustavi;

else, if $\delta(q_i, z_r) \downarrow = (q_j, z_w, D)$ **then** TS storii naslednje:

 - preide v stanje q_j ;
 - vpiše z_w skozi okno (v celici po oknom zamenja simbol z_r s simbolom z_w);
 - Premakne okno na sosednjo celico v smer D (levo (če $D=L$); desno (če $D=R$)) ali pa pusti okno, tam kjer je (stoj (če $D=S$)).



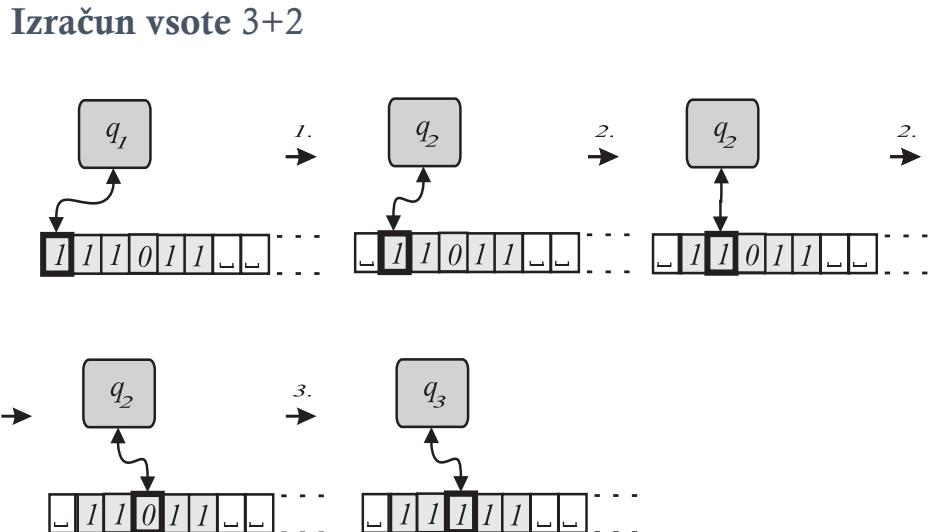
- ◆ V splošnem je TS sedmerka $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$. Konkretni TS ima podane konkretnie $Q, \Sigma, \Gamma, \delta, F$. **(konec definicije)**

- ◆ **Primer.** Sestavimo TS T , ki bo izračunal vsoto $m+n$ naravnih števil m in n . Vhodna podatka m,n naj bosta zapisana unarno v *vhodni besedi* $1^m 0 1^n$. Preden se T ustavi, naj na traku vrne rezultat $m+n$ v besedi 1^{m+n} . (Npr., pri vh.besedi 111011 naj vrne besedo 11111.)
- ◆ **Algoritem (zamisel).** Če je prvi simbol vh.besede 1, ga izbriši (ukaz 1) in premikaj okno v desno čez simbole 1 (ukaz 2), dokler ne prebereš simbola 0. Nato zamenjaj ta simbol s simbolom 1 in se ustavi (ukaz 3). Posebnost: če je prvi simbol vhodne besede 0, ga izbriši in se ustavi (ukaz 4).
- ◆ **Turingov stroj T**

$T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$, kjer so:

- ◆ $Q = \{q_1, q_2, q_3\}$
- ◆ $\Sigma = \{0, 1\}$
- ◆ $\Gamma = \{0, 1, \sqcup\}$
- ◆ $F = \{q_3\}$
- ◆ δ ima naslednje ukaze:

1. $\delta(q_1, 1) = (q_2, \sqcup, R)$
2. $\delta(q_2, 1) = (q_2, 1, R)$
3. $\delta(q_2, 0) = (q_3, 1, S)$
4. $\delta(q_1, 0) = (q_3, \sqcup, S)$



◆ **Primer.** Sestavimo drugačen TS T' , ki bo izračunal vsoto $m+n$ naravnih števil m in n .

◆ **Algoritem (zamisel).** Premikaj okno v desno, dokler ne prebereš \sqcup . Premakni okno na levo sosednjo celico (na zadnji simbol vhodne besede) in izbrisši simbol v njej. Če je bil izbrisani 0, se ustavi, sicer premikaj okno v levo dokler ne prebereš 0. Namesto 0 vpiši 1 in se ustavi.

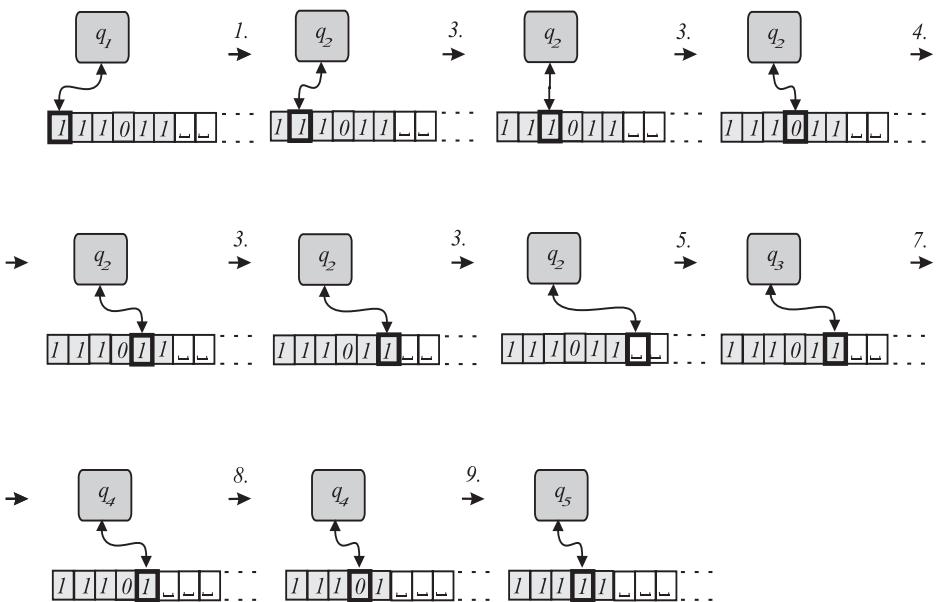
◆ Turingov stroj T'

$T' = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$, where:

- ◆ $Q = \{q_1, q_2, q_3, q_4, q_5\}$
- ◆ $\Sigma = \{0, 1\}$ $\Gamma = \{0, 1, \sqcup\}$ $F = \{q_5\}$
- ◆ δ has the following instructions:

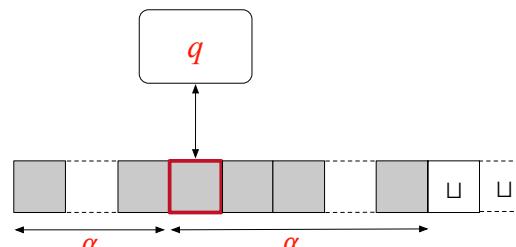
1. $\delta(q_1, 1) = (q_2, 1, R)$
2. $\delta(q_1, 0) = (q_2, 0, R)$
3. $\delta(q_2, 1) = (q_2, 1, R)$
4. $\delta(q_2, 0) = (q_2, 0, R)$
5. $\delta(q_2, \sqcup) = (q_3, \sqcup, L)$
6. $\delta(q_3, 0) = (q_5, \sqcup, S)$
7. $\delta(q_3, 1) = (q_4, \sqcup, L)$
8. $\delta(q_4, 1) = (q_4, 1, L)$
9. $\delta(q_4, 0) = (q_5, 1, S)$

Izračun vsote $3+2$



Definicije.

- Če je Turingov stroj T v danem trenutku videti kot



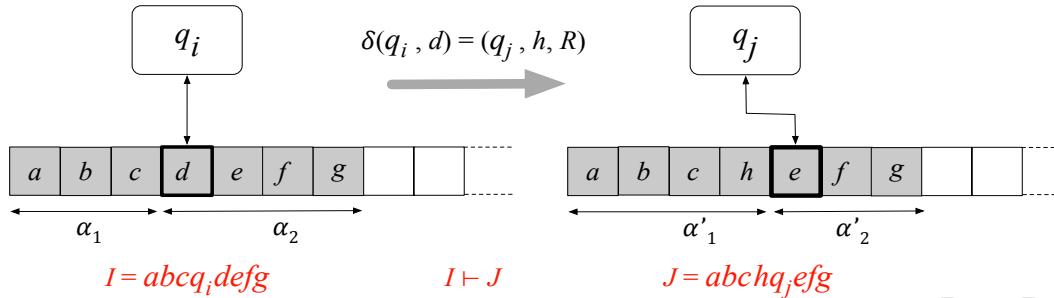
Beseda α_2 se začne s simbolom pod oknom in konča s skrajno desnim nepraznim simbolom.

se beseda $I = \alpha_1 q \alpha_2$ imenuje **trenutni opis (TO)** Turingovega stroja T .

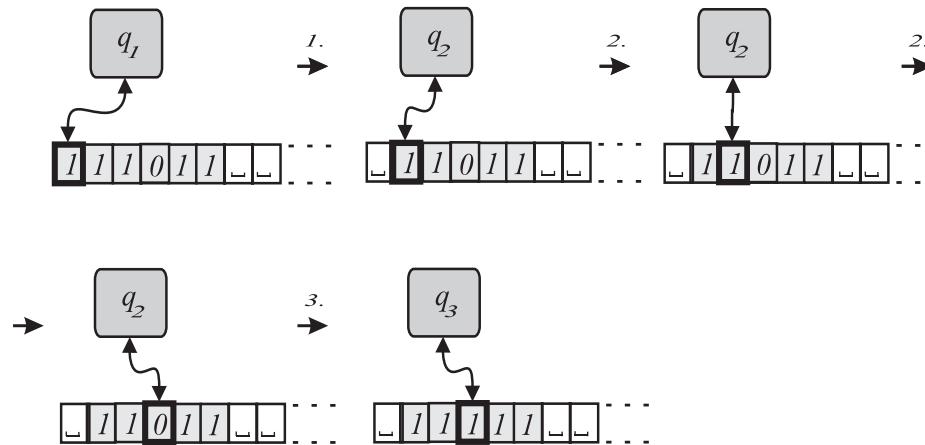
Inuitivno: TO je "trenutna slika enot Turingovega stroja" (angl. snapshot) med zaporednima ukazoma.

- TO I **neposredno preide** v TO J -- kar zapišemo $I \vdash J$ -- če v TP stroja T obstaja ukaz, katerega izvedba spremeni I v J . Refleksivno tranzitivno zaprtje relacije \vdash je relacija \vdash^* ; če velja $I \vdash^* J$, rečemo, da TO I **preide** v TO J .

- Primer.**



- ◆ **Primer.** Denimo, da je zaporedje trenutnih slik nekega Turingovega stroja, ko izvaja svoj program δ , naslednje:



- ◆ **Izračun**, ki ga Turingov stroj pri tem opravi, je opisan z zaporedjem trenutnih opisov

$$q_1111011 \vdash \square q_211011 \vdash \square 1 q_21011 \vdash \square 11 q_2011 \vdash \square 11 q_3111$$

Intuitivno. Izračune in računanje lahko predstavimo z zaporedji trenutnik opisov.

7.3 RabaTuringovega stroja

Osnovne računske naloge, ki jih izvajajo Turingovi stroji, so tri :

- Računanje vrednosti funkcij

``Za dano funkcijo φ in argumente a_1, \dots, a_k , izračunaj $\varphi(a_1, \dots, a_k)$.''

- Razpoznavanje množic

``Za dano množico S in objet x ugotovi, ali velja ali ne velja $x \in S$.''

- Generiranje množic

``Generiraj zaporedje x_1, x_2, x_3, \dots , ki ima natančno elemente dane množice S .''

- ◆ **Računanje vrednosti funkcij s TS.**
- ◆ Vsak TS T inducira, za vsak $k \geq 1$, funkcijo $\varphi_T^{(k)}$, ki preslika k besed v besedo. Definirajmo funkcijo $\varphi_T^{(k)}$.

Definicija. Bodi $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ TS in $k \geq 1$. **k -mestna lastna funkcija** stroja T je parcialna funkcija $\varphi_T^{(k)} : (\Sigma^*)^k \rightarrow \Sigma^*$, definirana kot sledi:

Če vhodne podatke stroja T sestavlja k besed $u_1, \dots, u_k \in \Sigma^*$, potem je vrednost funkcije $\varphi_T^{(k)}$ pri argumentnih u_1, \dots, u_k podana s

$$\varphi_T^{(k)}(u_1, \dots, u_k) := \begin{cases} \nu, & \text{če se } T \text{ ustavi} \wedge \text{vrne na traku besedo } \nu \wedge \nu \in \Sigma^*; \\ \uparrow, & \text{če se } T \text{ ne ustavi} \vee \text{na traku ne vrne besede iz } \Sigma^*. \end{cases}$$

- ◆ Interpretacija argumentov u_1, \dots, u_k in rezultata ν je odvisna od področja uporabe. Npr., če interpretiramo besede u_1, \dots, u_k kot kode naravnih števil n_1, \dots, n_k , potem lahko interpretiramo $\varphi_T^{(k)}$ kot aritmetično funkcijo $(\mathbb{N}^k \rightarrow \mathbb{N})$, katere vrednost $\varphi_T^{(k)}(u_1, \dots, u_k)$ je koda ν nekega naravnega števila m .

- ◆ V praksi pogosto naletimo na *obratno nalog*:

``Za dano k -mestno funkcijo $\varphi : (\Sigma^*)^k \rightarrow \Sigma^*$ najdi TS T , ki računa njene vrednosti.''

Drugače: za dano k -mestno funkcijo φ konstruiraj TS T , da bo $\varphi(u_1, \dots, u_k) = \varphi_T^{(k)}(u_1, \dots, u_k)$.
- ◆ Izkazalo se je tole: *Ovisno od funkcije φ je, v kolikšni meri je TS* (model računanja) *zmožen računati njene vrednosti.* Glede na to razlikujemo *tri vrste funkcij.* Intuitivno povedano je funkcija φ :
 - ◆ **izračunljiva**, če obstaja TS T , ki zmore **izračunti** vrednost φ **pri poljubnih argumentih**;
 - ◆ **parcialna izračunljiva**, če obstaja TS T , ki zmore **izračunati** vrednost φ povsod, kjer je φ **definirana**;
 - ◆ **neizračunljiva**, če *ne obstaja* TS T , ki bi bil zmožen **izračunati** vrednost φ povsod, kjer je φ **definirna**.

Definicija. Naj bo $\varphi : (\Sigma^*)^k \rightarrow \Sigma^*$ funkcija. Tedaj:

- ◆ φ je **izračunljiva**, če \exists TS, ki izračuna φ povsod na $\text{dom}(\varphi) \wedge \text{dom}(\varphi) = (\Sigma^*)^k$;
- ◆ φ je **parcialna izračunljiva (p.i.)**, če \exists TS, ki izračuna φ povsod na $\text{dom}(\varphi)$;
- ◆ φ je **neizračunljiva**, če $\neg \exists$ TS, ki izračuna φ povsod na $\text{dom}(\varphi)$.

Opomba. Izračunljiva funkcija je p.i. funkcija, ki je celo totalna.

- ◆ **Razpoznavanje množic s TS.**
- ◆ Vsak TS T sprejme nek jezik.

Definicija. Bodi $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ TS in $w \in \Sigma^*$ beseda. T sprejme w , če $q_1 w \vdash^* \alpha_1 p \alpha_2$ za neko stanje $p \in F$ in $\alpha_1 \alpha_2 \in \Gamma^*$. **Jezik, ki ga sprejme** TS T , je množica $L(T) = \{w \mid w \in \Sigma^* \wedge T \text{ sprejme } w\}$.

Torej: Beseda je sprejeta, če se T , ko jo prebere, znajde v kakem *končnem* stanju.

Jezik, ki ga T sprejme, pa je množica vseh takih besed.

- ◆ Interpretacija jezika $L(T)$ je poljubna, odvisna od uporabe TS. Npr., če besede $w \in \Sigma^*$ interpretiramo kot kode naravnih števil, lahko $L(T)$ interpretiramo kot podmnožico množice \mathbb{N} .

- V praksi pogosto naletimo na *obratno nalog*:
 ``Za dano množico $S \subseteq \Sigma^*$ najdi TS T , ki sprejme S .''
Povedano drugače: za dano množico S konstruiraj TS T , da bo $S = L(T)$.
- Izkazalo se je tole: *Odvisno od množice S je, v kolikšni meri je TS (model računanja) zmožen razpoznavati njene pripadnike.* Glede na to ločimo *tri vrste množic*. Intuitivno povedano je množica S :
 - **odločljiva**, če obstaja TS T , ki zmore **odgovoriti** DA/NE na vpr. ``Ali je $x \in S?$ '' za **poljuben** x ;
 - **polodločljiva**, če obstaja TS T , ki zmore **odgovoriti** DA na vpr. ``Ali je $x \in S?$ '' za **poljuben** $x \in S$;
 - **neodločljiva**, če ne obstaja TS T , ki zmore **odgovoriti** DA/NE na vpr. ``Ali je $x \in S?$ '' za **poljuben** x .

Definicija. Naj bo $S \subseteq \Sigma^*$ jezik. Potem:

- S je **odločljiv**, če \exists TS, ki odgovori DA/NE na vpr. ``Ali je $x \in S?$ '' za poljuben $x \in \Sigma^*$;
- S je **polodločljiv**, če \exists TS, ki odgovori DA na vpr. ``Ali je $x \in S?$ '' za poljuben $x \in S$;
- S je **neodločljiv**, če $\neg\exists$ TS, ki bi odgovoril DA/NE na vpr. ``Ali je $x \in S?$ '' za poljuben $x \in \Sigma^*$.

- ◆ Kaže, da pri nekaterih S ne moremo algoritmično odločiti vpr. **Ali je $x \in S$?**
Zakaj?

Naj b $S = L(T)$ polodločljiv in $x \in \Sigma^*$ poljubna vhodna beseda. Potem:

- ◆ če x je v S , potem se bo T gotovo ustavil (in odgovoril DA).
- ◆ če x ni v S , potem se lahko zgodi, da se T ne bo ustavil (in zato nikoli odgovoril NE).
Zakaj? Če v resnici $x \notin S$, definicija polodločljivega S ne določa, kaj T v tem primeru odgovori oz. počne.
Torej se lahko ustavi in odgovori NE, ali pa se tudi nikoli ne ustavi. (Vedno predpostavljam, da ne laže.)

Torej: dokler T računa, ne moremo vedeti

- ◆ ali se bo T nekoč ustavil (če ga pustimo računati) in odgovoril DA ali NE,
- ◆ ali pa bo T računal brez prestanka in zato nikoli odgovoril (niti DA niti NE)

Povedano drugače:

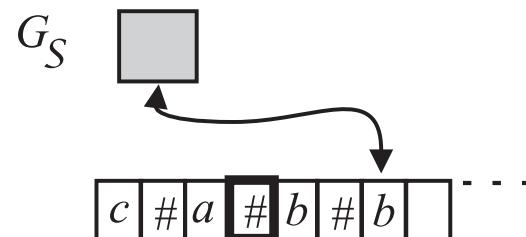
- ◆ Če v resnici $x \in S$, potem se bo T zagotovo ustavil in odgovoril DA.
- ◆ Če v resnici $x \notin S$, potem se T
 - ◆ bodisi ustavi in odgovori NE;
 - ◆ bodisi nikoli ne ustavi in zato nikoli nič ne odgovori. (Dokler računa ne vemo, kaj se bo zgodilo.)

► **Generiranje množic s TS**

- Nekateri TS T generirajo jezike.
- **Definicija.** Naj bo $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ TS. Pravimo, da je T **generator**, če na trak izpiše zaporedje (nekih) besed iz Σ^* , razmejenih s simbolom $\# \in \Gamma - \Sigma$. **Jezik, ki ga generira** T , je množica $G(T) = \{w \mid w \in \Sigma^* \wedge T \text{ izpiše } w\}$.

Beseda w je v $G(T)$, če jo T izpiše ali bi jo slejko prej izpisal (tj. po končno mnogo korakih od začetka generiranja). Vrstni red izpisanih besed ni predpisan. Beseda je lahko izpisana večkrat.

- **Primer.** Besede c, a, b so v jeziku $G(T) = \{c, a, b, \dots\}$, ki ga generira T .



- ◆ Zopet v praksi pogosto naletimo na *obratno nalog*:
``Za dano množico $S \subseteq \Sigma^*$ najdi TS T , ki generira natanko njene elemente.''
Povedano drugače: za dano množico S konstruiraj TS T , da bo $S = G(T)$.
- ◆ **Ugotovitev.** Nekatere množice se dajo generirati, druge pa ne.
Primeri. Množico \mathbb{N} naravnih števil se dá generirati: $1\#2\#3\#\dots n\#(n+1)\#\dots$ Tudi množico \mathbb{Z} celih števil se dá generirati : $0\#1\#-1\#2\#-2\#3\#-3\#\dots n\#-n\#(n+1)\#-(n+1)\#\dots$ Prav tako tudi množici \mathbb{P} (praštevila) in \mathbb{Q} (racionalna števila). (**Vaja: razmisli kako.**) Toda množice \mathbb{R} realnih števil se ne dá generirati; tudi množice $[0,1]$ realnih števil, ki so med 0 in 1, se ne dá generirati.
Očitno je: Množice, ki niso števne, se ne dajo generirati. Kaj pa števne množice?
- ◆ **Vprašanje:** *Kdaj se števna množica S dá algoritmično generirati ? Kdaj se vsi njeni elementi dajo zaporedoma izpisati z nekim algoritmom (Turingovim strojem), tako da se bo vsak element množice S (in nič drugega) slejkoprej pojavil v tem zaporedju?*
Opomba. Če je taka S neskončna, se ne bo nikoli izpisala v celoti, toda vsak njen posamični element se bo izpisal v končnem času.

◆ Izračunljivo preštevni (i.p.) jeziki (množice)

- ◆ Denimo, da se množica S da generirati. Torej se bo *vsak njen posamični element slejkoprej izpisal*. Izberimo poljuben $x \in S$. Recimo, da se x (prvič) izpiše kot n -ti po vrsti, kjer je $n \in \mathbb{N}$. Torej lahko govorimo o prvem, drugem, tretjem, ... n -tem izpisanim (generiranem) elementu množice S . Torej vsakemu elementu množice S pripada natanko eno naravno število (njegova zaporedna številka v zaporedju). Pravimo, da se S dá **oštevilčiti**. (enumerate)
- ◆ Zanimajo nas množice S , ki se dajo algoritmično generirati, tj. generirati z nekim TS. Pravimo, da so take množice *izračunljivo preštevne*. (angl. *computably enumerable*, kratko c.e.)

Definicija. Množica S je **izračunljivo preštevna (i.p.)**, če obstaja TST, da je $S = G(T)$.

Povedano drugače: S je izračunljivo preštevna, če jo lahko generira kak Turingov stroj.

◆ **Izrek.** S je izračunljivo preštevna natanko tedaj, ko S je polodločljiva.

Dokaz. Izpustim (glej mojo knjigo). □

- ◆ **Povzetek.** Doslej smo *definirali* že nekaj vrst formalnih jezikov (oz. množic): *regularne, kontekstno-neodvisne, odločljive, polodločljive (i.p.), neodločljive jezike*. Obstoj *regularnih* in *kontekstno-neodvisnih* jezikov smo že dokazali, obstoj ostalih pa bomo dokazali v nadaljevanju. Dokazali bomo tudi, da med razredi teh jezikov veljajo naslednji odnosi.

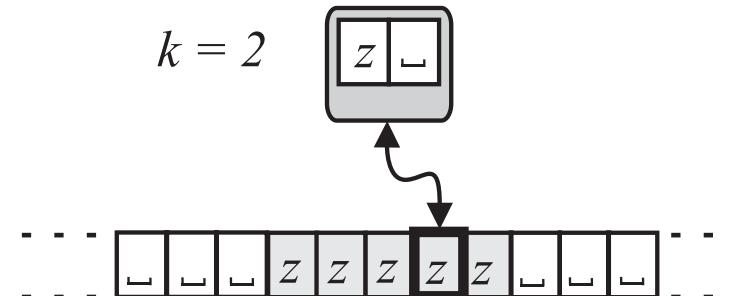


7.4 Različice Turingovega stroja

- ◆ Med razlogi za sprejetje Turingovega stroja kot *splošnega modela računanja* je tudi ta, da je *osnovni* model TS **ekvivalenten** vsem različicam TS (čeprav so te po svoji računski moči videti močnejše). V nadaljevanju bomo različice opisali in nakazali, kako dokažemo, da so ekvivalentne osnovnemu TS. Vsako različico dobimo z neko posplošitvijo osnovnega TS. Različice so:
 - ◆ TS s končni pomnilnikom
 - ◆ TS z večslednim trakom
 - ◆ TS z neomejim trakom
 - ◆ Večtračni TS
 - ◆ TS z večdimenzionalnim trakom
 - ◆ Nedeterministični TS

TS s končnim pomnilnikom.

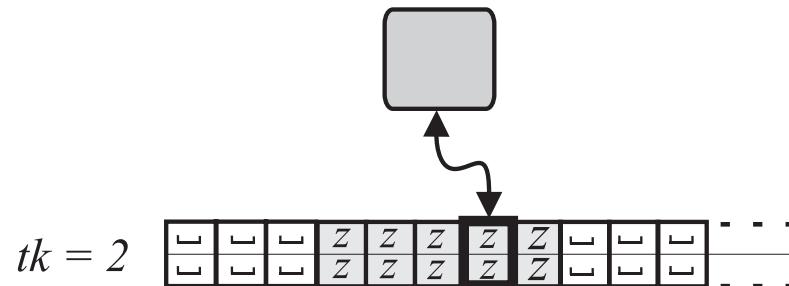
- Ta različica (varianta) V ima v nadzorni enoti končno velik pomnilnik, ki lahko hrani $k \geq 1$ tračnih simbolov in uporablja med računanjem. Turingov program (TP) je zato funkcija $\delta_V : Q \times \Gamma \times \Gamma^k \rightarrow Q \times \Gamma \times \{L, R, S\} \times \Gamma^k$.
- Primer. Pri $k = 2$ je V videte takole:



- Čeprav se zdi, da je V močnejši od osnovnega modela T , ni tako; T lahko izračuna vse, kar lahko izračuna V . Dokaz (ideja): pokažemo, da T lahko simulira vsako potezo stroja V . (Obratno je trivialno, saj je T poseben primer $V(k=0)$.) \square

TS z večslednim trakom.

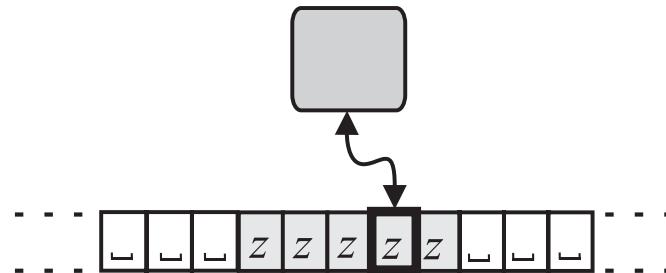
- Ta različica V ima trak razdeljen na $tk \geq 2$ vzporednih sledi (angl. tracks). Na vsaki sledi so simboli tračne abecede Γ . Okno vidi tk -terko simbolov, po enega na vsaki sledi. Turingov program je $\delta_V : Q \times \Gamma^{tk} \rightarrow Q \times \Gamma^{tk} \times \{L, R, S\}$.
- Primer. Pri $tk = 2$ je V videti takole:



- Čeprav se zdi, da je V močnejši od osnovnega modela T , ni tako; T lahko izračuna vse, kar lahko izračuna V . **Dokaz** (ideja): pokažemo, da T lahko simulira vsako potezo stroja V . (Obratno je trivialno: T je poseben primer $V(tk = 1)$.) \square

TS z neomejenim trakom.

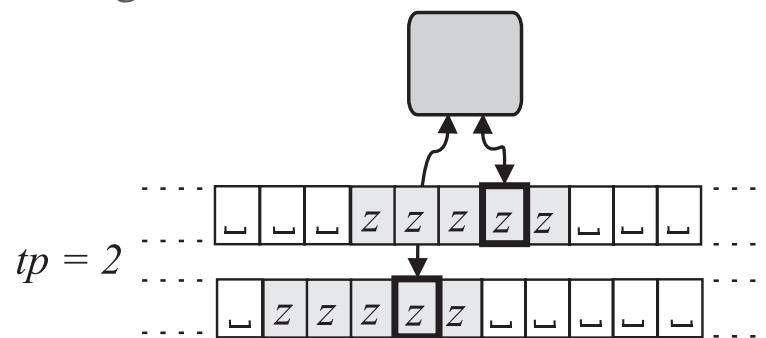
- Ta različica V ima trak, ki je (potencialno) neomejen v obe smeri. Turingov program stroja V je funkcija $\delta_V : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$.



- Spet se zdi, da je V močnejši od osnovnega modela T . Vendar ni tako: T lahko izračuna vse, kar lahko izračuna V . **Dokaz** (ideja): pokažemo, da T lahko simulira vsako potezo stroja V . (Obratno je trivialno, saj je T poseben primer V). □

Večtračni TS.

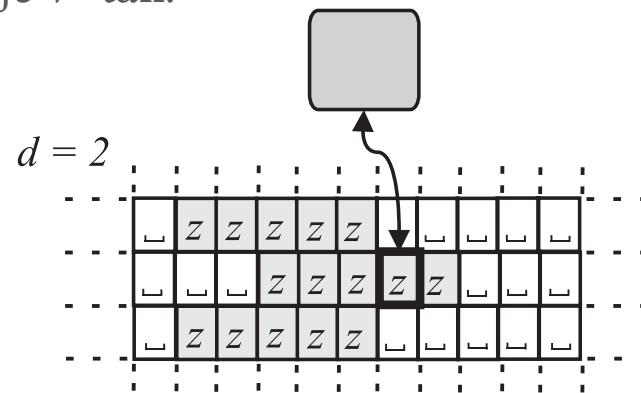
- Ta varianta V ima $tp \geq 2$ neomejenih trakov. Vsak trak ima svoje okno, ki se lahko premika neodvisno od ostalih. TP je $\delta_V : Q \times \Gamma^{tp} \rightarrow Q \times (\Gamma \times \{L, R, S\})^{tp}$.
- Primer. Za $tp = 2$ izgleda V tako:



- Tudi ta različica ni močnejša od osnovnega modela T : T lahko izračuna vse, kar lahko izračuna V . **Dokaz** (ideja): pokažemo, da T lahko simulira vsako potezo stroja V . (Obratno je trivialno, saj je T poseben primer V ($tp = 1$).) \square

TS z večdimenzionalnim trakom.

- Ta različica V ima d -dimenzionalen trak, $d \geq 2$. Okno se lahko premika v d dimenzijah, tj. $2d$ smereh $L_1, R_1, L_2, R_2, \dots, L_d, R_d$. Turingov program je funkcija $\delta_V : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L_1, R_1, L_2, R_2, \dots, L_d, R_d, S\}$.
- Primer. Za $d = 2$ je V tak:



- Ali je morda ta različica močnejša od osnovnega TS T ? Ne. T lahko izračuna vse, kar lahko izračuna V . Dokaz (ideja): pokažemo, da T lahko simulira vsako potezo stroja V . (Obratno je trivialno, saj je T poseben primer $V(d = 1)$.) \square

Nedeterministični TS.

- Ta V ima Turingov program δ , ki vsakemu paru priredi (q_i, z_r) končno množico možnih prehodov $\{(q_{j_1}, z_{w_1}, D_1), (q_{j_2}, z_{w_2}, D_2), \dots\}$, stroj V (nadzorna enota) pa izbere enega med njimi.
- Kako V izbere enega izmed možnih prehodov?

Privzamemo, da ima V čudežno sposobnost, da med več možnimi prehodi vedno izbere pravega (če je kakšen tak), tj. tistega, ki ga vodi k uspehu (npr. sprejetju, ali npr. koncu računanja funkcije). Če takega prehoda ni, V to čudežno ugotovi in se takoj ustavi.

Nedeterministični TM ni uresničljiv (realističen) model računanja, ker lahko ugotovi, kakšno računanje in kakšen rezultat bi sledil (v prihodnosti), če bi izbral ta ali oni možni prehod. Tak TS je sposoben v danem trenutku izbirati med različnimi prihodnostmi! Kljub temu se izkaže za koristno orodje, ki je omogočilo marsikatero ugotovitev ali odkritje. To bomo videli v naslednjih poglavjih.

- Zdi se, da je V zaradi svojih čudežnih sposobnosti močnejši od osnovnega modela T . In vendar ni tako: T lahko izračuna vse, kar lahko izračuna V . **Dokaz** (ideja): pokažemo, da T lahko simulira vsako potezo stroja V . (Obratno je trivialno: T je poseben V (≤ 1 možen prehod).) \square

◆ Pomen in uporaba različic TS.

- ◆ Različice so koristne, ko želimo dokazati, da obstaja Turingov stroj T za dani problem P . Običajno je dokaz eksistence takega T enostavnejši, če iščemo kako njegovo različico V (ali celo kombinacijo različic). (Različice so bolj "okretne".)

Pogosto se celo izognemo (zapleteni) konstrukciji dejanskega TS V , ki reši P . Kako?

Storimo naslednje:

1. Sestavimo intuitivni algoritem A ("recept", končno zaporedje ukazov) za reševanje P .
2. Rečemo: "Po Tezi o izračunljivosti obstaja TS T , ki stori enako kot naš intuitivni A ."

Odslej se lahko sklicujemo na V (kot TS za reševanje P) in obravnavamo V kot obstoječ TS.

- ◆ Ker nobena od različic TS po svoji moči ne prekaša osnovnega TS, je to še dodatno dejstvo, ki gre v prid Tezi o izračunljivosti.
- ◆ **Opomba.** Izračuni na različicah TS so običajno hitrejši (merjeno v številu korakov) in prostorsko manj potratni (merjeno v številu obiskanih celic). Toda to bo pomembeno šele pri **Računski zahtevnosti** (kjer bomo obravnavali časovno in prostorsko zahtevnost reševanja problemov).

7.5 Univerzalni Turingov stroj

- ◆ Spoznali bomo, kako je Turing odkril ključno dejstvo o svojih strojih.
Izvedeli bomo,
 - ◆ da Turingove stroje lahko **kodiramo** z besedami nad neko abecedo;
 - ◆ kako Turingovi stroji berejo kode drugih TS in „računajo“ z njimi;
 - ◆ kako je Turing to uporabil in konstruiral t.i. **univerzalni Turingov stroj (UTS)**, poseben TS, ki zmore izračunati vse, kar je izračunljivo na kateremkoli drugem TS.
 - ◆ kako je to odkritje sprožilo iskanje fizične realizacije UTS, in v 40-ih letih 20. stoletja privelo do konstrukcije prvega **splošno-namenskega računalnika**.

◆ Kodiranje Turingovih strojev

- ◆ Naj bo $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ poljuben *osnovni* model TS. Stroj T želimo **zakodirati**, tj. predstaviti z besedo nad neko *kodirno* abecedo. Kako to storimo?
 - ◆ Naj bo *kodirna abeceda* kar množica $\{0,1\}$.
 - ◆ Zakodirali bomo samó funkcijo δ . To bomo storili tako, da se bodo iz kode funkcije δ dali enostavno izluščiti parametri Q, Σ, Γ, F , ki tudi določajo T . To storimo takole:

1. Če je $\delta(q_i, z_j) = (q_k, z_\ell, D_m)$ ukaz v funkciji δ ,
ga predstavimo z besedo

$$K = 0^i 1 0^j 1 0^k 1 0^\ell 1 0^m \quad \text{kjer so } D_1=L, D_2=R, D_3=S.$$

2. Tako predstavimo vsak ukaz funkcije δ .
3. Iz dobljenih predstavitev K_1, K_2, \dots, K_r sestavimo kodo $\langle \delta \rangle$ funkcije δ takole:

$$\langle \delta \rangle = 1 1 1 K_1 1 1 K_2 1 1 \dots 1 1 K_r 1 1 1$$

- ◆ Koda $\langle T \rangle$ stroja T je potem kar koda $\langle \delta \rangle$ njegovega programa, tj., $\langle T \rangle := \langle \delta \rangle$.

◆ Primer.

- ◆ Kakšna je koda $\langle T \rangle$ prvega od TS, ki računata vsoto $m+n$ (pogl.7.2, str. 197)?
- ◆ Komponente stroja T so:
 - ◆ $Q = \{q_1, q_2, q_3\}$ oz. $Q = \{\mathbf{0,00,000}\}$
 - ◆ $\Sigma = \{0,1\}$
 - ◆ $\Gamma = \{0,1, \sqcup\}$ oz. $\Gamma = \{\mathbf{0,00,000}\}$ (ker so: $0=z_1$, $1=z_2$, $\sqcup=z_3$)
 - ◆ $F = \{q_3\}$

Turingov program δ stroja T ima štiri ukaze:

1. $\delta(q_1, 1) = (q_2, \sqcup, R)$ oz. $K_1 = \mathbf{01001001000100}$
2. $\delta(q_2, 1) = (q_2, 1, R)$ oz. $K_2 = \mathbf{00100100100100}$
3. $\delta(q_2, 0) = (q_3, 1, S)$ oz. $K_3 = \mathbf{001010001001000}$
4. $\delta(q_1, 0) = (q_3, \sqcup, S)$ oz. $K_4 = \mathbf{010100010001000}$

Koda $\langle \delta \rangle$ programa δ je

$\langle \delta \rangle = \mathbf{111010010010001001110010010010010011100101000100100011010100010001000111}$

- ◆ **Oštevilčenje Turingovih strojev**
- ◆ Kodo $\langle T \rangle$ Turingovega stroja lahko interpretiramo kot *dvojiški zapis* nekega *naravnega števila*. Temu številu pravimo **indeks Turingovega stroja T** .
 - ◆ **Primer.** Index TS T za računanje vsote $m+n$ (glej prejšnji primer) je naravno število 1331015301402912694716154818999989357232619946567. Indeksi strojev so *ogromna* števila. Vendar nas to ne bo motilo, ker indeksi strojev ne bodo argumenti aritmetičnih operacij.
- ◆ **Opazimo:** nekatera naravna števila *niso* indeksi strojev.
(Zakaj? Njihov dvojiški zapis nima zgradbe, ki jo imajo kode strojev.)
 - ◆ Da to nepraktično dejstvo (“lepotno napako”) odpravimo, vpeljemo naslednji **Dogovor:** Vsako naravno število, čigar dvojiški zapis ni oblike, ki jo imajo kode Turingovih strojev, bo indeks posebnega, umetno uvedenega TS, imenovanega **prazni TS**. Program δ tega TS je povsod nedefiniran; tj. pri vsaki vhodni besedi se ta TS takoj ustavi (v 0 korakih).
- ◆ Zdaj lahko rečemo: **Vsako naravno število je indeks natanko enega Turingovega stroja.**

- Za poljubni dani $n \in \mathbb{N}$ lahko iz n izluščimo komponente Q, Σ, Γ, F , ki skupaj z δ natančno določijo konkretni TS T , ki ima indeks n . Kako?
 - (1) preverimo, ali ima dvojiški zapis števila n zgradbo $111K_111K_211 \dots 11K_r111$. Če jo ima, potem je n indeks nekega TS T . (2) Nato pregledamo nize K_1, K_2, \dots, K_r , da zberemo vse podatke, ki določajo komponente Q, Σ, Γ, F tega TS T .
- Dobljeni Q, Σ, Γ, F skupaj z δ natančno doličajo *konkretni osnovni* Turingov stroj $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$, ki ima indeks n . Ta TS običajno označimo tudi s T_n .
- Zdaj pa naj n teče po zaporednih vrednostih $0, 1, 2, \dots$. Z zgornjim postopkom lahko konstruiramo *zaporedje* Turingovih strojev

$$T_0, T_1, T_2, \dots$$

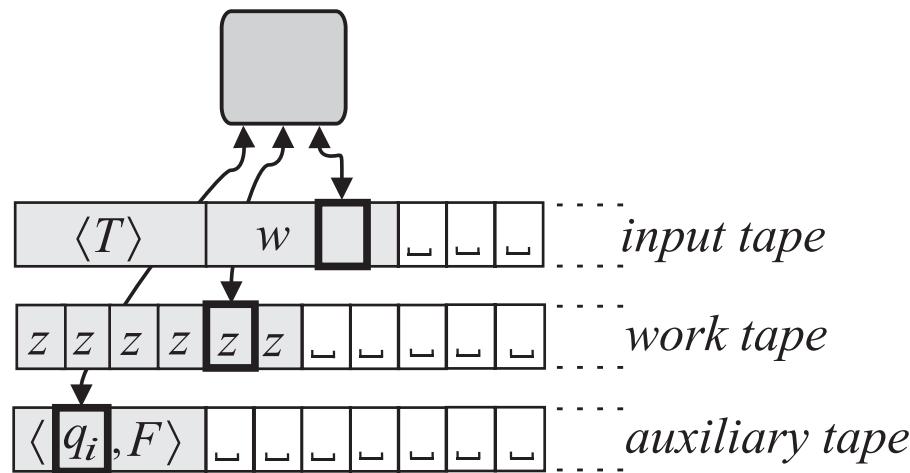
S tem smo **oštreviličili** (enumerirali) osnovne Turingove stroje. *Vsakemu osnovnemu TS pripada natančno določen indeks $n \in \mathbb{N}$ in vsakemu $n \in \mathbb{N}$ natančno odločen osnovni TS.)*

- ◆ **Obstoj univerzalnega Turingovega stroja**
- ◆ Leta 1936 je Turing --- potem, ko je našel ostevilčenje osnovnih TS --- prišel do *prelomnega odkritja* v zvezi s TS. Njegovo odkritje lahko zapišemo takole.
- ◆ **Trditev.** Obstaja Turingov stroj U , ki lahko izračuna vse, kar se da izračunati s katerimkoli (drugim) Turingovim strojem.
- ◆ **Ideja dokaza.** Turing si je zamislil in matematično *konstruiral* poseben TS, ki je sposoben *simulirati* katerikoli drugi TS. Mi si bomo dokazovanje, da tak poseben stroj obstaja, olajšali s *Tezo o izračunljivosti* (v naslednjih dveh korakih):
 - a) *Zamislili* si bomo nek stroj Z in *zapisali intuitivni algoritem*, ki naj ga izvaja stroj Z . (Stroj Z bo kar ena od različic osnovnega Turingovega stroja, toda z *intuitivno* opisanim programom.)
 - b) Nato se bomo sklicali na *Tezo o ozračunljivosti*, rekoč: “*Teza o ozračunljivosti zagotavlja, da obstaja Turingov stroj U , ki počne isto kot opisani algoritem na zamišljenem stroju Z .*”

Potem, če želimo, se lahko posvetimo dejanski konstrukciji TS U .

◆ Dokaz.

(a) Stroj Z si zamislimo kot večtračni TS (ki nima Turingovega programa).



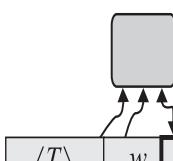
Nadzorna enota nima pravega Turingovega programa, pač pa *intuitivno* zasnovan algoritmom, kot je opisano na naslednji strani.

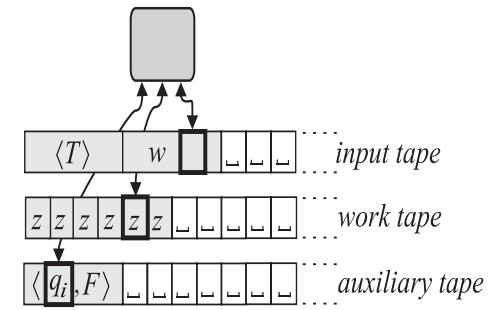
Vhodni trak vsebuje vhodno besedo, ki ima dva dela: kodo $\langle T \rangle$ poljubnega TS $T = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$, in poljubno besedo w (vh. besedo za T).

Delovni trak je spočetka prazen. Z ga bo uporabljal *natanko tako*, kot bi T uporabljal svoj trak pri dani vhodni besedi w .

Pomožni trak je spočetka prazen. Z ga bo uporabljal za zapisovanje *trenutnega stanja*, v katerem bi bil T v danem trenutku, in za *preverjanje*, ali je to stanje *končno stanje* stroja T .

Nadzorna enota stroja Z naj izvaja naslednji *intuitivo zasnovan algoritem*:

- Preveri, ali je vhodna beseda oblike $\langle T, w \rangle$, kjer je $\langle T \rangle$ koda nekega osnovnega TS. Če ni, se ustavi..
 - Iz $\langle T \rangle$ izlušči F in zapiši $\langle q_1, F \rangle$ na pomožni trak.
 - Kopiraj w na delovni trak in postavi okno na začetek tega traku.
 - // Naj bo na pomožnem traku $\langle q_i, F \rangle$, v oknu na delovnem traku pa z_r .
Če je $q_i \in F$, se ustavi. // T bi se ustavil v končnem stanju q_i .
 - Na vhodnem traku išči v kodi $\langle T \rangle$ zapis ukaza $\delta(q_i, z_r) = \dots$
 - Če ne najdeš, se ustavi. // T bi se ustavil v nekončnem stanju q_i .
 - // Ukaz $\delta(q_i, z_r) = \dots$ je bil najden in prebran; denimo, da je to ukaz $\delta(q_i, z_r) = (q_j, z_w, D)$.
Na delovnem traku izpiši simbol z_w in premakni okno v smeri D .
 - Na pomožnem traku nadomesti q_i v $\langle q_i, F \rangle$ s q_j .
 - Skoči na korak 4.

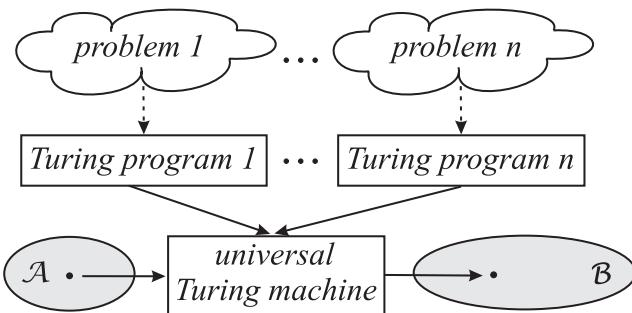


- (b) Zgornji algoritem lahko izvede tudi človek. Po Tezi o izračunljivosti zato obstaja Turingov stroj $U = (Q_U, \Sigma_U, \Gamma_U, \delta_U, q_1, \sqcup, F_U)$, čigar program δ_U izvaja ravno naš intuitivni algoritem. Ta TS je iskani **univerzalni Turingov stroj (UTS)**!

- ◆ **Konstrukcija univerzalnega Turingovega stroja**
- ◆ UTS $U = (Q_U, \Sigma_U, \Gamma_U, \delta_U, q_1, \sqcup, F_U)$ je bil dejansko *konstruiran* (podrobno opisan).
- ◆ Bilo je pričakovano, da bo $\langle U \rangle$ zalo dolgo zaporedje simbolov 0 in 1.
 - ◆ Res, koda $\langle U \rangle$ stroja U , ki sta ga 1989 sestavila Penrose & Deutsch je obsegala $\approx 5,500$ bitov.
- ◆ Toda obstajajo različice UTS, ki so *ekvivalentne* U (zmorejo isto kot U).
 - ◆ Raziskovalci so se osredotočili na različice UTS z *neomejenimi trakovi*.
- ◆ Pojavilo se je vprašanje: Kakšen je *najmanjši* tak UTS?
 - ◆ **Vprašanje:** *Kako meriti velikost teh UTS?*
 - ◆ Shannon je za mero predlagal produkt $|Q_U| \cdot |\Gamma_U|$ (= **max** število ukazov v δ_U . **Dokaži**)
 - ◆ Boljša mera bi bila **dejansko** število ukazov v programu δ_U .

- ◆ Kmalu je postalo jasno, da imata $|Q_U|$ in $|\Gamma_U|$ nasprotni težnji (angl. trade-off), tj. zmanjšanje enega povzroči povečanje drugega.
- ◆ Raziskovalci so zato definirali "razrede" UTS in se osredotočili nanje.
 - ◆ Razred **UTS(s,t)** ($s,t \geq 2$) vsebuje vse UTS, ki imajo s stanj in t tračnih simbolov.
 - ◆ 1996 Rogožin konstruiral UTS v razredih
 - ◆ UTM(2,18), ... UTS z 2 stanjema in 18 tračnimi simboli
 - ◆ UTM(3,10), ...
 - ◆ **UTM(4,6)**, ... UTS s 4 stanji in 6 tračnimi simboli
 - ◆ UTM(5,5), ...
 - ◆ UTM(7,4), ...
 - ◆ UTM(10,3), ...
 - ◆ UTM(24,2). ... UTS s 24 stanji in 2 tračnima simboloma
- ◆ Med temi ima univerzalni Turingov stroj v UTS(4,6) najmanj ukazov: **22**.
- ◆ Iskanje še manjših UTS danes še vedno poteka.

- **Pomen univerzalnega Turingovega stroja**
- Turingovo odkritje univerzalnega Turingovega stroja je bil *teoretični dokaz*, da je **splošno-namenski računski stroj vsaj načeloma možen.**



Denimo, da ima i -ti uporabnik ($i = 1, 2, \dots$) rač. problem $\Pi_i(x)$, kjer je x formalni parameter, ki predstavlja vh. podatke problema. Uporabnik želi rešitev primerka $\Pi_i(a)$, tj. rešitev problema pri dejanskem parametru a . Opišimo reševanje brez in z UTS.

- Uporabnik realizira TS T_i s TP P_i za reševanje $\Pi_i(x)$ na T_i . Nato zažene T_i z vh. besedo a , da T_i računa $\Pi_i(a)$.
- Uporabnik sestavi le program P_i za $\Pi_i(x)$ na T_i (ne realizira T_i). Nato zažene (že obstoječi) UTS U z vh. besedo $\langle T_i, a \rangle$. U računa $\Pi_i(a)$ tako, da simulira T_i pri vh. besedi a .

- Turing pa je bil prepričan, da se dá tak računski stroj tudi *dejansko* sestaviti: **Možno je fizično realizirati računski stroj, ki zmore izračunati vse, kar je izračunljivo na kateremkoli drugem TS (oz. kateremkoli drugem fizičnem računskem stroju).**

Turing je torej *napovedal* napravo, ki ji danes pravimo **splošno-namenski računalnik**.

- ◆ **Posledice: splošno-namenski računalnik**
- ◆ Konstrukcija *splošno-namenskega računalnika* se je pričela okrog 1940. Raziskovalci so razvili prve take stroje, ki jih danes pravimo **računalniki**.
 - ◆ Primeri: ENIAC, EDVAC, IAS. Do srede 50-ih let se je pojavil kak *ducat* računalnikov.
- ◆ Razvoj teh računalnikov ni povsem verno sledil zgradbi univerzalnega TS. Zakaj? Razlogov za to je bilo več:
 - ◆ pomembno je vplivala želja po *čim večji časovni učinkovitosti* računalnika; in
 - ◆ *tehnološke danosti* tistega časa (*elektronika* se je šele začela razvijati).

- ◆ Glavne razlike, ki so nastale med UTS in konstruiranimi računalniki so bile (s stališča TS):
 - ◆ Celice so bile oštevilčene (česar pri TS ni bilo).
 - ◆ Program je bil zapisan na traku (namesto v nadzorni enoti).
 - ◆ Nadzorna enota je dobila:
 - ◆ neposredni dostop do katerekoli celice v konstantnem času (nepotrebnega okna ni bilo več).
 - ◆ drugačne naloge. V vsakem koraku je morala nadzorna enota (običajno) storiti naslednje:
 1. prebrati ukaz iz celice;
 2. prebrati operande ukaza iz celic;
 3. izvršiti operacijo nad operandi;
 4. izpisati rezultat v celico.
 - ◆ nove komponente:
 - ◆ programski števec (za določitev celice, od kjer se bo preberal ukaz),
 - ◆ registre (za operande, ...),
 - ◆ akumulator (za rezultat operacije).
- ◆ Spremembam se je prilagodila terminologija: *glavni pomnilnik* (\approx trak), *program* (\approx Turingov program), *processor* (\approx nadzorna enota), *spominska lokacija* (\approx celica), *naslov* (\approx "zaporedna številka celice"). Splošno zgradbo teh računalnikov so poimenovali *von Neumannova arhitektura*.

7.6 Prvi osnovni rezultati

- ◆ V prejšnjih poglavjih smo opisali *osnovne pojme* in *zamisli Teorije izračunljivosti*. Natančneje:
 - ◆ Formalno smo definirali pojme *algoritem*, *računanje*, *izračun* in *izračunljiva funkcija*;
 - ◆ Formalno smo definirali pojme *odločljiva*, *polodločljiva*, *neodločljiva* in *i.p. množica*.
- ◆ Zdaj lahko začnemo raziskovati lastnosti teh pojmov in odnose med njimi. Torej začnemo iskati in dokazovati prve, osnovne *izreke* o teh pojmih.
- ◆ Na naslednji strani je nekaj osnovnih izrekov o *odločljivih* in *polodločljivih množicah* --- take množice bodo igrale *ključno* vlogo v nadaljevanju.

◆ **Izreki.** Naj bodo S, A, B poljubne množice. Velja naslednje:

- a) S je odločljiva $\Rightarrow S$ je polodločljiva
- b) S je odločljiva $\Rightarrow \overline{S}$ je odločljiva
- c) S in \overline{S} sta polodločljivi $\Rightarrow S$ je odločljiva (t.i. Postov izrek)
- d) A in B sta polodločljivi $\Rightarrow A \cap B$ in $A \cup B$ sta polodločljivi
- e) A in B sta odločljivi $\Rightarrow A \cap B$ in $A \cup B$ sta odločljivi

Dokaz. Dokazi so lahki. **Poskusite** (uporabite definiciji (pol)odločljive množice). \square

◆ Na kratko opišimo še bistvo treh pomembnih, a zahtevnejših izrekov:

- ◆ *Lema o dopolnjevanju* (*Padding Lemma*)
- ◆ *Izrek o parametrizaciji* (*Parametrization (s-m-n) Theorem*)
- ◆ *Izrek o rekurziji* (*Recursion (Fixed-Point) Theorem*).

7.7 Slovar

Turing machine Turingov stroj **naive set theory** naivna teorija množic paradox paradoks, protislovje **intuitionism** intuicionizem **logicism** logicizem **formalism** formalizem **Hilbert's program** Hilbertov program **model of computation** računski model **μ -recursive function** μ -rekurzivna funkcija **general recursive functions** splošno rekurzivna funkcija **λ -calculus** λ -račun **Post machine** Postov stroj **Markov algorithms** algoritmi Markova, Markovski algoritmi **computability thesis** teza o izračunljivosti **tape** trak **cell** celica **tape alphabet** tračna abeceda **empty space** presledek **input word** vhodna beseda **input alphabet** vhodna abeceda **control unit** nadzorna enota **state (initial, final)** stanje (začetno, končno) **Turing program** Turingov program **transition function** funkcija prehodov **window** okno **instantaneous description** trenutni opis **directly changes** neposredno preide **changes** preide **elementary task** osnovna naloga **function computation** računanje (vrednosti) funkcij **set recognition** razpoznavanje množic **set generation** generiranje množic **kary proper function** k-mestna lastna funkcija **computable function** izračunljiva funkcija **partial computable function** parcialna izračunljiva funkcija **incomputable function** neizračunljiva funkcija **language accepted by** jezik, ki ga sprejme **decidable** odločljiv **semi-decidable** polodločljiv **undecidable** neodločljiv **to halt** ustaviti se **language generated by** jezik, ki ga generira **enumerable** prešteven **computably enumerable (c.e.)** izračunljivo prešteven (i.p., c.e.) **finite storage** končni pomnilnik **multiple-track** večsledni **two-way infinite** dvosmerni **multiple-tape** večtračni **multidimensional** večdimenzionalni **universal TM** univerzalni TS **coding** kodiranje **index** indeks **enumeration** oštevilčenje **general-purpose** splošno-namenski

8

Neodločljivost

Vsebina

- ◆ Računski problemi
- ◆ Reševanje računskih problemov
- ◆ Neizračunljiv problem – Problem ustavitve
- ◆ Osnovne vrste odločitvenih problemov
- ◆ Drugi neizračunljivi problemi

8.1 Računski problemi

- ◆ V prejšnjih poglavjih smo spoznali:
 - ◆ kako s TS *računamo vrednosti funkcij*,
 - ◆ kako s TS *razpoznavamo množice*,
 - ◆ kako s TS *generiramo množice*.
- ◆ Te **osnovne računske naloge** so tesno povezane s Turingovim strojem.
- ◆ **Toda:** človek se v vsakdanjem življenju sooča z realnimi, pogosto mnogo bolj zapletenimi računskimi nalogami (v primerjavi z zgornjimi tremi), ki lahko zahtevajo dosti bolj zapleteno reševanje. Vsem tem realnim računskim nalogam pravimo **računski problemi**.

- ◆ **Vrste računskih problemov**
- ◆ Definirajmo naslednje štiri vrste računskih problemov:
 - ◆ **Odločitveni problemi** (imenovani tudi **da/ne** problemi). Rešitev odločitvenega problema je odgovor DA ali NE (torej en *bit*).
 - ◆ **Primeri:** Ali dana množica števil S vsebuje kako praštevilo?
Ali dani graf $G(V,E)$ vsebuje kak Hamiltonov cikel?
 - ◆ **Problemi iskanja.** Pri dani množici S in lastnosti P je rešitev takega problema element $x \in S$, ki ima lastnost P .
 - ◆ **Primeri:** V dani množici števil S poišči največje praštevilo.
V danem uteženem grafu $G(V,E,c)$ poišči najkrajši Hamiltonov cikel.

(nadaljevanje)

- ◆ **Problemi preštevanja.** Pri dani množici S in lastnosti P je rešitev takega problema število elementov množice S , ki imajo lastnost P .
 - ◆ **Primeri:** Koliko praštevil je v dani množici števil S ?
Koliko Hamiltonovih ciklov je v danem grafu $G(V,E)$?
- ◆ **Problemi generiranja.** Pri dani množici S in lastnosti P je rešitev takega problema zaporedje (seznam) elementov množice S , ki imajo lastnost P .
 - ◆ **Primeri:** Izpiši vsa praštevila, ki so v dani množici števil S .
Izpiši vse Hamiltonove cikle, ki so v danem grafu $G(V,E)$.

- ◆ **Katero vrsto problemov bomo obravnavali?**
- ◆ **Odgovor:** Osredotočili se bomo na **odločitvene probleme**.

Zakaj? Odločitveni problemi sprašujejo po najenostavnem možnem rezultatu, tj. odgovoru DA/NE, ki ga predstavlja en sam bit. Zato smo *pragmatični* in upamo, da bo raziskovanje odločitvenih problemov obrodilo spoznanja, ki bodo koristna tudi pri raziskavah ostalih vrst problemov.

- ◆ **Pozor:** To ne pomeni, da so druge vrste računskih problemov manj pomembne od odločitevenih problemov. Njihovo obravnavo želimo le odložiti, dokler ne bomo podrobneje spoznali odločitvenih problemov.

8.2 Reševanje računskih problemov

- ◆ Zastavi se vprašanje:

Ali lahko uporabimo pridobljeno znanje
o treh osnovnih računskih nalogah, ki jih izvaja Turingov stroj,
pri reševanju *računskih problemov*?

- ◆ Da; to znanje bomo uporabili pri raziskovanju *odločitvenih problemov*.
V nadaljevanju bomo:
 1. vzpostavili povezavo med množicami (formalnimi jeziki) in *odločitvenimi problemi*;
 2. uporabili znanje o množicah (formalnih jezikih) na *odločitvenih problemih*.

◆ Jezik odločitvenega problema

- ◆ Obstaja tesna zveza med odločitvenimi problemi in množicami, ki omogoča, da prevedemo vprašanja o odločitvenih problemih na vprašanja o množicah. Zvezo bomo konstruirali v štirih korakih.

①

Naj bo D poljuben *odločitveni problem*.

- ② V praksi smo običajno soočeni s konkretnim **primerkom** d problema D . Primerek d nastane iz D , ko zamenjamo **spremenljivke** (formalne parametre) v definiciji D z **dejanskimi podatki** (dejanskimi parametri). Problem D si zato lahko predstavljamo kot *množico vseh njegovih primerkov*. Rekli bomo, da je primerek $d \in D$ **pozitiven** oz. **negativen**, če je odgovor na d DA oz. NE. Torej:

Naj bo d *primerek problema D* .

Primer. Naj bo $D_{\text{Pra}} = \text{"Ali } n \text{ praštevilo?"}$ odločitveni problem. Če zamenjamo *spremenljivko n z dejansko vrednostjo*, npr. 4, dobimo *primerek* $d_1 = \text{"Ali je 4 praštevilo?"}$ problema D_{Pra} . Ta primerek je negativen, ker je njegova rešitev odgovor NE. Nasprotno, od 2009 vemo, da je rešitev primerka $d_2 = \text{"Ali je } 2^{43112609}-1 \text{ praštevilo?"}$ odgovor DA; zato vemo, da je d_2 pozitiven primerek problema D_{Pra} .



- ③ Kako naj TS *izračuna* odgovor, ki pripada primerku d odločitvenega problema D ?

Opis primerka d v naravnem jeziku vsebuje razne dejanske vrednosti (konkretna števila, matrike, grafe, ...). Za izračun odgovora s *strojem*—bodisi abstraktnim, kot je TS, bodisi sodobnim računalnikom—moramo *predstaviti* te vrednosti v obliki, ki bo razumljiva temu stroju. Kako?

Ker računski stroj uporablja neko *vhodno* abecedo Σ (npr. $\Sigma = \{0, 1\}$), moramo izbrati neko *injektivno* funkcijo **koda** : $D \rightarrow \Sigma^*$, ki bo preslikala (kodirala) poljuben primerek $d \in D$ v besedo iz Σ^* (različna primerka bo preslikala v različni besedi). Taki funkciji rečemo **kodirna funkcija**. Množica $\text{koda}(D)$ vsebuje kode *vseh* primerkov problema D , zato je jezik nad abecedo Σ ($\text{koda}(D) \subseteq \Sigma^*$). Običajno bomo namesto $\text{koda}(d)$ pisali $\langle d \rangle$.

Primer. Za kodiranje primerkov problema $D_{\text{Pra}} = \text{“Ali je } n \text{ praštevilo?”}$ lahko izberemo funkcijo $\text{koda} : D_{\text{Pra}} \rightarrow \{0, 1\}^*$, ki $n \in \mathbb{N}$ preslika v njegov dvojiški zapis. Npr., $\langle \text{“Ali je 4 praštevilo?”} \rangle = 100$ in $\langle \text{“Ali je 5 praštevilo?”} \rangle = 101$. \square

Torej:

Naj bo koda : $D \rightarrow \Sigma^*$ kodirna funkcija.

- ④ Zdaj bi lahko začeli iskati TS, ki bi izračunal odgovor na dani primerek $d \in D$. Vendar tega ne bomo storili; nadaljevali bomo drugače. Kako?

Zberimo kode vseh pozitivnih primerkov problema D v množici $L(D)$.

$L(D)$ je podmnožica množice Σ^* ($L(D) \subseteq \Sigma^*$), torej jezik nad abecedo Σ .

Jezik $L(D)$ je pridružen odločitvenemu problemu D . Tu je uradna definicija:

Definicija. Jezik odločitvenega problema D je množica $L(D)$, definirana z

$$L(D) = \{\langle d \rangle \in \Sigma^* \mid d \text{ je pozitiven primerek odločitvenega problema } D\}.$$

Primer. Jezik odločitvenega problema $D_{\text{Pra}} = \text{"Ali je } n \text{ praštevilo?"}$ je jezik

$$L(D_{\text{Pra}}) = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 110101, \dots\}.$$

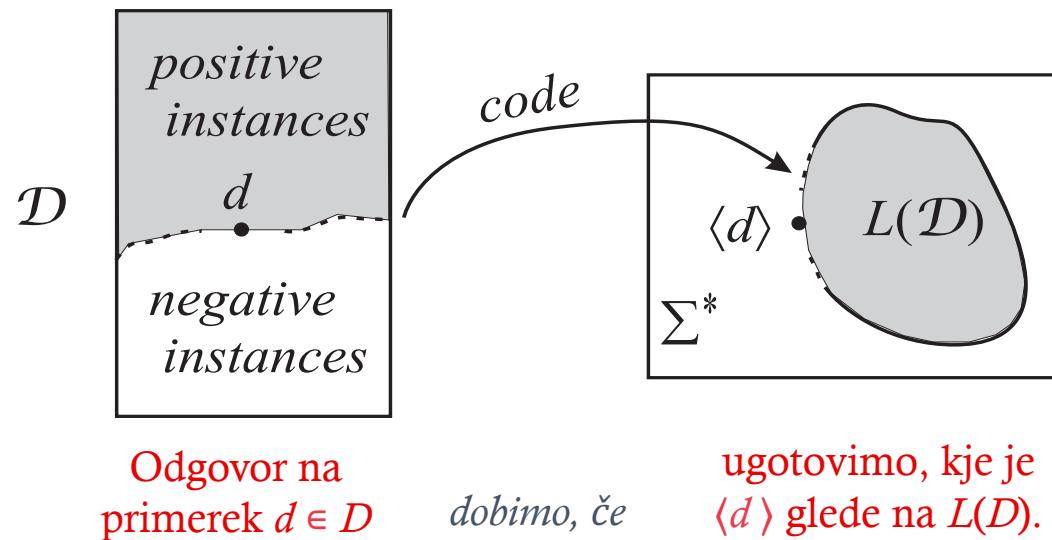
(Tu smo uporabili kodirno funkcijo iz prejšnjega primera.) \square

- ◆ Zdaj pa opazimo, da velja naslednja ekvivalenca:

$$d \in D \text{ je pozitiven primerek} \Leftrightarrow \langle d \rangle \in L(D) \quad (*)$$

To je iskana povezava med odločitvenimi problemi in množicami (formalnimi jeziki).

- **Kaj smo s tem dosegli?** Ekvivalenca \ast pove, da lahko *računanje odgovora na primerek $d \in D$* nadomestimo z ugotavljanjem, ali velja ali ne $\langle d \rangle \in L(D)$. Torej: **Reševanje odločitvenega problema D prevedemo na razpoznavanje množice $L(D)$ v Σ^* .**



- ◆ Povezava $*$ je pomembna, ker omogoča, da pri obravnavi *odločitvenih problemov* uporabimo dognanja, ki smo jih razvili za razpoznavanje množic.
- ◆ **Vprašanje:** Kaj nam *razpoznavnost* jezika $L(D)$ pove o *izračunljivosti problema D* ?
 - ◆ $L(D)$ je **odločljiv** \Rightarrow Obstaja algoritem, ki na poljuben $d \in D$ odgovori DA/NE.
Dokaz. Naj bo $L(D)$ odločljiv. Tedaj \exists TS, ki za poljuben $\langle d \rangle \in \Sigma^*$ odloči, ali je/ni $\langle d \rangle \in L(D)$. Nato $*$. \square
 - ◆ $L(D)$ je **polodločljiv** \Rightarrow Obstaja algoritem, ki
 - ◆ na *vsak pozitiven* $d \in D$ odgovori DA;
 - ◆ na *negativen* $d \in D$ odgovori NE ali pa sploh *ne odgovori*.**Dokaz.** Naj bo $L(D)$ polodločljiv. Potem \exists TS, ki sprejme vsak $\langle d \rangle$, kjer $\langle d \rangle \in L(D)$; če $\langle d \rangle \notin L(D)$, TS zavrne $\langle d \rangle$ ali pa se ne ustavi. Potem uporabi $*$. \square
 - ◆ $L(D)$ je **neodločljiv** \Rightarrow Ni algoritma, ki bi na poljuben $d \in D$ odgovoril DA/NE.
Dokaz. Naj bo $L(D)$ neodločljiv. Tedaj $\neg\exists$ TS, ki za poljuben $\langle d \rangle \in \Sigma^*$ odloči, ali je/ni $\langle d \rangle \in L(D)$. Nato $*$. \square

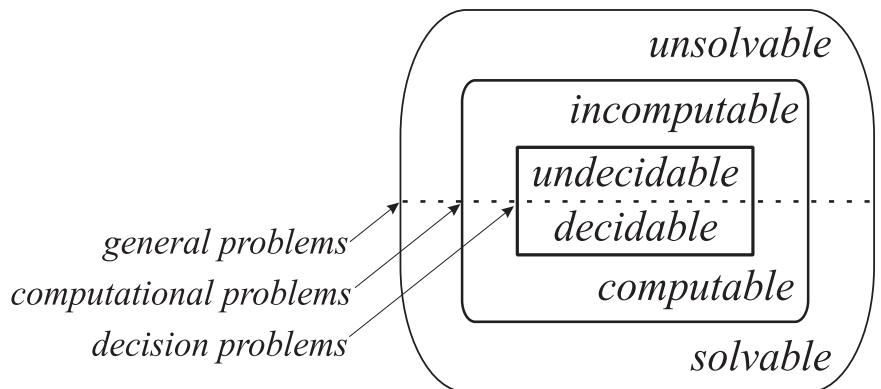
- Zdaj lahko razširimo izraze, ki se nanašajo na množice, še na *odločitvene probleme*.

Definicija. Naj bo D odločitveni problem. Tedaj:

- problem D je **odločljiv** (ali *izračunljiv*) če je jezik $L(D)$ odločljiv;
- problem D je **polodločljiv** če je jezik $L(D)$ polodločljiv;
- problem D je **neodločljiv** (ali *neizračunljiv*) če je jezik $L(D)$ neodločljiv.

- Terminologija.

Namesto *odločljiv/neodločljiv* problem lahko rečemo tudi *izračunljiv/neizračunljiv* problem. Vendar je slednje poimenovanje bolj splošno: nanaša se lahko *na vse vrste računskih problemov*, ne le na odločitvene. Izraza *rešljiv/nerešljiv* sta še splošnejša: lahko se nanašata na *računske* in tudi *ne-računske* probleme (npr. psihološke, službene, družbene, politične, ipd.).



8.3 Neizračunljiv problem – Problem ustavitve

- ◆ Zdaj vemo, **kaj so odločljivi, polodločljivi in neodločljivi DA/NE problemi.**
- ◆ Ne vemo pa, **ali sploh obstajajo polodločljivi problemi** (ki niso odločljivi) in ali obstajajo **neodločljivi problem** (ki niso polodločljivi).
- ◆ **Kako najti tak problem (če sploh obstaja)?** To je uspelo 1936 Turingu. Turing se je zavedal, da težavo pri računanju predstavlja tudi njegov stroj, ker se ta *včasih ne ustavi*. “*Koristilo bi*”-- je razmišljal -- “*če bi za poljuben TS T in poljubno vhodno besedo w lahko preverili, ali se T nad w ustavi ali ne ustavi.*”
 - ◆ Če bi bilo to možno, potem bi za dani par $\langle T, w \rangle$ najprej preverili, ali se T nad w ustavi. Če se ustavi, bi brez skrbi zagnali T nad w . Če pa se ne ustavi, bi poskušali dopolniti T (npr. Turingov program) tako, da bi se novi T ustavil tudi nad w .

- ◆ **Problem ustavitev**

- ◆ Turing je definiral naslednji odločitveni problem, t.i. *Problem ustavitev*.

Definicija. Problem ustavitev D_{Halt} je odločitveni problem

D_{Halt} = “Ali se TS T pri vhodni besedi $w \in \Sigma^*$ ustavi?”

- ◆ Potem je dokazal naslednji izrek.

Izrek. Problem ustavitev D_{Halt} je neodločljiv.

Opomba. Posledica je, da *ni algoritma*, ki bi bil sposoben za *poljuben* par T, w odgovoriti DA/NE na vprašanje “Ali se T pri vhodni besedi w ustavi?”

Drugače: *Vsak algoritem za reševanje Problema ustavitev, ki bi ga razvili danes ali v prihodnosti, bo zagotovo odpovedal* (ne vrnil odgovora) *pri vsaj enim paru* T, w .

- ◆ **Dokaz.**
- ◆ Predem se lotimo dokaza, definirajmo dve množici, ki bosta imeli pri tem pomembno vlogo. To sta *univerzalni* in *diagonalni jezik*.

Definicija. **Univerzalni jezik**, označen s K_0 , je jezik *Problema ustavitve*, torej

$$K_0 = L(D_{Halt}) = \{ \langle T, w \rangle \mid T \text{ se pri vhodu } w \text{ ustavi} \}.$$

Drugi jezik nastane iz jezika K_0 , ko zahtevamo, da je $w := \langle T \rangle$.

Definicija. **Diagonalni jezik**, označen s K , je definiran s

$$K = \{ \langle T, \langle T \rangle \rangle \mid T \text{ se pri vhodu } \langle T \rangle \text{ ustavi} \}.$$

- ◆ **Opomba:**
 - ◆ K je jezik odločitvenega problema D_H = “Ali se TS T pri vhodni besedi $\langle T \rangle$ ustavi?”
 - ◆ D_H je *podproblem* problema D_{Halt} (ker je dobljen iz D_{Halt} z omejitvijo w na $w := \langle T \rangle$).

- ◆ Zdaj lahko začnemo z dokazom. Načrt dokaza je sledeč:
 - ◆ dokazali bomo, da je K neodločljiva množica; (glej lemo spodaj)
 - ◆ potem bo sledilo, da je tudi K_0 neodločljiva množica; (ker $K \subseteq K_0$)
 - ◆ to pa bo pomenilo, da je D_{Halt} neodločljiv problem. (po def. neodločljivega problema)
- ◆ Spodnja lema je poučna; uporablja premeteno definiran TS S nad lastno kodo $\langle S \rangle$.

Lema. K je neodločljiva množica.

Dokaz leme. (dokaz s protislovjem)

★ Predpostavimo, da bi bila množica K odločljiva.

- ◆ Potem bi obstajal TS D_K , ki bi za poljuben T odgovoril na vprašanje $\langle T, T \rangle \in? K$ z

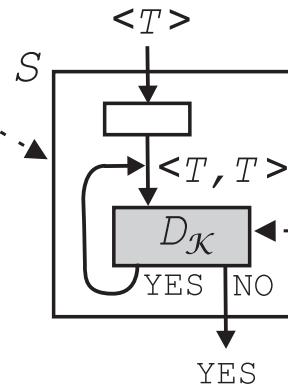
$$D_K(\langle T, T \rangle) = \begin{cases} \text{DA}, & \text{če } T \text{ se nad } \langle T \rangle \text{ ustavi;} \\ \text{NE}, & \text{če } T \text{ se nad } \langle T \rangle \text{ ne ustavi.} \end{cases}$$

- ◆ Zdaj pa konstruirajmo nov TS S .

Naš namen je sestaviti S tako, da bo -- ko bo za vhod dobil svojo lastno kodo $\langle S \rangle$ -- razkril nesposobnost TS D_K , da pravilno napove, ali se S nad $\langle S \rangle$ ustavi.

Zamislimo si TS S takole:

The shrewd Turing machine S . . . uncovers the incapability of D_K to answer whether S halts on its own code $\langle S \rangle$.



The supposed machine D_K answers, for arbitrary T , whether T halts on its own code $\langle T \rangle$.

- ◆ S deluje takole. Vhodna beseda stroja S je koda $\langle T \rangle$ poljubnega TS T . S podvoji $\langle T \rangle$ v besedo $\langle T, T \rangle$, jo preda kot vhodno besedo TS D_K in ga zažene. Po predpostavki ★ se D_K zagotovo ustavi in odgovori DA/NE na vprašanje $\langle T, T \rangle \in? K$. Če odgovori DA, mu S ponovi isto vprašanje. Če pa D_K odgovori NE, S vrne svoj odgovor DA in konča.
- ◆ *Toda S je zvito sestavljen:* če dobi na vhod svojo lastno kodo $\langle S \rangle$, razgali nesposobnost TS D_K , da v tem primeru izračuna pravilen odgovor. Poglejmo zakaj.
Ko S prejme vhodno besedo $\langle S \rangle$, jo (kot običajno) podvoji in preda stroju D_K , da ta v končnem času odgovori z DA/NE na vprašanje $\langle S, S \rangle \in? K$. (Zdaj se za začenejo D_K težave.)

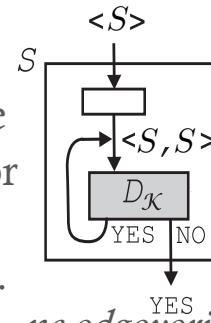
Poglejmo posledice odgovora na vprašanje $\langle S, S \rangle \in? K$:

- a) Denimo, da je D_K odgovoril DA. Tedaj S ponovi vprašanje $\langle S, S \rangle \in? K$ stroju D_K , ki pa seveda ponovi svoj odgovor $D_K(\langle S, S \rangle) = \text{DA}$. Tako se S vrati v zanki in nikoli ustavi. Toda D_K hkrati vstrajno napoveduje prav nasprotno, tj. da se bo S pri vhodu $\langle S \rangle$ ustavil. Sklep: v primeru a) D_K ne odgovori pravilno.
- b) Denimo, da je D_K odgovoril NE. Tedaj S vrne odgovor DA in se ustavi. Toda D_K je napovedal, da se S pri $\langle S \rangle$ ne ustavi. Sklep: v primeru b) D_K ne odgovori pravilno.

Torej D_K ni zmožen pravilno odgovoriti na vprašanje $\langle S, S \rangle \in? K$. To nasprotuje predpostavki \star , da je K odločljiva množica (in D_K pripadajoči TS). Predpostavka \star ne velja. Posledica: K je neodločljiva množica.

Lema je dokazana.

- ◆ Ker je K neodločljiva, je neodločljiv tudi DA/NE problem D_H . Ker je D_H podproblem problema D_{Halt} , mora biti tudi ta neodločljiv D_{Halt} (v nasprotnem bi bil odločljiv tudi D_H . Zakaj? TS, ki bi reševal D_{Halt} , bi lahko uporabili za reševanje D_H .)



8.4 Osnovne vrste odločitvenih problemov

- ◆ Zdaj vemo, da poleg *odločljivih* DA/NE problemov obstajajo tudi *neodločljivi* DA/NE problemi (zaenkrat je tak le eden, Problem ustavitve).
- ◆ Kaj pa *polodločljivi* DA/NE problemi? Ali obstajajo? Odgovor je **da**. **Zakaj?**
(Odgovor. Odločljiv problem obstaja. Njegov jezik je odločljiva množica. Vsaka odločljiva množica je tudi polodločljiva (glej izreke v 7.2). Zato je vsak odločljiv problem tudi polodločljiv.)
- ◆ Ali obstajajo *polodločljivi* DA/NE problemi, ki so *neodločljivi*? Vprašajmo drugače: Ali obstaja DA/NE problem, pri katerem vsak algoritem lahko zagotovi odgovor samo pri pozitivnih primerkih problema? Odgovor je **da**. Poglejmo, zakaj.

- ◆ Obstajajo neodločljive množice, ki so polodločljive
- ◆ **Izrek.** K_0 je polodločljiva množica.

Dokaz. Poiskati moramo TS, ki sprejme K_0 . Zamisel je sledeča. Iskani TS naj za poljubneni dani par $\langle T, w \rangle$ simulira T nad w . Če se simulacija konča -- torej se T nad w ustavi -- naj TS odgovori DA in se ustavi. Če tak TS obstaja, bo odgovoril DA natanko takrat, ko je $\langle T, w \rangle \in K_0$. Tak TS pa že poznamo: *univerzalni Turingov stroj* U . Torej je K_0 polodločljiva množica. □

Opomba. Zato se K_0 imenuje *univerzalni jezik*.

Neposredna posledica zadnjih dveh izrekov je:

Korolar. K_0 je neodločljiva (a še vedno) polodločljiva množica.

- ◆ Na podoben način dokažemo, da je tudi K neodločljiva (a še vedno) polodločljiva množica.

- ◆ Obstajajo neodločljive množice, ki niso polodločljive
- ◆ Opazujmo množico $\overline{K_0}$ (komplement množice K_0).

Izrek. $\overline{K_0}$ ni polodločljiva množica.

Dokaz. Denimo, da bi bila $\overline{K_0}$ polodločljiva. (Dokazali smo že, da je K_0 polodločljiva.) Potem bi bili K_0 in $\overline{K_0}$ obe polodločljivi in zato (izrek v razdelku 7.6) obe odločljivi. To bi bilo v protislovju s korolarjem (prejšnja stran). Sklep: $\overline{K_0}$ ni polodločljiva množica. □

- ◆ Na enak način dokažemo, da tudi \overline{K} ni polodločljiva množica.
- ◆ **Primer.** DA/NE problema, katerih jezika sta $\overline{K_0}$ in \overline{K} , označimo z $D_{\overline{Halt}}$ in $D_{\overline{H}}$:
 - ◆ $D_{\overline{Halt}} =$ “Ali se TS T pri vhodni besedi $w \in \Sigma^*$ ne ustavi?”
 - ◆ $D_{\overline{H}} =$ “Ali se TS T pri vhodni besedi $\langle T \rangle$ ne ustavi?”

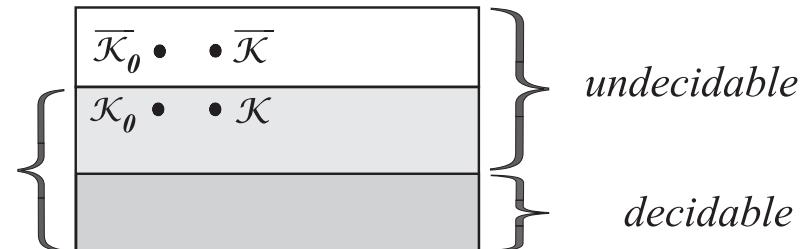
◆ Osnovne vrste odločitvenih problemov

- ◆ Dokazali smo, da obstajajo *neodločljive množice*, ki so *polodločljive* (npr. K_0 in K) in da obstajajo *neodločljive množice*, ki niso *polodločljive* (npr. $\overline{K_0}$ in \overline{K}). Ugotovili smo tudi, da so vse *odločljive množice* tudi *polodločljive*.
- ◆ Razred množic je torej razdeljen v tri (neprazne) podrazrede, kot kaže slika:

The class of all sets:

- ◆ neodločljive (niti polodločljive)
- ◆ neodločljive (toda polodločljive)
- ◆ odločljive

semi-decidable

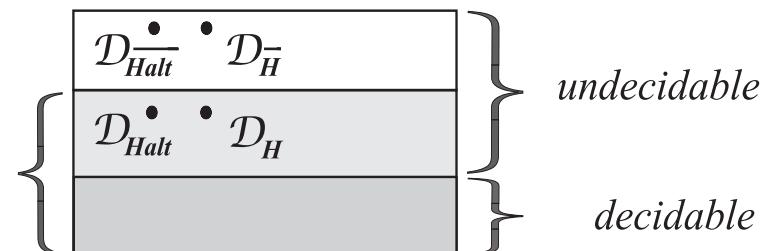


(nadaljevanje)

- ◆ Množice lahko obravnavamo, kot da so jeziki DA/NE *problemov*.
 - ◆ Npr., K_0 in K smo *defirali* kot jezika DA/NE problemov D_{Halt} in D_H . Kaj pa $\overline{K_0}$ in \overline{K} ? Ti množici sta jezika (sorodnih) DA/NE problemov $D_{\overline{\text{Halt}}}$ in $D_{\overline{H}}$:
 - ◆ $D_{\overline{\text{Halt}}} = \text{"Ali se TS } T \text{ pri vhodni besedi } w \in \Sigma^* \text{ ne ustavi?"}$
 - ◆ $D_{\overline{H}} = \text{"Ali se TS } T \text{ pri vhodni besedi } \langle T \rangle \text{ ne ustavi?"}$
- ◆ Razred DA/NE problemov je zato (tako kot razred množic) razdeljen v tri podrazrede:

The class of all decision problems:

- ◆ neodločljivi (niti polodločljivi)
- ◆ neodločljivi (toda polodločljivi)
- ◆ odločljivi



Torej razred *polodločljivih* problemov zajema vse *odločljive* in *nekatere* (ne vse) *neodločljive* probleme. Razred *neodločljivih* problemov zajema *nekatere* (ne vse) *polodločljive* in vse, ki niso niti *polodločljivi*.

(nadaljevanje)

- ◆ Vidimo, da je vsak odločitveni problem D ene od treh vrst:

- ◆ **D je odločljiv.**

Obstaja algoritmom, ki reši (odgovori z DA/NE na) *poljuben* primerek $d \in D$.
Takemu algoritmu včasih rečemo *odločevalnik* (angl. *decider*) za problem D .

- ◆ **D je polodločljiv (a neodločljiv).**

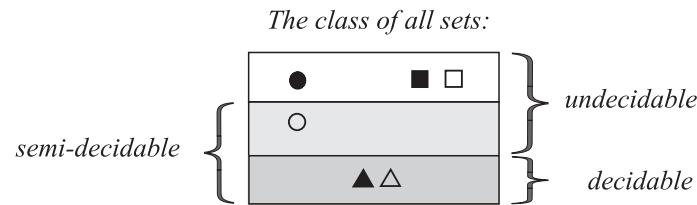
Ni algoritma, ki bi rešil *poljuben* primerek $d \in D$. Obstaja pa algoritmom, ki reši *poljuben pozitiven* $d \in D$, tj. *odgovori DA če in samo če* je d pozitiven primerek D .
(Pri *vsaj enim negativnem* primerku pa se nikoli *ne ustavi*).

Takemu algoritmu včasih rečemo *razpoznavalnik* (angl. *recognizer*) za D .

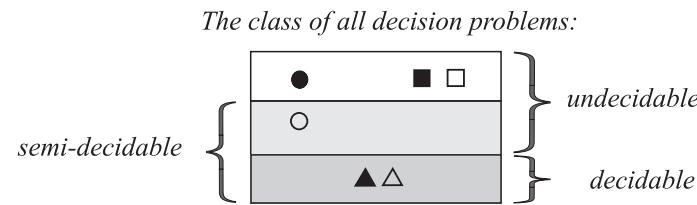
- ◆ **D ni niti polodločljiv.**

Ni algoritma, ki bi rešil *poljuben* $d \in D$; še več, ni niti algoritma, ki bi rešil *poljuben pozitiven* $d \in D$. Vsak algoritmom za D bo odpovedal (se nikoli ustavil) pri *vsaj enim pozitivnem* in *vsaj enim negativnem* primerku problema D .

- **Komplementarne množice in pripadajoči odločitveni problemi**
- Iz prejšnjih izrekov sledi, da so za odločljivost množice S in komplementa \bar{S} naslednji tri možnosti:
 - S in \bar{S} sta *obe odločljivi* ($\Delta\blacktriangle$, glej sliko spodaj);
 - S in \bar{S} sta *obe neodločljivi*, pri čemer je *ena polodločljiva*, druga pa *ne* ($\circ\bullet$);
 - S in \bar{S} sta *obe neodločljivi*, pri čemer nobena ni *polodločljiva* ($\blacksquare\blacksquare$).



- Enako velja za odločljivost pripadajočih *odločitvenih problemov*:



8.5 Drugi neizračunljivi problemi

- Ali poleg *Problema ustavitve* obstajajo še drugi neizračunljivi problemi?
Da, obstajajo!
- Po letu 1940 je bilo odkritih *veliko* neizračunljivih problemov. Prvi med njimi so se nanašali na lastnosti in delovanje *računski modelov*. Po 1944 pa so se začela vrstiti odkritja tudi bolj *realističnih* neizračunljivih problemov. Danes jih odkrivamo *na raznih področjih znanosti*.
- V tem razdelku bomo našteli nekaj izbranih neizračunljivih problemov, urejenih po področjih (iz računalništva in matematike). **Za nobenega od njih ne obstaja algoritem, ki bi ga zmogel rešiti v popolnosti (rešiti poljuben primerek).**

◆ Problem o algoritmih in programih

- ◆ TERMINATION OF ALGORITHMS (PROGRAMS) USTAVLJIVOST ALGORITMOV (PROGRAMOV)
Let A be an arbitrary algorithm and d be arbitrary input data. Questions:
 - ◆ D_{Term} = “Does A terminate on *every* input data?”
 - ◆ “Does A terminate on input data d ?”
- ◆ CORRECTNESS OF ALGORITHMS (PROGRAMS) PRAVILNOST ALGORITMOV (PROGRAMOV)
Let P be an arbitrary computational problem and A an arbitrary algorithm. Question:
 - ◆ D_{Corr} = “Does the algorithm $\text{code}(A)$ *correctly* solve the problem $\text{code}(P)$?”
- ◆ EXISTENCE OF SHORTER EQUIVALENT PROGRAMS KRAJŠI EKVIVALENTNI PROGRAM
Let $\text{code}(A)$ be a program describing an algorithm A . Question:
 - ◆ “Given a program $\text{code}(A)$, is there a *shorter equivalent* program?”

◆ Problem o jezikih in gramatikah

◆ AMBIGUITY OF CFG GRAMMARS DVOUMNOST KONTEKSTNO-NEODVISNIH GRAMATIK

Let G be a context-free grammar. Question:

- ◆ “Is there a word that can be generated by G in two different ways?”

◆ EQUIVALENCE OF CFG GRAMMARS EKVIVALENTNOST KONTEKSTNO-NEODVISNIH GRAMATIK

Let G_1 and G_2 be CFGs. Question:

- ◆ “Do G_1 and G_2 generate the same language?”

◆ OTHER PROPERTIES OF CFGs AND CFLs RAZNE LASTNOSTI KNG IN KNJ

Let G and G' be arbitrary CFGs, and let C and R be an arbitrary CFL and a regular language, respectively. As usual, Σ is the alphabet. Questions:

- ◆ “Is $L(G) = \Sigma^*$?” “Is $L(G)$ regular?” “Is $R \subseteq L(G)$?”
- ◆ “Is $L(G) = R$?” “Is $L(G) \subseteq L(G')$?” “Is $L(G) \cap L(G') = \emptyset$?”
- ◆ “Is $L(G) \cap L(G')$ CFL?” “Is C ambiguous CFL?” “Is there a palindrome in $L(G)$?”

- Problem o izračunljivih funkcijah
- INTRINSIC PROPERTIES OF COMPUTABLE FUNCTIONS RAZNE LASTNOSTI IZRAČUNLJIVIH FUNKCIJ
 - Let $\varphi: A \rightarrow B$ and $\psi: A \rightarrow B$ be arbitrary computable functions. Questions:
 - ◆ “Is $\text{dom}(\varphi)$ empty?”
 - ◆ “Is $\text{dom}(\varphi)$ finite?”
 - ◆ “Is $\text{dom}(\varphi)$ infinite?”
 - ◆ “Is $A - \text{dom}(\varphi)$ finite?”
 - ◆ “Is φ total?”
 - ◆ “Can φ be extended to a total computable function?”
 - ◆ “Is φ surjective?”
 - ◆ “Is φ defined at x ?”
 - ◆ “Is $\varphi(x) = y$ for at least one x ?”
 - ◆ “Is $\text{dom}(\varphi) = \text{dom}(\psi)$?”
 - ◆ “Is $\varphi = \psi$?”

- **Problemi iz teorije števil, algebre in matematične analize**
- SOLVABILITY OF DIOPHANTINE EQUATIONS **REŠLJIVOST DIOFANTSKEH ENAČB**
 - Let $p(x_1, \dots, x_n)$ be an arbitrary polynomial with unknowns x_1, \dots, x_n and rational coefficients.
 - Question:
 - ◆ “Does a Diophantine equation $p(x_1, \dots, x_n) = 0$ have a solution?”
- MORTAL MATRIX PROBLEM **NIČELNI PRODUKT MATRIK**
 - Let M be a finite set of $n \times n$ matrices with integer entries. Question:
 - ◆ “Can the matrices of M be multiplied in some order, possibly with repetition, so that the product is zero matrix O ?”
- EXISTENCE OF ZEROS OF FUNCTIONS **REALNE NIČLE FUNKCIJ**
 - Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be an arbitrary elementary function. Question:
 - ◆ “Is there a real solution to the equation $f(x) = 0$?”

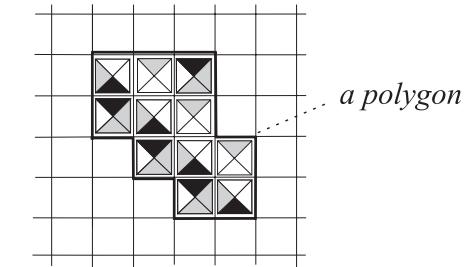
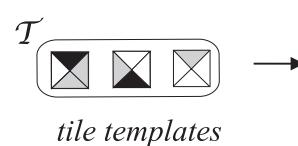
A function $f(x)$ is *elementary* if it can be constructed from a finite number of exponentials, logarithms, roots, real constants, and the variable x by using function composition and the four basic operations $+$, \square , \times , and \div .

◆ Problemni tlakovanja

◆ DOMINO TILING PROBLEM TLAKOVANJE POLIGONOV

Let T be a finite set of tile templates, each with an unlimited number of copies. Question:

- ◆ “Can every finite polygon be regularly T -tilied?”

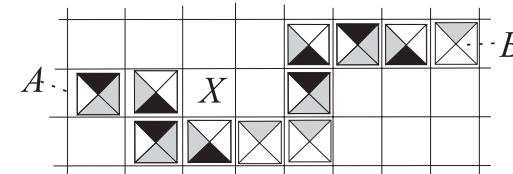
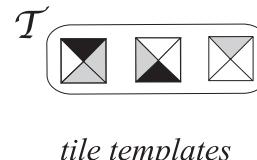


◆ DOMINO SNAKE PROBLEM TLAKOVANJE POTI

a regular T -tiling of the polygon

Let T be a finite set of tile templates and A, B, X arbitrary 1×1 squares in \mathbb{Z}^2 . Question:

- ◆ “Is there a path between A and B which avoids X and can be regularly T -tilied?”



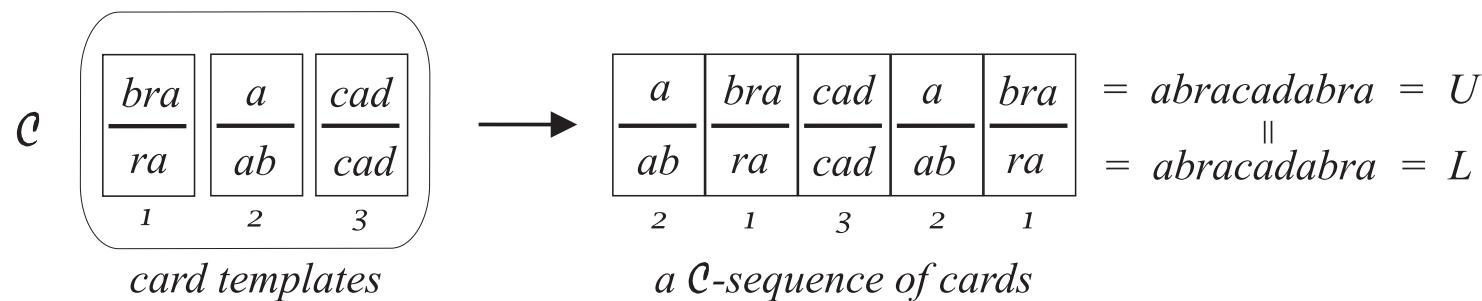
a regular T -tiling of a path $p(A,B,X)$

Postov korespondenčni problem

POST'S CORRESPONDENCE PROBLEM POSTOV KORESPONDENČNI PROBLEM

Let C be a finite set of card templates, each with an unlimited number of copies. Question:

- ◆ “Is there a finite C -sequence such that $U=L$?”



- ◆ **Problem garača (busy beaver problem)**
- ◆ Povedano enostavno: **garač je** najbolj *produktiven* TS med TS *iste vrste*.
- ◆ Na kakšno *vrsto* Turingovih strojev mislimo zgoraj?

Mislimo na Turingove stroje, ki ne porabljajo časa s pisanjem simbolov, različnih od 1, in tudi ne z zadrževanjem okna na obiskani celici. Pa jih razdelimo v razrede \mathcal{T}_n , $n = 1, 2, \dots$, tako da bo \mathcal{T}_n vseboval vse take TS, ki imajo *enako število stanj*.

Definicija. Naj bo \mathcal{T}_n ($n \geq 1$) razred vseh TS, ki imajo:

- ◆ neomejen trak v obe smeri;
- ◆ n nekončnih stanj (skupaj s q_1) in eno končno stanje, q_{n+1} ;
- ◆ vhodno abecedo $\Sigma = \{0, 1\}$ in tračno abecedo $\Gamma = \{0, 1, \square\}$;
- ◆ Program δ , ki na trak *vedno izpiše le* 1, in *okno vedno premakne* (L ali R).

Izrek. (Radó) Pri vsakem $n \geq 1$ je v \mathcal{T}_n *končno mnogo* Turingovih strojev.

- ◆ **Definicija.** TS $T \in \mathcal{T}_n$ je **ustavlјiv**, če se T pri praznem vhodu ε ustavi.
- Izrek.** (Radó) Za vsak $n \geq 1$ obstaja v \mathcal{T}_n ustavlјiv T .
Torej je v vsakem razredu \mathcal{T}_n vsaj eden in kvečjemu $|\mathcal{T}_n|$ ustavlјivih TS.
- ◆ Sledi, da v \mathcal{T}_n obstaja ustavlјiv TS T^* , ki zapusti na traku, ko se ustavi, največje število simbolov 1 med vsemi ustavlјivimi TS v \mathcal{T}_n . Temu T^* rečemo **n-garač**. Rekli bomo, da je TS T **garač**, če obstaja $n \geq 1$, da je T n -garač.
- ◆ BUSY BEAVER PROBLEM **PROBLEM GARAČA**
 - Naj bo $T \in \bigcup_{i \geq 1} \mathcal{T}_i$ poljuben TS. Vprašanje:
 - ◆ “Ali je T garač?” (tj. “Ali obstaja $n \geq 1$, pri katerem je $T = n$ -garač?”)
- ◆ **Definicija.** **Garačeva funkcija** $s(n)$ je definirana takole:
 $s(n)$ = ‘število simbolov 1, ki jih zapusti n -garač na traku, ko se ustavi’.
- Izrek.** Garačeva funkcija je *neizračunljiva*.

8.6 Dictionary

undecidability neodločljivost computational problems računski problemi decision problem odločitveni problem search problem problem iskanja, iskalni problem counting problem problem preštevanja generation problem problem generiranja language of a decision problem jezik odločitvenega problema instance primerek problema, naloga coding function kodirna funkcija code koda decidable, semi-decidable, undecidable decision problem odločljiv, polodločljiv, neodločljiv odločitveni problem computable/incomputable problem izračunljiv/neizračunljiv problem solvable/unsolvable problem rešljiv/nerešljiv problem halting problem problem ustavitev universal language univerzalni jezik diagonal language diagonalni jezik termination of ustavljenost correctness pravilnost ambiguity dvoumnost intrinsic property vsebovana (naravna, bistvena) lastnost solvability of Diophantine equations rešljivost Diofantskih enačb tiling problem problem tlakovanja Post's correspondence problem Postov korespondenčni problem busy beaver problem problem garača stopper ustavljen stroj

9

Hierarhija Chomskega

Vsebina

- ◆ ZAENKRAT IZPUSTIM

10

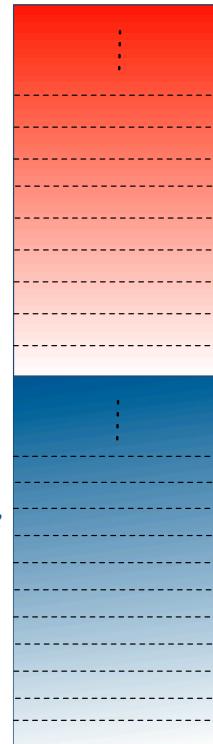
Računska zahtevnost

Vsebina

- ◆ Uvod
- ◆ Deterministični čas in prostor (razreda DTIME, DSPACE)
- ◆ Nondeterministični čas in prostor (razreda NTIME, NSPACE)
- ◆ Stiskanje traku, linearna pohitritev in zmanjšanje števila trakov v TS
- ◆ Relacije med razredi DTIME, DSPACE, NTIME, NSPACE
- ◆ Razredi P, NP, PSPACE, NPSPACE
- ◆ Problem $P =? NP$
- ◆ NP-polni in NP-težki problemi

10.1 Uvod

- ◆ Doslej nas je zanimalo, kaj se dá rešiti **vsaj načeloma** (ne glede na potreben čas/prostор) *incomputable* in česa se ne dá rešiti **niti načeloma** *problems* (tudi če bi bila na voljo neomejen čas in prostor).
- ◆ Odkrili smo, da obstajajo problemi, ki so **izračunljivi** (dejansko ali načeloma) a tudi problemi, ki so **neizračunljivi**. *computable problems*



*No algorithm can solve any of the incomputable problems *in general*. There are *infinitely many* different degrees of incomputability. There is *no* most incomputable problem!*

*Each computable problem has *an algorithm that solves it*. Intuition tells us: given more time/space, larger instances or more difficult problems can be solved.*

How much time/space can we afford?

10.2 Deterministični čas in prostor (DTIME, DSPACE)

- ❖ **Vprašanje:** Koliko časa oz. prostora potrebuje algoritem, da reši (odločljiv) odločitveni problem D ?

Zaradi zveze $*$ med odločitvenimi problemi in njihovimi jeziki lahko vprašanje zastavimo s pomočjo formalnih jezikov:

- ❖ **Vprašanje:** Koliko korakov oz. celic na traku potrebuje TS, da razpozna (odločljiv) jezik $L(D)$ problema D ?

V nadaljevanju bomo ti vprašanji formulirali natančneje.

Deterministična časovna zahtevnost & razredi DTIME

- ◆ **Definicija.** Naj bo $M = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ DTS s $k \geq 1$ neomejenimi trakovi. Pravimo, da ima DTS M (**deterministično**) **časovno zahtevnost** $T(n)$, če za vsako vhodno besedo $w \in \Sigma^*$ dolžine $|w| = n$ naredi *kvečjem* $T(n)$ korakov (potez), preden se ustavi.
 - ◆ V definiciji velja naslednja **predpostavka**: M prebere *celo* vhodno besedo w . Posledica: $T(|w|) \geq |w| + 1$, tj. $T(n) \geq n + 1$. Sledi: $T(n)$ vsaj linearna funkcija.
- ◆ Torej TS M (det.) časovne zahtevnosti $T(n)$ (**zagotovo**) odgovori na vprašanje
$$w \in? L(M)$$
v *kvečjemu* $T(|w|)$ korakih.

To je motivacija za naslednjo definicijo.

(nadaljevanje)

- ◆ **Definicija.** Jezik L ima **(deterministično) časovno zahtevnost $T(n)$** , če obstaja DTS M z (det.) časovno zahtevnostjo $T(n)$, za katerega je $L = L(M)$. Razred vseh jezikov z (det.) čas. zahtevnostjo $T(n)$ je

$$\text{DTIME}(T(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima (det.) časovno zahtevnost } T(n)\}$$

Intuitivno: DTIME($T(n)$) vsebuje natanko vse jezike L , pri katerih se dá odgovor na poljubno vprašanje $w \in? L$ deterministično izračunati v času $\leq T(|w|)$.

- ◆ Zgornji definiciji ustreza (zaradi *) podobna definicija, ki se nanaša na odločitvene probleme:

Definicija. Odločitveni problem D ima **(deterministično) časovno zahtevnost $T(n)$** , če ima njegov jezik $L(D)$ (det.) časovno zahtevnost $T(n)$.

Razred odločitvenih problemov z (det.) časovno zahtevnostjo $T(n)$ je

$$\text{DTIME}(T(n)) = \{D \mid D \text{ je odločitveni problem} \wedge D \text{ ima (det.) čas. zaht. } T(n)\}$$

Intuitivno: DTIME($T(n)$) vsebuje natanko vse DA/NE probleme D , pri katerih se dá poljuben primerek $d \in D$ deterministično rešiti v času $\leq T(|\langle d \rangle|)$.

Deterministična prostorska zahtevnost & razredi DSPACE

- ◆ **Definicija.** Naj bo $M = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ DTS z 1 vhodnim trakom in $k \geq 1$ delovnimi trakovi. Pravimo, da ima DTS M (deterministično) prostorsko zahtevnost $S(n)$, če za vsako vhodno besedo $w \in \Sigma^*$ dolžine $|w| = n$ porabi kvečjemu $S(n)$ celic na vsakem delovnem traku, preden se ustavi.
 - ◆ *Opomba:* celice na vhodnem traku se pri porabi prostora ne upoštevajo.
 - ◆ V definiciji velja naslednja **predpostavka:** M uporabi vsaj eno celico na vsakem delovnem traku (tisto, ki je pod oknom pred zagonom M). Sledi: $S(|w|) \geq 1$.
- ◆ Torej TS M (det.) prostorske zahtevnosti $S(n)$ (zagotovo) odgovori na vprašanje
$$w \in ? L(M)$$
na kvečjemu $S(|w|)$ celicah vsakega od delovnih trakov.

To je motivacija za naslednjo definicijo.

(nadaljevanje)

- ◆ **Definicija.** Jezik L ima **(deterministično) prostorsko zahtevnost** $S(n)$, če obstaja DTS M z (det.) prostorsko zahtevnostjo $S(n)$, za katerega je $L = L(M)$. Razred vseh jezikov z (det.) prostorsko zahtevnostjo $S(n)$ je

$$\text{DSPACE}(S(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima (det.) prostorsko zahtevnost } S(n)\}$$

Intuitivno: $\text{DSPACE}(S(n))$ vsebuje natanko vse jezike L , pri katerih se dá odgovor na poljubno vprašanje $w \in? L$ deterministično izračunati na $\leqslant S(|w|)$ celicah (na vsakem od delovnih trakov).

- ◆ Zgornjima definicijama ustrezata (zaradi *) podobni definiciji za DA/NE probleme:

Definicija. Odločitveni problem D ima **(deterministično) prostorsko zahtevnost** $S(n)$, če ima njegov jezik $L(D)$ (det.) prostorsko zahtevnost $S(n)$. Razred vseh odločitvenih problemov z (det.) prostorsko zahtevnostjo $S(n)$ je

$$\text{DSPACE}(S(n)) = \{D \mid D \text{ je odločitveni problem} \wedge D \text{ ima (det.) prost. zaht. } S(n)\}$$

Intuitivno: $\text{DSPACE}(S(n))$ vsebuje natanko vse DA/NE probleme D , pri katerih se dá poljuben primerek $d \in D$ deterministično rešiti na prostoru $\leqslant S(|\langle d \rangle|)$.

10.3 Nedeterministični čas in prostor (NTIME, NSPACE)

Recimo, da bi imeli na voljo tudi *nedeterministične* algoritme oz. TS (NTS).

- ◆ **Vprašanje:** Koliko časa oz. prostora bi potreboval *nedeterministični* algoritmom za reševanje (odločljivega) odločitvenega problema D ?

Zaradi zveze * lahko vprašanje zastavimo s pomočjo formalnih jezikov:

- ◆ **Vprašanje:** Koliko korakov oz. celic na traku bi rabil *nedeterministični* TS za razpoznavanje (odločljivega) jezika $L(D)$ problema D ?

V nadaljevanju bomo ti vprašanji formulirali natančneje.

Nedeterministična časovna zahtevnost & razredi NTIME

- ◆ **Definicija.** Naj bo $N = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ nedeterministični TS s $k \geq 1$ trakovi. Pravimo, da ima NTS N nedeterministično časovno zahtevnost $T(n)$, če za vsako vhodno besedo $w \in \Sigma^*$ dolžine $|w| = n$ obstaja izračun, v katerem N naredi *kvečjemu* $T(n)$ korakov (potez), preden se ustavi.
 - ◆ V definiciji spet velja naslednja **predpostavka**: N prebere celo vhodno besedo w . Posledica: $T(|w|) \geq |w| + 1$, tj. $T(n) \geq n + 1$. Sledi: $T(n)$ vsaj linearna funkcija.
- ◆ Torej NTS N nedet. časovne zahtevnosti $T(n)$ (zagotovo) odgovori na vprašanje
$$w \in? L(N)$$
v *kvečjemu* $T(|w|)$ korakih.

To je motivacija za naslednjo definicijo.

(nadaljevanje)

- ◆ **Definicija.** Jezik L ima **nedeterministično časovno zahtevnost** $T(n)$, če obstaja NTS N z nedet. časovno zahtevnostjo $T(n)$, tako da je $L = L(N)$. Razred vseh takih jezikov je

$$\text{NTIME}(T(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima nedet. časovno zahtevnost } T(n)\}$$

Intuitivno: NTIME($T(n)$) vsebuje natanko vse jezike L , pri katerih se dá odgovor na poljubno vprašanje $w \in? L$ nedeterministično izračunati v času $\leqslant T(|w|)$.

- ◆ Zgornji definiciji ustreza (zaradi *) podobna definicija, ki se nanaša na odločitvene probleme:
Definicija. Odločitveni problem D ima nedeterministično časovno zahtevnost $T(n)$, če ima njegov jezik $L(D)$ nedet. časovno zahtevnost $T(n)$.
Razred odločitvenih problemov z nedet. časovno zahtevnostjo $T(n)$ je

$$\text{NTIME}(T(n)) = \{D \mid D \text{ je odločitveni problem} \wedge D \text{ ima nedet. čas. zaht. } T(n)\}$$

Intuitivno: NTIME($T(n)$) vsebuje natanko vse odločitvene probleme D , pri katerih se dá poljuben primerek $d \in D$ nedeterministično rešiti v času $\leqslant T(|\langle d \rangle|)$.

Nedeterministična prostorska zahtevnost & razredi NSPACE

- ◆ **Definicija.** Naj bo $N = (Q, \Sigma, \Gamma, \delta, q_1, \sqcup, F)$ NTS z 1 *vhodnim trakom* in $k \geq 1$ *delovnimi trakovi*. Rečemo, da ima NTS N **nedeterministično prostorsko zahtevnost** $S(n)$, če za vsako vhodno besedo $w \in \Sigma^*$ dolžine $|w| = n$ obstaja izračun, v katerem N porabi *kvečjemu* $S(n)$ celic na vsakem delovnem traku, preden se ustavi.
 - ◆ *Opomba:* celice na *vhodnem traku* se pri porabi prostora *ne upoštevajo*.
 - ◆ V definiciji velja naslednja **predpostavka:** N uporabi *vsaj eno* celico na vsakem delovnem traku (tisto, ki je pod oknom pred zagonom N). Sledi: $S(|w|) \geq 1$.
- ◆ Torej NTS N nedet. prostorske zahtevnosti $S(n)$ (zagotovo) odgovori na vprašanje
$$w \in? L(N)$$
na *kvečjemu* $S(|w|)$ celicah vsakega od delovnih trakov.

Spet je to motivacija za naslednjo definicijo.

(nadaljevanje)

- ◆ **Definicija.** Jezik L ima **nedeterministično sprostorsko zahtevnost** $S(n)$, če obstaja NTS N z nedet. prostorsko zahtevnostjo $S(n)$, tako da je $L = L(N)$. Razred vseh takih jezikov je

$$\text{NSPACE}(S(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima nedet. prostorsko zahtevnost } S(n)\}$$

Intuitivno: NSPACE($S(n)$) vsebuje natanko vse jezike L , pri katerih se dá odgovor na poljubno vprašanje $w \in? L$ nedeterministično izračunati na $\leqslant S(|w|)$ celicah (na vsakem od delovnih trakov).

- ◆ Ustreznega definicija za odločitvene probleme je:

Definicija. Odločitveni problem D ima nedeterministično prostorsko zahtevnost $S(n)$, če ima njegov jezik $L(D)$ nedet. prostorsko zahtevnost $S(n)$.

Razred odločitvenih problemov z nedet. prostorsko zahtevnostjo $S(n)$ je

$$\text{NSPACE}(S(n)) = \{D \mid D \text{ je odločitveni problem.} \wedge D \text{ ima nedet. prost. zaht. } S(n)\}$$

Intuitivno: NSPACE ($S(n)$) vsebuje natanko vse odločitvene probleme D , pri katerih se dá poljuben primerek $d \in D$ nedeterministično rešiti na prostoru $\leqslant S(|\langle d \rangle|)$.

Povzetek razredov zahtevnosti

- ◆ Razredi z vidika formalnih jezikov in TS:

$\text{DTIME}(T(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima (det.) časovno zahtevnost } T(n)\}$

$\text{DSPACE}(S(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima (det.) prostorsko zahtevnost } S(n)\}$

$\text{NTIME}(T(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima nedeterministično časovno zahtevnost } T(n)\}$

$\text{NSPACE}(S(n)) = \{L \mid L \text{ je jezik} \wedge L \text{ ima nedeterministično prostorsko zahtevnost } S(n)\}$

- ◆ Isti razredi z vidika odločitvenih problemov in algoritmov:

$\text{DTIME}(T(n)) = \{D \mid D \text{ je odločitveni problem} \wedge L(D) \text{ ima (det.) časovno zahtevnost } T(n)\}$

$\text{DSPACE}(S(n)) = \{D \mid D \text{ je odločitveni problem} \wedge L(D) \text{ ima (det.) prostorsko zahtevnost } S(n)\}$

$\text{NTIME}(T(n)) = \{D \mid D \text{ je odločitveni problem} \wedge L(D) \text{ ima nedet. časovno zahtevnost } T(n)\}$

$\text{NSPACE}(S(n)) = \{D \mid D \text{ je odločitveni problem} \wedge L(D) \text{ ima nedet. prostorsko zahtevnost } S(n)\}$

- ◆ Intuitivno:

$\text{DTIME}(T(n)) = \{\text{vsi odločitveni problemi, rešljivi z determinističnim algoritmom v času } T(n)\}$

$\text{DSPACE}(S(n)) = \{\text{vsi odločitveni problemi, rešljivi z determinističnim algoritmom na prostoru } S(n)\}$

$\text{NTIME}(T(n)) = \{\text{vsi odločitveni problemi, rešljivi z nedeterminističnim algoritmom v času } T(n)\}$

$\text{NSPACE}(S(n)) = \{\text{vsi odločitveni problemi, rešljivi z nedeterminističnim algoritmom na prostoru } S(n)\}$

10.4 Stiskanje traku, linearna pohitritev in zmanjšanje števila trakov

- ◆ Pokazali bomo, da
 - ◆ prostorsko zahtevnost vedno lahko *zmanjšamo za konstantni faktor* (če merimo porabo prostora v številu potrebnih celic, lahko to število zmanjšamo tako, da kodiramo več tračnih simbolov v enega, tj. z večjo tračno abecedo; očitno pri tem velikost celice ignoriramo);
 - ◆ časovno zahtevnost vedno lahko *zmanjšamo za konstantni faktor* (če merimo porabo časa v številu korakov, lahko to število zmanjšamo z grupiranjem več zaporednih korakov v enega, tj. z definiranjem zahtevnejših ukazov; pri tem čas izvedbe zahtevnejšega ukaza ignoriramo)
- ◆ **Posledica:** v funkcijah $T(n)$ in $S(n)$ lahko *zanemarimo konstantni faktor* in se osredotočimo le na *hitrost naraščanja* funkcij $T(n)$ in $S(n)$

Stiskanje traku

- ◆ **Motivacija.** Prostor, ki ga TS potrebuje za izračun, smo definirali kot največje število celic, ki jih TS med izračunom uporabi na kakem od delovnih trakov.
Zamisel: zakodirajmo več simbolov z enim simbolom iz večje tračne abecede.
 - ◆ **Primer.** Razdelimo dano besedo 00110110 dožine 8 v pare 00 11 01 10 in kodirajmo pare simbolov iz stare abecede {0,1} s simboli nove abecede {0,1,2,3}, npr. takole: 00→0, 01→1, 10→2, 11→3. Beseda je zdaj 0312. Torej, *s povečanjem tračne abecede lahko zmanjšamo število porabljenih celic.*
- ◆ **Izrek.** Če ima jezik prostorsko zahtevnost $S(n)$, potem ima za poljuben $c > 0$ tudi prostorsko zahtevnost $cS(n)$. To velja tudi za nedet. prostorsko zahtevnost.
(Pozor: c je lahko <1 .)
Dokaz. Podoben kot v zgornjem primeru. □

Korolar: Za poljuben $c > 0$ je $\text{DSPACE}(S(n)) = \text{DSPACE}(cS(n))$
in $\text{NSPACE}(S(n)) = \text{NSPACE}(cS(n))$

Linearna pohitritev

- ◆ Ali velja kaj podobnega tudi za časovno zahtevnost? Časovno zahtevnost merimo v številu korakov, ki jih med izračunom opravi TS. Zamisel: združimo več zaporednih korakov v nov, večji korak (večji ukaz). Izkaže se, da je to vedno možno, če sta izpolnjena dva pogoja:
 - ◆ TS mora imeti vsaj 2 trakova (tj. $k \geq 2$)
 - ◆ Veljati mora $\inf_{n \rightarrow \infty} T(n)/n = \infty$.

Intuitivno: $T(n)$ mora naraščati vsaj malo hitreje kot n . Potem bo po branju vhodne besede ostalo vsaj malo časa za računanje. ($\inf_{n \rightarrow \infty} T(n)/n = \lim_{n \rightarrow \infty} \text{glb}\{T(n)/n, T(n+1)/(n+1), T(n+2)/(n+2), \dots\}$)
- ◆ **Izrek.** Naj bo $\inf_{n \rightarrow \infty} T(n)/n = \infty$ in $k \geq 2$. Potem:
Če ima jezik časovno zahtevnost $T(n)$, potem ima za poljuben $c > 0$ tudi časovno zahtevnost $cT(n)$. To velja tudi za nedet. časovno zahtevnost. (**Pozor:** c je lahko < 1 .)

Korolar: Če $\inf T(n)/n = \infty$, potem je za poljuben $c > 0$
 $\text{DTIME}(T(n)) = \text{DTIME}(cT(n))$
in $\text{NTIME}(T(n)) = \text{NTIME}(cT(n))$

Povzetek

- Pri $c > 0$ in dveh razumnih pogojih velja:

$$\text{DTIME}(T(n)) = \text{DTIME}(cT(n))$$

$$\text{NTIME}(T(n)) = \text{NTIME}(cT(n))$$

$$\text{DSPACE}(S(n)) = \text{DSPACE}(cS(n))$$

$$\text{NSPACE}(S(n)) = \text{NSPACE}(cS(n))$$

- To pomeni, da pozitivne konstante c ne vplivajo na vsebino teh razredov.

- Primer: $\text{DTIME}(0.33 n^2) = \text{DTIME}(4n^2) = \text{DTIME}(7n^2) = \dots = \text{DTIME}(n^2) =$

- Namesto

“ D je v razredu $\text{DTIME}(n^2)$ ”

rečemo tudi:

“ D ima (det.) časovno zahtevnost reda $O(n^2)$.

Zmanjšanje števila trakov v TS

- ♣ Časovno zahtevnost smo definirali s pomočjo TS, ki ima $k \geq 1$ trakov.
Vprašanje: Če ima TS s k trakovi časovno zahtevnost $T(n)$, kolikšna je časovna zahtevnost ekvivalentnega TS z manj trakovi?
Odgovor: Zmanjšanje k na 1 poveča časovno zahtevnost s $T(n)$ na največ $T^2(n)$, zmanjšanje k na 2 pa poveča časovno zahtevnost s $T(n)$ na največ $T(n) \log T(n)$.
Izrek. Če $L \in \text{DTIME}(T(n))$ pri $k > 1$, potem je $L \in \text{DTIME}(T^2(n))$ pri $k=1$.
Če $L \in \text{NTIME}(T(n))$ pri $k > 1$, potem je $L \in \text{NTIME}(T^2(n))$ pri $k=1$.
Če $L \in \text{DTIME}(T(n))$ pri $k > 2$, potem je $L \in \text{DTIME}(T(n)\log T(n))$ pri $k=2$.
Če $L \in \text{NTIME}(T(n))$ pri $k > 2$, potem je $L \in \text{NTIME}(T(n)\log T(n))$ pri $k=2$.
- ♣ Prostorsko zahtevnost smo definirali s TS, ki ima $k \geq 1$ delovnih trakov in 1 vh. trak.
Vprašanje: Kako zmanjšanje števila k vpliva na prostorsko zahtevnost ekvivalentnega TS?
Odgovor: Zmanjšanje števila delovnih trakov ne vpliva na prostorsko zahtevnost TS.
Izrek. Če $L \in \text{DSPACE}(S(n))$ pri $k > 1$, potem je $L \in \text{DSPACE}(S(n))$ pri $k=1$.
Če $L \in \text{NSPACE}(S(n))$ pri $k > 1$, potem je $L \in \text{NSPACE}(S(n))$ pri $k=1$.

10.5 Relacije med razredi DTIME, DSPACE, NTIME, NSPACE

- ◆ Ali med naštetimi razredi zahtevnosti obstajajo kake *relacije vsebovanja* \subseteq ? Zanimajo nas vsebovanja
 - ◆ med razredi *iste vrste* (npr. med razredi $\text{DTIME}(T(n))$ pri različnih funkcijah $T(n)$)
 - ◆ med razredi *različnih vrst* (npr. med razredoma DTIME in NTIME pri neki $T(n)$)
- ◆ Videli bomo, da
 - ◆ vsaka vrsta razredov CLASS (= DTIME, NTIME, DSPACE, NSPACE) tvori neskončno hierarhijo $\text{CLASS}(f_1(n)) \subsetneq \text{CLASS}(f_2(n)) \subsetneq \text{CLASS}(f_3(n)) \subsetneq \dots$, kjer so $f_i(n)$, $i = 1, 2, \dots$ ustrezne funkcije;
 - ◆ zamenjava nedeterminističnega algoritma z ekvivalentnim determinističnim algoritmom povzroči
 - ◆ kvečjemu eksponentno povečanje časovne zahtevnosti;
 - ◆ kvečjemu kvadratno povečanje prostorske zahtevnosti.

Relacije med razredi zahtevnosti iste vrste

- ◆ Hierarhije, izrek o vrzeli, ...

ZAENKRAT PRESKOČIM.

Relacije med razredi zahtevnosti različnih vrst

- ◆ Naslednji izrek opisuje relacije vsebovanja med različnimi vrstami razredov zahtevnosti.
Izrek.
 - ◆ $\text{DTIME}(T(n)) \subseteq \text{DSPACE}(T(n))$
Kar se da rešiti *deterministično v času* reda $O(T(n))$, se da rešiti *deterministično na prostoru* reda $O(T(n))$.
 - ◆ $L \in \text{DSPACE}(S(n)) \wedge S(n) \geq \log_2 n \Rightarrow \exists c : L \in \text{DTIME}(c^{S(n)})$
Kar se da rešiti *deterministično na prostoru* reda $O(S(n))$, se da rešiti *deterministično v času* reda $O(c^{S(n)})$. Konstanta c je odvisna od L .
 - ◆ $L \in \text{NTIME}(T(n)) \Rightarrow \exists c : L \in \text{DTIME}(c^{T(n)})$
Kar se da rešiti *nedeterministično v času* reda $O(T(n))$, se da rešiti *deterministično v času* reda $O(c^{T(n)})$.
Posledica: **zamenjava nedeterminističnega algoritma z ekvivalentnim determinističnim algoritmom povzroči kvečjemu eksponentno povečanje časa, ki je potreben za izračun rešitve primerka problema.**
 - ◆ $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S^2(n))$, če $S(n) \geq \log_2 n \wedge S(n)$ popolnoma prostorsko predstavljiva.
Kar se da rešiti *nedeterministično na prostoru* reda $O(S(n))$, se da rešiti *deterministično na prostoru* reda $O(S(n)^2)$.
Posledica: **zamenjava nedeterminističnega algoritma z ekvivalentnim determinističnim algoritmom povzroči kvečjemu kvadratno povečanje prostora, ki je potreben za izračun rešitve primerka problema.**
Opomba: Ta izrek se imenuje Savitchev izrek.

Predstavljive funkcije zahtevnosti

- ◆ Da bi se izognili *patološkim* primerom (ki so sicer nezanimivi in redki), moramo zahtevati, da sta funkciji $S(n)$, $T(n)$ ``predstavljeni.'' Opišimo, kaj to pomeni.
 - ◆ **Definicija.** Funkcija $S(n)$ je **prostorsko predstavljava**, če obstaja TS M s prostorsko zahtevnostjo $S(n)$ in naslednjo lastnostjo: za vsak $n \in \mathbb{N}$ obstaja vhodna beseda dolžine n , pri kateri M med računanjem uporabi natanko $S(n)$ celic traku. Če pa za vsak $n \in \mathbb{N}$ pri vsaki vhodni besedi dolžine n stroj M uporabi natanko $S(n)$ celic, rečemo, da je $S(n)$ **popolnoma prostorsko predstavljava**. Funkcija $T(n)$ je **časovno predstavljava**, če obstaja TS M s časovno zahtevnostjo $T(n)$ in naslednjo lastnostjo: za vsak $n \in \mathbb{N}$ obstaja vhodna beseda dolžine n , pri kateri M med računanjem napravi natanko $T(n)$ korakov. Če pa za vsak $n \in \mathbb{N}$ pri celo vsaki vhodni besedi dolžine n stroj M napravi natanko $T(n)$ korakov, rečemo, da je $S(n)$ **popolnoma časovno predstavljava**.
- ◆ Razred prostorsko prestavljenih in razred časovno predstavljenih funkcij sta zelo bogata. Vsebujejo vse običajne funkcije, večina jih je celo popolnoma predstavljenih.
- ◆ **Opomba.** Enakovreden izraz za *predstavljeni* funkcijo je *verna* funkcija.

10.6 Razredi P, NP, PSPACE, NPSPACE

- ◆ Velik **praktični** pomen imajo razredi zahtevnosti
 - ◆ $\text{DTIME}(T(n))$,
 - ◆ $\text{NTIME}(T(n))$,
 - ◆ $\text{DSPACE}(S(n))$,
 - ◆ $\text{NSPACE}(S(n))$,

kjer sta funkciji $T(n)$ in $S(n)$ poljubna **polinoma**.

Zakaj polinomi?

- Zahteva računanja po nekem *računskem viru* (času, prostoru, energiji, procesorjih,...) je **razumna**, če je *navzgor omejena s polinomom*. Npr, reševanje primerkov velikosti n (nekega problema), ki zahteva $T(n)$ časa, je razumno, če je $T(n) \leq \text{poly}(n)$, za vse n od *nekega* n_0 dalje.
- Kaj če zahteva n^i polinomsko omejena? Tabela kaže, da **eksponentna** časovna zahtevnost, npr. $T(n) = 2^n$ ali $T(n) = 3^n$, postane *nesprejemljivo velika* (v primerjavi s polinomsko zahtevnostjo) celo pri majhnih n (npr. $n > 20$).

$T(n)$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
n	0,00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s	0.00006 s
n^2	0.0001 s	0.0004 s	0.0009 s	0.0016 s	0.0025 s	0.0036 s
n^3	0.001 s	0.008 s	0.027 s	0.064 s	0.125 s	0.216 s
n^5	1 s	3.2 s	24.3 s	1.7 min	5.2 min	13.0 min
...
2^n	0.001 s	1.0 s	17.9 min	12.7 dni	35.7 let	366 stoletij
3^n	0.059 s	58 min	6.5 let	3855 stoletij	$2 \cdot 10^8$ stoletij	$1.3 \cdot 10^{13}$ stoletij

P, NP, PSPACE, NPSPACE

- ◆ **Definicija.** Razredi zahtevnosti P, NP, PSPACE in NPSPACE so definirani:
 - ◆ **P** = $\bigcup_{i \geq 1} \text{DTIME}(n^i)$
razred vseh odločitvenih problemov, ki so *deterministično* rešljivi v *polinomsko omejenem času*
 - ◆ **NP** = $\bigcup_{i \geq 1} \text{NTIME}(n^i)$
razred vseh odločitvenih problemov, ki so *nedeterministično* rešljivi v *polinomsko omejenem času*
 - ◆ **PSPACE** = $\bigcup_{i \geq 1} \text{DSPACE}(n^i)$
razred vseh odločitvenih problemov, ki so *deterministično* rešljivi na *polinomsko omejenem prostoru*
 - ◆ **NPSPACE** = $\bigcup_{i \geq 1} \text{NSPACE}(n^i)$
razred vseh odločitvenih problemov, ki so *nedeterministično* rešljivi na *polinomsko omejenem prostoru*

Osnovne relacije (med P, NP, PSPACE, NPSPACE)

- ◆ **Izrek.** Veljajo naslednja vsebovanja: $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}$

Dokaz.

- ◆ (**P ⊆ NP**) Vsak DTS polinomske časovne zahtevnosti je *trivialni* NTS (vsak ukaz da na izbiro kvečjemu en prehod) enake (polinomske) časovne zahtevnosti.
- ◆ (**NP ⊆ PSPACE**) Če je $L \in \mathbf{NP}$, potem (po definiciji) $\exists k$, da je $L \in \text{NTIME}(n^k)$. Zato je (po enem od osnovnih izrekov) $L \in \text{NSPACE}(n^k)$, in zato (po Savitchevem izreku) $L \in \text{DSPACE}(n^{2k})$. Sledi (iz definicije **PSPACE**), da je $L \in \mathbf{PSPACE}$.
- ◆ (**PSPACE = NPSPACE**) Izrek dokažemo v dveh korakih: najprej \subseteq , potem \supseteq .
 - ◆ (**PSPACE ⊆ NPSPACE**) Vsak DTS polinomske prostorske zahtevnosti je *trivialni* NTS (v katerem vsak ukaz da na izbiro ≤ 1 prehod) enake (polinomske) prostorske zahtevnosti.
 - ◆ (**NPSPACE ⊆ PSPACE**) $\mathbf{NPSPACE} = (\text{definicija } \mathbf{NPSPACE}) = \bigcup_{i \geq 1} \text{NSPACE}(n^i) \subseteq (\text{Savitchev izrek}) \subseteq \bigcup_{i \geq 1} \text{DSPACE}(n^{2i}) \subseteq (\text{definicija } \mathbf{PSPACE}) \subseteq \mathbf{PSPACE}$.



10.7 Problem $P =? NP$

- Pravkar smo dokazali: **$PSPACE = NPSPACE$** .
To lahko razumemo takole:

*Kadar je prostorska zahtevnost polinomska,
nedeterminizem ne poveča računske moči.*

- Ali kaj podobnega velja tudi za časovno zahtevnost? Vemo že, da je $P \subseteq NP$.
Vprašamo:
 - Ali je **$P = NP$** ?
 - ali pa je **$P \subsetneq NP$** ?
- Kljub intenzivnim raziskavam v zadnjih desetletjih, je vprašanje **$P =? NP$** brez odgovora; predstavlja **glavni nerešeni problem teoretičnega računalništva**.

Zakaj je vprašanje $P =? NP$ pomembno ?

- ◆ Mnogo pomembnih odločitvenih problemov je v **NP**. Za vsak tak problem D obstaja nedeterministični algoritem N_D , ki reši D v nedet. polinomsko omejenem času.
- ◆ Toda nedeterministični algoritmi *niso realistični*; noben realni računski stroj jih ne more neposredno izvajati. (Kako naj bi *realni* računalnik *brez napak* izbral *vedno pravo* izmed več alternativnih potez?)
- ◆ Zato moramo *nadomestiti* N_D z *ekvivalentnim determinističnim* algoritmom A_D , ki bo izračunal isti rezultat kot N_D , morda (ne nujno) s *sistematičnim simuliranjem* vsakega nedeterminističnega ukaza algoritma N_D z *zaporedjem determinističnih ukazov*.
- ◆ Zato bo A_D zahteval *več časa* (kot ga porabi N_D), da bo izračunal isti rezultat kot N_D .

Vprašanje: Koliko časa rabi A_D za reševanje D (tj. za reševanje primerkov problema D)?

- ◆ Spomino se izreka: $L \in \text{NTIME}(T(n)) \Rightarrow \exists c : L \in \text{DTIME}(c^{T(n)})$. Izrek nam pove, da zamenjava N_D z ekvivalentnim A_D povzroči kvečjemu eksponentno povečanje časa, ki je potreben za reševanje problema D .
- ◆ V našem primeru je $D \in \mathbf{NP}$, tj. $D \in \text{NTIME}(n^k)$ za neki $k \geq 1$. Po zgornjem izreku je zato $D \in \text{DTIME}(c^{n^k})$. Torej je D deterministično rešljiv v času $\text{konst} \times c^{n^k}$. Pozor: to je zgornja meja za det. čas. zaht. problema D ; njegova dejanska det. čas. zaht. bi bila lahko tudi nižja od $\text{konst} \times c^{n^k}$ (a bi še vedno veljalo $D \in \text{DTIME}(c^{n^k})$). Zato se zastavi vprašanje:
- ◆ *Ali lahko deterministični A_D kljub zgornji meji $O(c^{n^k})$ reši D v polinomsko omejenem času?*
 - ◆ Denimo, da lahko. Ali potem to velja še za *kake druge* probleme v \mathbf{NP} ? Ali morda velja celo za *vse* probleme v \mathbf{NP} ? To je enakovredno vprašanju ``Ali je $\mathbf{P} = \mathbf{NP}$ '' (na kratko: $\mathbf{P} = ? \mathbf{NP}$).
- ◆ Če v resnici velja $\mathbf{P} = \mathbf{NP}$ (tega ne vemo!), potem je *vsak* $D \in \mathbf{NP}$ deterministično rešljiv v *polinomsko omejenem času*. Zato lahko vprašanje $\mathbf{P} = ? \mathbf{NP}$ razumemo tudi takole:
Če je časovna zahtevnost polinomska, ali nedeterminizem ne poveča računske moči?

- ◆ **Kako se lotiti iskanja odgovora na $P =? NP$?**
- ◆ Prevladujoče *mnenje* je, da $P \neq NP$ (torej $P \subsetneq NP$).
Zakaj? Če bi veljalo $P = NP$, bi bile nekatere posledice “preveč” presenetljive (skorajda neverjetne).
- ◆ Zato poskušamo dokazati, da je $P \subsetneq NP$.
- ◆ Kako? Obstajajo razne **metode**. Med njimi je naslednja:
 1. poiščimo “najtežji” problem X v **NP**;
 2. dokažimo, da ta problem X ni v **P**.

Podlaga za to metodo je *intuicija*, ki nam pravi tole:

- ◆ Če *sploh* obstaja kak problem v **NP-P**, potem je to gotovo tisti, recimo mu X, ki je “najtežji” v **NP** (ali pa *eden od* “najtežjih”, če je takih več).
- ◆ Za tak X bo gotovo *lažje* (ali pa vsaj *ne težje*) dokazati, da ni v **P**, kot bi bilo dokazati, da kak drug problem, “lažji” od X, ni v **P**.

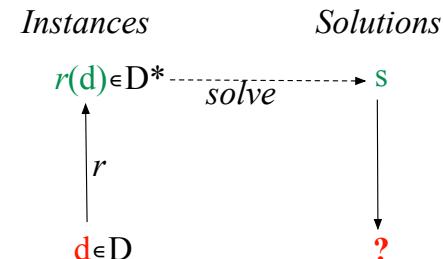
Prevedbe problemov

- Kdaj je nek problem ``najtežji'' v **NP**? Kako naj to lastnost sploh definiramo?
Intuitivno: Problem D^* je ``najtežji'' v **NP**, če je vsak drug $D \in \text{NP}$ ``kvečjemu tako težek kot'' D^* .

- **Zamisel:** Denimo, da bi v **NP** obstajal D^* , na katerega bi se dal ,,,hitro prevesti'' vsak $D \in \text{NP}$.

Prevedbo **definirajmo** takole: problem D se dá ``hitro'' **prevesti** na problem D^* ,

- če obstaja funkcija $r : D \rightarrow D^*$,
- ki ``hitro'' preslika poljuben primerek $d \in D$ v primerek $r(d) \in D^*$,
- tako da se rešitev s primerka $r(d)$ dá ``hitro'' transformirati v rešitev ``?'' primerka d .



Intuitivno: Reševanje problema D bi se dalo ``hitro'' nadomestiti z reševanjem problema D^* . Zato bi bil D kvečjemu tako težek kot D^* ! (Tu nič ne govorimo o težavnosti reševanja D^* .)

- Če bi tak $D^* \in \text{NP}$ obstajal, bi bil vsak drug $D \in \text{NP}$ kvečjemu tako težek kot D^* ; z drugimi besedami: D^* bi bil *najtežji problem v NP* (ali pa eden od njih).

Polinomske-časovne prevedbe

- ◆ Še vedno smo dolžni pojasniti, kaj pomeni, da se dá D *hitro* prevesti na D^* . Definirajmo: ``hitro'' = v determinističnem polinomskem času.
- ◆ Zdaj imamo vse pripravljeno za naslednjo definicijo:

Definicija. Problem $D \in \mathbf{NP}$ je **polinomsko-časovno prevedljiv** na problem D' , tj. $D \leqslant^p D'$, če obstaja deterministični TS M polinomske časovne zahtevnosti, ki preslikava poljuben primerek $d \in D$ v primerek $d' \in D'$, tako velja d je pozitiven $\Leftrightarrow d'$ je pozitiven.

Relaciji \leqslant^p rečemo **polinomska-časovna prevedba** (*polinomska prevedba*, kadar je jasno, da gre za čas).

Intuitivno: M v polinomskem času nadomesti $d \in D$ z $d' \in D'$, ki ima enak odgovor kot d .

Kako to stori? M na podlagi vh.besede $\langle d \rangle$ vrne v času $\text{poly}(|\langle d \rangle|)$ besedo $M(\langle d \rangle)$, za katero velja $M(\langle d \rangle) \in L(D') \Leftrightarrow \langle d \rangle \in L(D)$. (Tu je seveda $M(\langle d \rangle) = \langle d' \rangle$, koda primerka d' .)
- ◆ **Ugotovili smo:** Najtežji problem v **NP** je vsak problem D^* , za katerega velja
 - ◆ $D^* \in \mathbf{NP}$ in
 - ◆ Za vsak $D \in \mathbf{NP}$: $D \leqslant^p D^*$.

V naslednjem razdelku bomo rekli, da je tak problem **NP-poln**.

10.8 NP-polni in NP-težki problemi

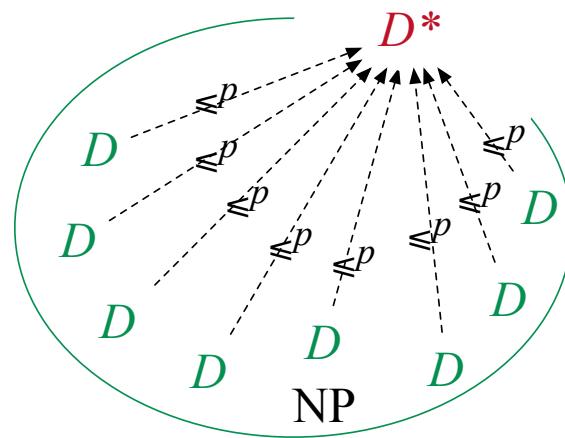
- ◆ Zdaj uvedemo pojma **NP-polnega** in **NP-težkega** problema.
- ◆ Razložimo, da **obstaja NP-poln** odločitveni problem.
- ◆ Opisemo običajno **metodo** za dokazovanje **NP-polnosti** oz. **NP-težkosti** drugih računskih problemov.

NP-polni in NP-težki problemi

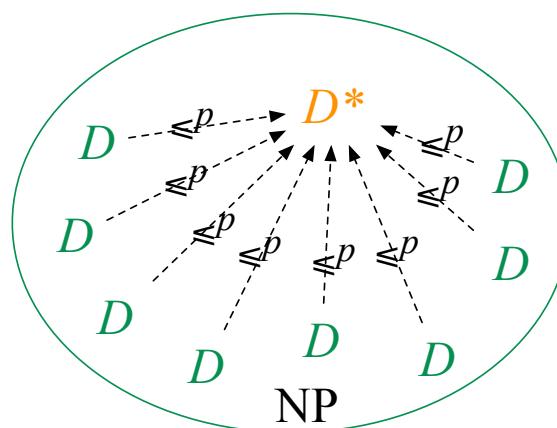
- ◆ Videli smo, da je *najtežji* problem v **NP** vsak problem D^* , ki izpolnjuje pogoja
 - ◆ $D^* \in \mathbf{NP}$
 - ◆ $D \leqslant^p D^*$, za vsak $D \in \mathbf{NP}$
- ◆ Taki problemi so tako pomembni, zato si zaslužijo posebno ime.
Definicija. Problem D^* je **NP-težek**, če za vsak $D \in \mathbf{NP}$ velja $D \leqslant^p D^*$.
Problem D^* je **NP-poln**, če velja
 - ◆ $D^* \in \mathbf{NP}$ in
 - ◆ $D \leqslant^p D^*$, za vsak $D \in \mathbf{NP}$.
- ◆ Torej: problem je **NP-poln** \Leftrightarrow problem je v razredu **NP** \wedge problem je **NP-težek**.

(nadaljevanje)

- NP-polnost ozziroma NP-težkost problema D^* lahko ilustriramo takole:



D^* is NP-hard



D^* is NP-complete

Črtkane puščice predstavljajo polinomske-časovne prevedbe $D \leqslant^p D^*$ (za posamične $D \in \text{NP}$)
Pozor: NP-težek problem je lahko v NP ali izven NP.

Obstaja NP-poln problem

- ◆ **Vprašanje.** Ali sploh obstaja kak **NP-poln** problem; tj., ali D^* sploh obstaja?
Odgovor. Da, obstaja. Danes poznamo že na tisoče takih problemov!
Prvega sta 1971 neodvisno odkrila Stephen Cook in Leonid Levin.
- ◆ **Definicija. Logični izraz** (Boolov izraz) induktivno definiramo takole:
 - ◆ Logične spremenljivke x_1, x_2, \dots so logični izrazi.
 - ◆ Če sta E in F logična izraza, so $\neg E$, $E \vee F$ in $E \wedge F$ logični izrazi.
- ◆ **Definicija.** Logični izraz E je **izpolnljiv**, če obstaja prireditev logičnih vrednosti RESNIČNO/NERESNIČNO njegovim spremenljivkam, da ima E logično vrednost RESNIČNO.
- ◆ **Definicija.** Problem **izpolnljivosti** je odločitveni problem
 $SAT = ``\text{Ali je logični izraz } E \text{ izpolnljiv?}''$
- ◆ **Izrek (Cook-Levin).** SAT je **NP-poln** problem.
- ◆ **Torej:** Za problem D^* lahko vzamemo SAT.



► Ideja dokaza.

Letos preskočim.



Dokazovanje NP-polnosti problemov

- V nadaljevanju bomo potrebovali naslednja dva izreka:

Izrek. Naj bo $D \leq^p D'$. Potem velja:

- $D' \in \mathbf{P} \Rightarrow D \in \mathbf{P}$
- $D' \in \mathbf{NP} \Rightarrow D \in \mathbf{NP}$.

Intuitivno: Če je problem D prevedljiv v polinomskem času na kak problem v \mathbf{P} (oz. v \mathbf{NP}), potem je tudi sam D v \mathbf{P} (oz. v \mathbf{NP}).

- **Izrek.** Relacija \leq^p je tranzitivna.

Torej: $D \leq^p D' \wedge D' \leq^p D'' \Rightarrow D \leq^p D''$.

(nadaljevanje)

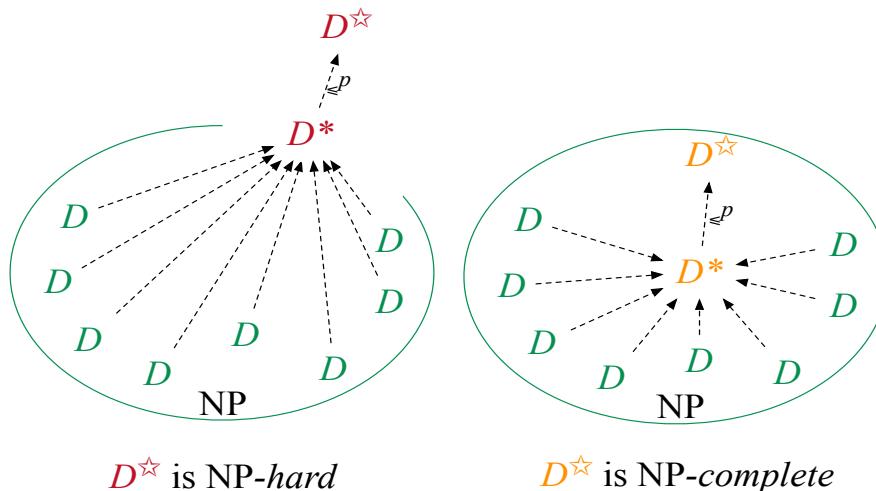
• **Korolar.** Velja sledeče:

- D^* je NP-težek $\wedge D^* \leqslant^p D^\star \Rightarrow D^\star$ je NP-težek
- D^* je NP-poln $\wedge D^* \leqslant^p D^\star \wedge D^\star \in \text{NP} \Rightarrow D^\star$ je NP-poln

• Korolar nam razkrije *metodo* za *dokazovanje* NP-težkosti oz. NP-polnosti problema D^\star :

Za dokaz, da je D^\star
NP-težek:

Nek znano NP-težki problem D^* prevedi v polinomskem času na D^\star .



Za dokaz, da je D^\star
NP-poln:

Nek znano NP-polni problem D^* prevedi v polinomskem času na D^\star in dokaži, da je D^\star v NP.

Primeri NP-polnih problemov

- S to metodo je bila dokazana NP-polnost oz. NP-težkost že nekaj tisoč problemov. Spodaj navajamo samo tri (za več glejte npr. Wikipedia: List of NP-complete problems).

- RAZDELITEV (PARTITION)

Primerek: Končna množica A naravnih števil.

Vprašanje: Ali obstaja podmnožica $B \subseteq A$, da je $\sum_{a \in B} a = \sum_{a \in A - B} a$?

- HAMILTONOV CIKEL (HAMILTONIAN CYCLE)

Primerek : Graf $G(V, E)$.

Vprašanje : Ali $G(V, E)$ vsebuje Hamiltonov cikel?

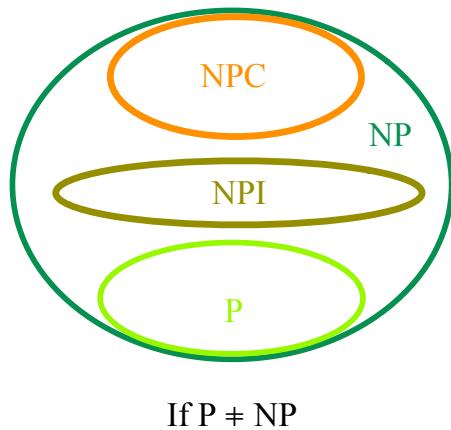
- KOŠI (BIN PACKING)

Primerek : Končna množica U naravnih števil in naravni števili c in k .

Vprašanje : Ali obstaja razdelitev množice U na disjunktne množice U_1, U_2, \dots, U_k , tako da je vsota števil v vsaki U_i kvečjemu c ?

Povzetek

- Če je $P \neq NP$, potem so razmere v razredu **NP** naslednje:



Tu so:

- **NPC** je razred vseh **NP-polnih** problemov.
- **NPI** je razred vseh **NP-vmesnih** problemov. Vmesnih??

Izrek (Ladner): Če $P \neq NP$, potem v **NP** obstaja problem, ki ni niti v **P** niti v **NPC**.

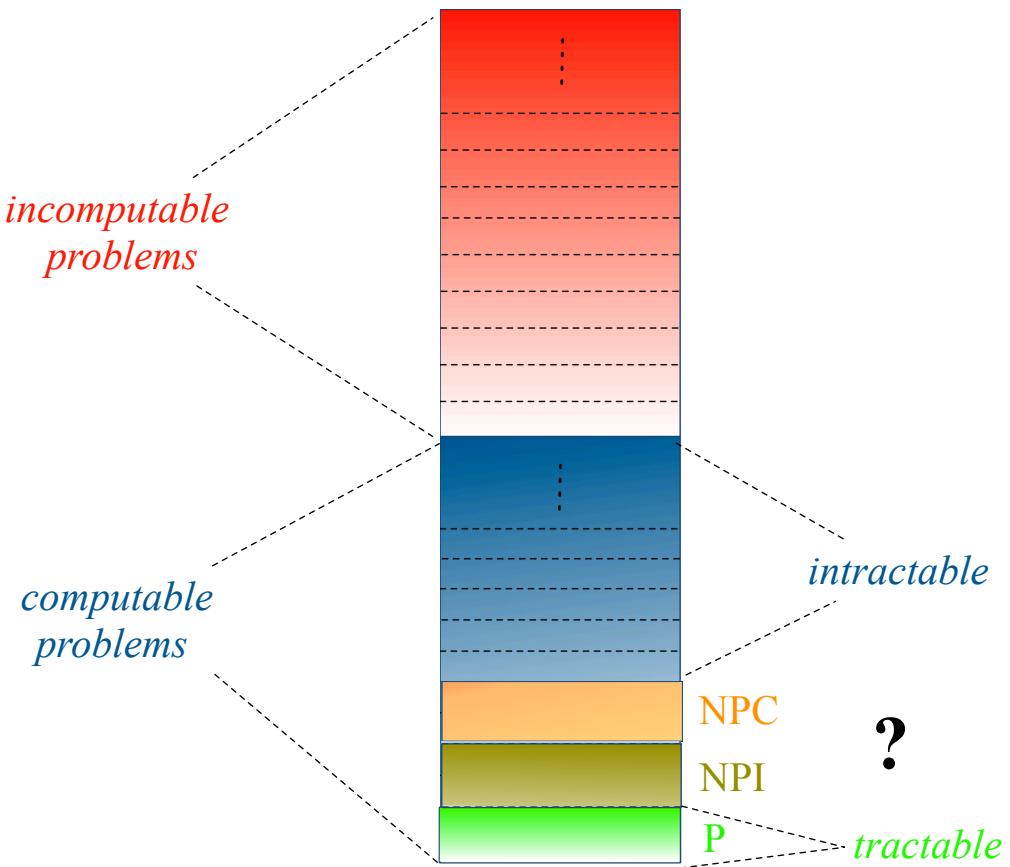
Rekli bomo, da je tak problem **NP-vmesen**.
Kandidat za **NP-vmesni** problem je odločitveni problem
„Ali je n sestavljeno število?“

Če je $P \neq NP$, potem velja:

Če je problem v **NPC** ali **NPI**,
potem njegova deterministična časovna zahtevnost *ni* polinomsko omejena.

(nadaljevanje)

- Pravimo, da so problemi, ki so v razredu **P**, **obvladljivi** (*tractable*). Ostali izračunljivi problemi so **neobvladljivi** (*intractable*).
- Izjema so problemi v razredih **NPC** in **NPI**. Za te probleme še ni znano, ali so obvladljivi ali neobvladljivi. (*Domnevamo pa, da so neobvladljivi.*)



10.9 Slovar

computational complexity računska zahtevnost computational resource računski vir (non)deterministic time complexity (ne)deterministična časovna zahtevnost complexity class razred zahtevnosti (non)deterministic space complexity (ne)deterministična prostorska zahtevnost tape compression stiskanje trakov linear speedup pohitritev reduction in the number of tapes zmanjšanje števila trakov “well-behaved” function „lepa, pohlevna” funkcija space/time constructible function prostorsko/časovno predstavljiva (ali verna) funkcija fully space/time constructible function popolnoma prostorsko/časovno predstavljiva (ali verna) funkcija polynomial polinom (non)deterministic polynomial time/space complexity (ne)deterministična polinomska časovna/prostorska zahtevnost hard/difficult problem teoretični problem problem reduction prevedba problema easy problem lahek problem polynomial-time reduction polinomska časovna prevedba logarithmic-space reduction logaritmična prostorska prevedba NP-complete problem NP-poln problem NP-hard problem NP-teoretični problem Boolean expression Boolov izraz satisfiable izpolnljiv satisfiability problem problem izpolnljivosti NP-intermediate problem NP-vmesni problem (in)tractable (ne)obvladljiv

11

Neobvladljivi problemi

Vsebina

- ▶ Prestavim na APS2.

12

Reševanje neobvladljivih problemov

Vsebina

- ▶ Prestavim na APS2

KONEC