Introducere in teoria limbajelor formale

Limbaje formale și translatoare (Compilatoare)

November 6, 2020

Mihai-Lica Pura

Cuprins

- Introducere: Definiție, importanță, istoric
- Gramatici formale
- Arbori de analiză gramaticală
- Limbaje regulare
  - automate finite
  - expresii regulare
- Limbaje independente de context
  - automate finite cu stivă

Definiția limbajului

- Știința care se ocupă cu studiul limbajelor (naturale) se numește *lingvistică*.
- Limbajul este un sistem de comunicare bazat pe cuvinte și pe combinarea cuvintelor pentru a forma propoziții și fraze.
  - fonologia - regulile după care simbolurile sunt utilizate pentru a forma cuvinte sau morfeme (microsintaxă)
  - sintaxa - regulile după care cuvintele sau morfemele se combină pentru a forma propoziții și fraze
  - semiotică - procesul prin care semnele sunt legate de anumite înțelesuri (semantică)

Definiția limbajului

- Comunicarea înseamnă schimbul de informații, cunoștințe, credințe, opinii, dorințe, ordine, amenințări, mulțumiri, promisiuni, declarații, sentimente, etc.
- Clasificare
  - Comunicare lingvistică - bazată pe un limbaj
  - Comunicare non-lingvistică - zâmbetul, râsul, strigatul, ridicatul din umeri, etc.

Importanța limbajului

- Limbajul arată felul în care noi percepem realitatea
- De exemplu:
  - în limba română spunem:
    Avionul zboară pe cer.
  - în limbile franceză și engleză spunem:
    L'avion vole dans le ciel.
    The plane flies in the sky.
- Iar acest lucru se regăsește și în felul în care au fost proiectate limbajele formale, adică felul în care noi prezentăm realitatea calculatoarelor

Importanța limbajului

Importanța limbajului

- Versiunea "Weltanschauung" extremă a ipotezei Sapir-Whorf
- Determinismul lingvistic
  - structura unui limbaj influențează sau determină percepția asupra lumii
  - percepția asupra lumii
    - descrie consistent și integral existența
    - oferă cadrul de lucru teoretic pentru a genera, susține și aplica cunoașterea

Paradigme de programare

Programarea structurată

- paradigmă de programare axată pe îmbunătățirea calității, clarității și timpului de dezvoltare a codului unui program, prin utilizarea in extenso a **funcțiilor (subrutinelor), blocurilor de instrucțiuni, și a instrucțiunilor repetitive (for, while, ș.a.)**
- versus
- utilizarea salturilor de tipul **goto**, care produc un cod dezordonat (spaghetti), care este foarte greau de întreținut și de depanat
- Exemple: C, C++, C#, Java, ș.a.

Paradigme de programare

Programarea imperativă

- paradigmă de programare care descrie computațiile în termeni de **enunțuri** care modifică **starea** programului
- seamănă **modului imperativ** din limbajul natural, prin care enunțăm **comenzi** care cer efectuarea unor anumite acțiuni
- programele imperative definesc succesiuni de comenzi, care să fie executate de către computer
- Exemple: C, C++, C#, Java, ș.a.

Bubble sort în C

```
void bubble_sort(long list[], long n) {
  long c, d, t;
  for (c = 0 ; c < ( n - 1 ); c++) {
    for (d = 0 ; d < n - c - 1; d++) {
      if (list[d] > list[d+1])
      {
        t          = list[d];
        list[d]    = list[d+1];
        list[d+1] = t;
      }
    }
  }
}
```

Paradigme de programare

Programarea declarativă

- paradigmă de programare care exprimă logica unei computații, fără a descrie fluxul ei de control
- vine în contraponderea paradigmei imperative, care are nevoie de un algoritm explicit precizat
- scopul este minimizarea și chiar eliminarea efectelor secundare, descriind **ceea ce** trebuie să facă un program, iar nu **cum** trebuie să procedeze pentru a face acel lucru
- Exemple: limbaje de programare funcționale (vezi slide-ul următor), limbaje de programare logice (Prolog, ș.a.)

Programarea funcțională

- paradigmă de programare care definește computația ca o evaluare a unor funcții matematice și care evită folosirea stării și a variabilelor
- este bazată pe aplicarea funcțiilor, spre deosebire de paradigma imperativă care este bazată pe schimbările stării programului
- Exemple: Haskell, Scala, F#, ș.a.

Bubble sort în Scala

```
def bubblesort[A <% Ordered[A]](list: List[A]):List[A] = {
  def sort(as: List[A], bs: List[A]): List[A] =
    if (as.isEmpty) bs
    else bubble(as, Nil, bs)

  def bubble(as: List[A], zs: List[A], bs: List[A]):List[A]
    = as match {
      case h1 :: h2 :: t =>
        if (h1 > h2) bubble(h1 :: t, h2 :: zs, bs)
        else bubble(h2 :: t, h1 :: zs, bs)
      case h1 :: Nil => sort(zs, h1 :: bs)
  }

  sort(list, Nil)
}
```

Paradigme de programare

Programarea orientată pe obiecte

- paradigmă de programare care folosește pentru computații obiectele (de obicei instanțe ale unei clase)
- un obiect constă din gruparea unor atribute (câmpuri de date) și a unor metode și a interacțiunii dintre ele
- permite utilizarea unor tehnici de programare ca: abstractizarea, încapsularea, polimorfismul, moștenirea, ș.a.
- Exemple: C++, C#, Java, ș.a.

Paradigme de programare

Programarea event-driven

- paradigmă de programare în care fluxul programului este determinat de evenimente (înregistrarea unor valori de către senzori; acțiunile utilizatorului - clic cu mouse-ul, apăsarea unei taste, etc.; recepționarea unor mesaje de la alte fire de execuție sau de la alte programe)
- Exemple: C Win32 API

Metafizica limbajelor de programare

- Platon, în opera *Republica* descrie noțiunea de **lume a ideilor**: toate lucrurile din univers au un corespondent abstract în lumea ideilor
- de exemplu, o masă din lumea reală este doar o reflexie imperfectă a ideii de Masă din lumea idelor
- elementele abstracte din lumea ideilor se constituie într-o ierarhie de tipuri, care are ca și rădăcină un tip suprem
- scopul filosofului este a descoperi care este acest tip suprem din care sunt derivate toate cele care există

Paradigme de programare

Metafizica limbajelor de programare

- Platon - alegoria peșterii - tipul suprem este bunătatea
- neoplatonicii - tipul suprem este Dumnezeu
- programarea orientată pe obiecte - tipul suprem este clasa din care sunt derivate toate celelalte clase
- C# și Java oferă o percepție platonică asupra lumii, având clasa *Object* din care sunt derivate toate celelalte clase

Metafizica limbajelor de programare

- Aristotel în *Fizica* consideră că fiecare obiect din univers este construit folosind elemente primitive discrete numite atomi
- de exemplu, pentru a construi o masă:
  - este nevoie ca atomii să se aranjeze în ordinea corectă pentru a forma lemnul
  - iar apoi, elementele de lemn sunt aranjate pentru a crea masa
- limbajele de programare aristoteliene sunt cele care se bazează pe primitive discrete (char, int, float, double) pe baza cărora construiesc apoi elemente din ce în ce mai complexe
- de exemplu: pentru a reprezenta un șir de caractere se folosește un vector de tip char, iar nu o clasă de tip string

Istoria limbajului - creația

- Istoria culturală a tuturor popoarelor a consemnat preocuparea pentru “limba perfectă” (până în sec. XVII-lea)
- Prima variantă: **Dumnezeu i-a dat lui Adam o formă a limbii ebraice (perfecte)**, prin care El i-ar fi vorbit lui Adam: *“A dat apoi Domnul Dumnezeu lui Adam poruncă și a zis: "Din toți pomii din rai poți să mănânci, Iar din pomul cunoștinței binelui și răului să nu mănânci, căci, în ziua în care vei mânca din el, vei muri negreșit!”* (Geneza, 2:16-17)

Istoria limbajului - creația

- A doua variantă: **Adam ar fi inventat-o dând un nume animalelor**:
  "Și Domnul Dumnezeu, Care făcuse din pământ toate fiarele câmpului și toate păsările cerului, le-a adus la Adam, ca să vadă cum le va numi; așa ca toate ființele vii să se numească precum le va numi Adam. Și a pus Adam nume tuturor animalelor și tuturor păsărilor cerului și tuturor fiarelor sălbatice." (Geneza, 2:19-20)
- Și în care ar fi avut primul dialog cu Eva.

Istoria limbajului - creația

- A treia variantă: **Dumnezeu i-a dat lui Adam o gramatică generală**, o formă transcedentală cu care să construiască toate limbile posibile ("De Vulgari Elequentia", Dante).
  - **Dumnezeu Chomskyan** i-a dat lui Adam niște structuri sintactice profunde, ce se regăsesc în orice limbă creată ulterior de către oameni.
  - Dumnezeu i-a dat lui Adam (Adam a identificat în construirea primei limbi) niște **universalii semantice - un sistem de noțiuni atomice** - prin combinarea cărora se poate da seamă de întreg mobilierul universului.

Istoria limbajului - apariția limbilor popoarelor

- *"În vremea aceea era în tot pământul o singură limbă și un singur grai la toți."*(Geneza, 11: 11)
- **Versus**
- *"Din acestia [fii lui Noe] s-au format mulțime de popoare, care s-au așezat în diferite țări, fiecare după limba sa, după neamul său și după nația sa."*(Geneza, 10: 5)
- *"Aceștia sunt fiii lui Ham, după familii, limbă, țări și după nații."*(Geneza, 10: 20)
- *"Acestia sunt fiii lui Sem după familii, după limbă, după țări și după nații."*(Geneza, 10: 31)

Istoria limbajului - Gândire și limbaj

- Inițial se credea că există o gramatică universală a ideilor care reflectă însăși organizarea universului.
- Implică faptul că există o structură de noțiuni universale prezente în limba și în gândirea oricarui popor.
- Există deci un sistem al ideilor, același pentru toți oamenii, iar de la popor la popor aceleiași idei i se dau nume diferite.
- Ex: ideogramele chinezești
  - aceleași pentru chinezi, japonezi și coreeni
  - pronunțate cu sunete diferite pentru fiecare popor în parte
  - trimit la aceleași concepte (pentru aceste popoare diferite)

Istoria limbajului - Inventarea unui limbaj

- Inventarea unor limbi (ulterior numite filosofice și a priori) – ele nu sunt descoperite, ci construite pe baza unei viziuni filosofice asupra lumii.
- Scopurile căutării anterioare:
  - Convertirea necredincioșilor la creștinism,
  - Regăsirea comuniunii mistice cu Dumnezeu și cu lucrurile pe care le însemna limba perfectă a lui Adam.
- Scopurile invențiilor:
  - Favorizarea schimburilor comerciale,
  - Pătrunderea colonială,
  - Răspândirea științei.

Istoria limbajului - Inventarea unui limbaj

- “Common writing”, Lodwick (1647)
- “Ars Signorum”, Dalgarnus (1661)
- “Essay Towards a Real Character”, John Wilkins (1668)

- “Les estats et les empires de la lune”, Cyrano de Bergerac
- “Les estats et les empires du soleil”, Cyrano de Bergerac
- “The Man in the Moon”, Francis Godwin
- “La terre australe connue”, Gabriel de Foigny
- “L’Histoire des Sevarambes”, Deinis Veiras

Limbaje formale

- FORMÁL, -Ă, formali, -e, adj.
  1. Privitor la formă, care ține de formă, de aparență. (Adverbial) În aparență.
  2. Formulat precis; categoric, expres.
  3. Pătruns de formalism; făcut de formă (7).
  4. (Despre unele acte juridice) Care necesită anumite forme pentru a fi socotit legal și valabil.

  (dexonline)
- Limbajele naturale nu se bazează pe reguli stricte
- Rezultatul: *ambiguitatea semnatică* - același cuvânt sau aceeași propoziție/construcție/frază poate să aibă mai multe înțelesuri, adică poate să fie interpretată în mai multe moduri

Limbaje formale

Limbaje formale

- pentru oameni, acest lucru nu constituie neapărat o problemă
- oamenii deduc sensul imediat folosindu-se de context

> TAKE ADVANTAGE OF THE AMBIGUITY IN THE WORLD. LOOK AT SOMETHING AND THINK WHAT ELSE IT MIGHT BE.
>
> – Robert von Oech

- pentru calculatoare, ambiguitatea este o problema - soluția limbajele formale

Limbaje formale

- Limbajul este o modalitate sistematică de comunicare care utilizează sunete sau *simboluri convenționale*.
- În cazul limbajelor de programare, simbolurile convenționale sunt *șiruri de caractere*.
- Pentru a defini un șir de caractere trebuie să pornim de la un alfabet.

Limbaje formale

- Un **alfabet** este o mulțime finită de simboluri.
- Exemple:
  - $\Sigma_1 = \{$a, ă, â, b, c, d, $\ldots$, z$\}$ este mulțimea literelor din alfabetul limbii române.
  - $\Sigma_2 = \{0, 1, \ldots, 9\}$ este mulțimea cifrelor în baza 10.
  - $\Sigma_3 = \{$a, b, $\ldots$, z, $\#\ \}$ este mulțimea literelor din alfabetul latin, plus simbolul special $\#$.

Limbaje formale

- O succesiune finită de simboluri din alfabetul Σ se numește **șir de simboluri** peste alfabetul dat.
- Exemple:
  - *abfbz* este șir de caractere peste $\Sigma_1 = \{$a, b, c, d, $\ldots$, z$\}$,
  - *9021* este șir de caractere peste $\Sigma_2 = \{0, 1, \ldots, 9\}$,
  - *ab#bc* este șir de caractere peste $\Sigma_3 = \{$a, b, $\ldots$, z, $\#\}$.
- Un **limbaj** este o mulțime de șiruri (finită sau infinită) de simboluri peste același alfabet.

Limbaje formale

- Dacă limbajul este finit atunci el poate să fie definit prin **enumerare**.
- Dacă limbajul este infinit atunci există două mecanisme de definire:
  - prin **generare**
    - gramaticile formale
    - știe să genereze toate propozițiile din limbaj (și numai pe acestea), astfel încât alegând o propoziție din limbaj, va ajunge sa genereze propoziția respectivă într-un interval finit de timp
  - prin **recunoaștere**
    - automatele finite și expresiile regulare
    - știu să recunoască (să accepte ca fiind corecte) propozițiile limbajului dat (și numai pe acestea)

Gramatici formale

Cuprins

- Intuiție
- Definiție
- Relația de derivare
  - derivarea directă
  - derivarea în k pași
  - închiderea tranzitivă a relației de derivare
  - închiderea tranzitivă și reflexivă a relației de derivare
- formă propozițională
- propoziție
- limbaj
- Ierarhia Chomsky. Tipuri de gramatici formale

Gramatici formale

Intuiție

- fie o mulțime de cuvinte:
  { casa, strălucește, rapid, copila, stă, frumos, câinele, aleargă, alb }
- să formăm propoziții de câte trei cuvinte alese la întâmplare, care să aibă înțeles în limba română
- evident, avem toate șansele de a forma propoziții care nu au niciun înțeles, cum ar fi, de exemplu
  "rapid casa bine"

Gramatici formale

Intuiție

- dar dacă am împărți cuvintele în două categorii, nume și verbe:
  Nume - { *casa, rapid, copila, frumos, câinele, alb* }
  Verbe - { *strălucește, stă, aleargă* }
- și am forma propoziții de câte trei cuvinte alese la întâmplare astfel: primul și al doilea cuvânt din prima mulțime, iar al treilea cuvânt din a doua mulțime
- avem șanse mult mai mari de a forma propoziții cu înțeles, cum ar fi, de exemplu
  "casa albă strălucește"
- dar mai sunt încă șanse de a forma propoziții cu o anumită lipsă de sens, cum ar fi, de exemplu
  "casa câinele strălucește"

Gramatici formale

Intuiție

- dar dacă am împărți cuvintele din prima mulțime în două categorii, substantive și adjective:
  Substantive - *{ casa, copila, câinele }*
  Adjective - *{ rapid, frumos, alb }*
  Verbe - *{ strălucește, stă, aleargă }*
- și am forma propoziții de câte trei cuvinte alese la întâmplare astfel: primul cuvânt din prima mulțime, al doilea cuvânt din a doua mulțime, iar al treilea cuvânt din a treia mulțime
- avem șanse maxime de a forma propoziții cu înțeles, cum ar fi, de exemplu
  "casa albă strălucește"

Gramatici formale

Intuiție

- Mulțimea de cuvinte de la care am plecat, împreună cu mulțimea elementelor ajutătoare ({ *Start*, *Substantiv*, *Adjectiv*, *Verb*}) și împreună cu cele patru înlocuiri de mai jos formează o gramatică formală:
- *Start* $\rightarrow$ *Substantiv Adjectiv Verb*
  Substantiv $\rightarrow$ *casa* | *copila* | *câinele*
  Adjectiv $\rightarrow$ *rapid* | *frumos* | *alb*
  Verb $\rightarrow$ *strălucește* | *stă* | *aleargă*

Gramatici formale

O **gramatică** formală este un cvadruplu $G = (V_N, \Sigma, S, P)$, unde:

- V_N se numește mulțimea neterminalelor,
- Σ se numește mulțimea terminalelor,
- S este simbolul inițial (sau simbolul de start), $S \in V_N$,
- P este mulțimea regulilor de producție, ale cărei elemente au forma:
  $V^* V_N V^* \to V^*$, unde $V = V_N \cup \Sigma$ se numește alfabetul gramaticii.

Gramatici formale

- Fie V o mulțime de simboluri (un alfabet). Definim, prin inducție, următoarele mulțimi:
  - $V_0 = \{\epsilon\}$,
  - $V_1 = V$,
  - $\ldots$
  - $V_{i+1} = \{wv | w \in V_i \ , \ v \in V\}, \forall i > 0.$

Gramatici formale

- Operatorul **steluța Kleene** asupra unei mulțimi V este definit astfel:

$$V^* = \bigcup_{i \in \mathbb{N}} V_i = \{\epsilon\} \cup V_1 \cup V_2 \cup V_3 \cup \ldots$$

- Definiția operatorului **plusul Kleene** asupra unei mulțime V este:

$$V^+ = \bigcup_{i \in \mathbb{N} \setminus \{0\}} V_i = V_1 \cup V_2 \cup V_3 \cup \ldots$$

- V^* este mulțimea tuturor șirurilor de simboluri peste V, iar V^+ este mulțimea tuturor șirurilor de simboluri peste V cu excepția șirului vid.

Gramatici formale

Convențiile de notație

- literele mici de la începutul alfabetului latin (a, b, c, ...) reprezintă elemente din mulțimea Σ (simboluri terminale),
- literele mici de la sfârșitul alfabetului latin (u, v, x, ...) reprezintă elemente din mulțimea Σ^* (șiruri de simboluri terminale),
- literele mari de la începutul alfabetului latin (A, B, C, ...) reprezintă elemente din V_N (simboluri neterminale),
- literele mari de la sfârșitul alfabetului latin (U, V, X, ...) reprezintă elemente din $V_N \cup \Sigma$ (simboluri terminale sau neterminale),
- literele alfabetului grecesc reprezintă șiruri din $(V_N \cup \Sigma)^*$ (șiruri de simboluri terminale și neterminale).

Gramatici formale

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- V_N = {Subiect, Predicat, Propoziție},
- Σ = {Ana, Ion, învață, doarme, ., ' '},
- S este simbolul de start și anume Propoziție,
- $P = \{$
  - Propoziție $\rightarrow$ Subiect ' ' Predicat .,
  - Subiect $\rightarrow$ Ana | Ion,
  - Predicat $\rightarrow$ învață | doarme

  $\}$

Gramatici formale

Observații

- o gramatică definește numai **lexicul(vocabularul)** și **sintaxa** propozițiilor unui limbaj
- o gramatică formală **NU definește semantica** propozițiilor unui limbaj
- definiția unei gramatici definește:
  - lexicul(vocabularul) acestui meta-limbaj
  - sintaxa regulilor de producție
- definiția unei gramatici formale NU cuprinde semantica regulilor de producție
- aceasta va fi definită implicit prin definirea relației de derivare

Gramatici formale

- Fie M o mulțime. Orice mulțime de perechi ordonate $R = \{(a, b) |\ a, b \in M\}$, se numește relație binară peste M $(R \subseteq A \times A)$.
- Fie o gramatică $G = (V_N, \Sigma, S, P)$. **Derivarea într-un pas (derivarea directă)** este o relație binară între șiruri din V^* notată cu
  $\Rightarrow$ $(\Rightarrow \subseteq V^* \times V^*)$
  și definită astfel:
  $\alpha \Rightarrow \beta$ dacă și numai dacă $\alpha = xpy$ și $\beta = rqt$, $x, p, y, r, q, t \in V^*$ și $\exists\ p \rightarrow q \in P$.
- De exemplu:
  Subiect ' ' Predicat . $\Rightarrow$ Ana ' ' Predicat .

Gramatici formale

- Atunci când derivăm o formă propozițională, putem ajunge în situația în care aceasta să conțină mai multe simboluri neterminale. Prin urmare va trebui să alegem pentru care dintre simbolurile neterminale vom aplica următoarea regulă de producție pentru a face o derivare directă.
- Dacă se alege întotdeauna neterminalul cel mai din stânga, atunci derivarea se va numi **derivare stânga**.
  De exemplu:
  Subiect ' ' Predicat . $\Rightarrow$ Ana ' ' Predicat .
- Dacă se alege întotdeauna neterminalul cel mai din dreapta, atunci derivarea se va numi **derivare dreapta**.
  De exemplu:
  Subiect ' ' Predicat . $\Rightarrow$ Subiect ' ' doarme .

Gramatici formale

- Fie o gramatică $G = (V_N, \Sigma, S, P)$. **Derivarea în k pași** este o relație binară între șiruri din V^* notată cu $\Rightarrow^k$ și definită astfel:
  $\gamma \Rightarrow^k \delta$ dacă și numai dacă $\exists\ \alpha_1, \alpha_2, \ldots \alpha_{k+1}$ astfel încât $\alpha_1 = \gamma, \alpha_{k+1} = \delta$ și $\alpha_i \Rightarrow \alpha_{i+1}, \forall\ i \in [1, k]$.
- De exemplu:
  Subiect ' ' Predicat . $\Rightarrow^2$ Ana ' ' doarme .

Gramatici formale

- **Închiderea tranzitivă a relației de derivare** se notează $\Rightarrow^{+}$ și se definește folosind relația de derivare în k pași, astfel: $\gamma \Rightarrow^{+} \delta$ dacă și numai dacă $\exists\ k \geq 1$ astfel încât $\gamma \Rightarrow^{k} \delta$.
- **Închiderea tranzitivă și reflexivă a relației de derivare** se notează $\Rightarrow^{*}$ și se definește folosind închiderea trazitivă a relației de derivare, astfel: $\gamma \Rightarrow^{*} \delta$ dacă și numai dacă $\gamma = \delta$ sau $\gamma \Rightarrow^{+} \delta$.

Gramatici formale

- O **formă propozițională** peste o gramatica $G = (V_N, \Sigma, S, P)$ se definește recursiv (inductiv) în modul următor:
  1. **S** este o formă propozițională.
  2. Dacă **aBt** este o formă propozițională și dacă există o regulă de producție $B \to r \in P$, atunci **art** este o formă propozițională.
- De exemplu:
  Propoziție; Subiect Predicat.; Ana Predicat.; Subiect învață.; Ana învață.

Gramatici formale

- O **formă propozițională** peste o gramatica $G = (V_N, \Sigma, S, P)$ este orice șir $\alpha \in V^*$ cu proprietatea că $S \Rightarrow^* \alpha$.
- Forma propozițională este deci orice succesiune de simboluri terminale și/sau neterminale care poate fi obținută pornind de la simbolul de start prin 0 sau mai multe derivări.

Gramatici formale

- O forma propozițională peste o gramatică G, care conține numai simboluri terminale, se numește **propoziție** generata de G.
- De exemplu:
  Ana învață.; Ion doarme.

Gramatici formale

- O **propoziție** peste o gramatica $G = (V_N, \Sigma, S, P)$ este orice șir $x \in \Sigma^*$ cu proprietatea că $S \Rightarrow^+ x$.
- Propoziția este deci orice succesiune de simboluri terminale care poate fi obținută pornind de la simbolul de start prin cel puțin o derivare.

Gramatici formale

- **Limbajul generat de o gramatica** $G = (V_N, \Sigma, S, P)$ se notează cu L(G) și reprezintă mulțimea tuturor propozițiilor care pot fi generate de către această gramatică: $L(G) = \{x \in \Sigma^* | S \Rightarrow^+ x\}$.
- De exemplu: $L(G) = \{$Ana învață., Ion învață., Ana doarme., Ion doarme.$\}$
- O gramatica este o reprezentare finită (toate elementele sale sunt finite) a unui limbaj care poate să fie infinit.
- Nu orice limbaj are o reprezentare finită, cu alte cuvinte nu pentru orice limbaj există o gramatică care să îl reprezinte.
- Dacă este posibilă o astfel de construcție, atunci pentru un același limbaj dat, se pot construi mai multe gramatici distincte care să îl genereze.

Gramatici formale

- Singura modalitate prin care se pot genera propoziții cu un număr oarecare de simboluri este recursivitatea
- recursivitate **stanga**
  $A \Rightarrow^* A\alpha$
- recursivitate **dreapta**
  $A \Rightarrow^* \alpha A$
- recursivitate **directă**
  $A \rightarrow A\alpha$ și $A \rightarrow \alpha A$
- recursivitate **indirectă**
  $A \rightarrow \gamma B\beta, B \Rightarrow^* A\alpha$ și analog pentru recursivitatea dreapta

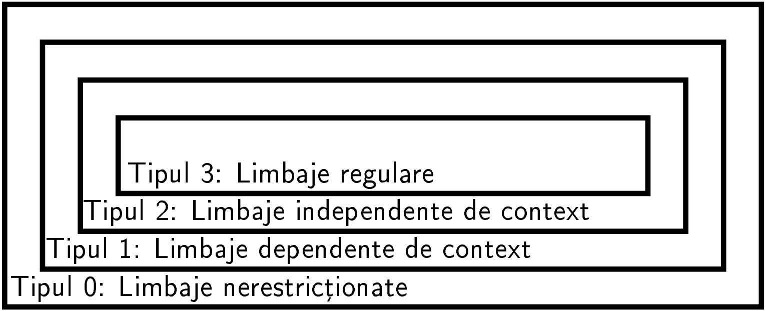
Gramatici formale

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- $V_N = \{$Subiect, Predicat, Propoziție, Frază$\}$,
- $\Sigma = \{$Ana, Ion, învață, doarme, ., ' '$\}$,
- S este simbolul de start și anume Frază,
- $P = \{$
  - **Frază** $\rightarrow$ Propoziție **Frază** $|$ Propoziție
  - Propoziție $\rightarrow$ Subiect ' ' Predicat .,
  - Subiect $\rightarrow$ Ana $|$ Ion,
  - Predicat $\rightarrow$ învață $|$ doarme

  $\}$

Ierarhia Chomsky



Ierarhia Chomsky

- este o ierarhie de clase de limbaje formale
- arată ce relație de incluziune există între clasele de limbaje formale
- fost descrisă pentru prima dată de către lingvistul Naom Chomsky, într-un articol publicat în anul 1956
- apartenența unui limbaj la unul dintre tipurile ierarhiei se determină în funcție de forma pe care o au regulile de producție ale unei gramatici care generează limbajul respectiv
- din această cauză, ierarhia Chomsky poate fi văzută și ca o ierarhie de clase de gramatici formale

Ierarhia Chomsky

Gramaticile de tipul 0 (gramatici nerestricționate)

- nu este impusă nicio restricție asupra regulilor de producție ale unei astfel de gramatici
- un limbaj este limbaj de tipul 0, dacă și numai dacă el este generat de către o gramatică de tipul 0

Ierarhia Chomsky

Gramaticile de tipul 1 (gramatici dependente de context)

- O gramatică $G = (V_N, \Sigma, S, P)$ se numește **gramatică dependentă de context** dacă și numai dacă **toate** regulile sale de producție au forma $\alpha A \beta \to \alpha \gamma \beta$, unde $A \in V_N$, $\alpha, \beta \in V^*$ și $\gamma \in V^+$.
- unii autori permit și regulile de producție de forma $S \to \epsilon$, însă cu condiția ca S să nu apară în partea dreaptă a niciunei reguli de producție
- un limbaj este dependent de context, dacă și numai dacă el este generat de către o gramatică dependentă de context

Ierarhia Chomsky

Gramaticile de tipul 1 (gramatici dependente de context)

Exemplu: $G = (V_N, \Sigma, S, P)$, unde:

- $V_N = \{$ Propoziție, Subiect, Atribut, Predicat, Complement, Substantiv, Verb, Adjectiv, Adverb$\}$,
- $\Sigma = \{$ a merge, a cânta, este, face, copilul, câinele, frumos, rău, bine, mult, ., ' '$\}$,
- S este Propoziție,
- $P = \{$
    - Propoziție $\rightarrow$ Subiect [Atribut] Predicat [Complement].,
    - Subiect $\rightarrow$ Substantiv | Verb,
    - Atribut $\rightarrow$ Adjectiv,
    - Predicat $\rightarrow$ Verb,
    - Complement $\rightarrow$ Adverb,
    - **Verb Atribut** $\rightarrow$ a merge Atribut | a cânta Atribut,
    - **Verb Complement** $\rightarrow$ este Complement | face Complement,
    - Substantiv $\rightarrow$ copilul | câinele,
    - Adjectiv $\rightarrow$ frumos | rău,
    - Adverb $\rightarrow$ bine | mult

  $\}$

Ierarhia Chomsky

Gramaticile de tipul 2 (gramatici independente de context)

- O gramatică $G = (V_N, \Sigma, S, P)$ se numește **gramatică independentă de context** dacă și numai dacă **toate** regulile sale de producție au forma $A \rightarrow u$, unde $A \in V_N$ și $u \in V^*$.
- un limbaj este independent de context, dacă și numai dacă el este generat de către o gramatică independentă de context
- cu exceptia gramaticii de pe slide-ul anterior, toate celelalte gramatice utilizate în prezentare sunt gramatici independente de context

Ierarhia Chomsky

Gramaticile de tipul 3 (gramatici regulare)

- O gramatică $G = (V_N, \Sigma, S, P)$ se numește **gramatică liniară dreapta** dacă este o gramatică independentă de context în care fiecare regulă de producție este de forma $A \to a$ sau de forma $A \to aB$ sau de forma $A \to \epsilon$, unde $a \in \Sigma$, iar $A, B \in V_N$.
- O gramatică $G = (V_N, \Sigma, S, P)$ se numește **gramatică liniară stânga** dacă este o gramatică independentă de context în care fiecare regulă de producție este de forma $A \to a$ sau de forma $A \to Ba$ sau de forma $A \to \epsilon$, unde $u \in \Sigma$, iar $A, B \in V_N$.
- O gramatică se numește **gramatică regulară** dacă este o gramatică liniară dreapta sau stânga.
- un limbaj este regular dacă și numai dacă el este generat de către o gramatică regulară

Ierarhia Chomsky

- ierarhia Chomsky arată relația de incluziune dintre cele patru clase de limbaje definite - prin urmare:
  - orice limbaj regular este și un limbaj independent de context
  - orice limbaj independent de context este și un limbaj dependent de context
  - și orice limbaj dependent de context este și un limbaj nerestricționat
- analog, se poate afirma că:
  - nu orice limbaj nerestricționat este și un limbaj dependent de context
  - nu orice limbaj dependent de context este și limbaj independent de context
  - nu orice limbaj independent de context este și limbaj regular

Ierarhia Chomsky

- având în vedere restricțiile impuse asupra regulilor de producție ale fiecărui tip de gramatici formale, se observă că limbajele regulare sunt cele mai simple, iar limbajele nerestricționate sunt cele mai complexe
- cu alte cuvinte, complexitatea limbajelor scade de la tipul 0 către tipul 3
- Exemple:
  - limbajele naturale sunt limbaje de tipul 0
  - limbajele de programare sunt limbaje de tipul 2
  - unele aspecte ale limbajelor de programare le-ar clasifica ca și limbaje de tipul 1, însă în practică se preferă definirea lor prin gramatici de tipul 1, adăugându-se câteva restricții suplimentare
  - limbajul atomilor lexicali ai unui limbaj de programare (cuvintele cheie, numele de funcții, variabile, structuri, clase, ș.a., constantele numerice, constantele literale, constantele de tip șir de caractere, operatorii, semnele de punctuație) este un limbaj de tipul 3

Ierarhia Chomsky

- gramaticile formale permit definirea limbajelor prin generare
  - pornind de la simbolul de start, prin derivări succesive, putem genera orice propoziție a limbajului
- cum se poate rezolva problema inversă?
  - fiind dată o propoziție oarecare, cum se poate stabili dacă ea aparține unui anumit limbaj (definit de o gramatică dată)?
  - pentru aceasta se folosește un "dispozitiv" capabil sa recunoască dacă o propoziție dată aparține sau nu limbajului pe care el îl definește

Ierarhia Chomsky

Tipuri de "dispozitive" necesare stabilirii faptului că o propoziție dată aparține sau nu unui limbaj dat, în funcție de complexitatea acestuia

Gramatici de tipul 3	**automate finite**
Gramatici de tipul 2	**automate finite cu memorie (stivă)**
Gramatici de tipul 1	mașină Turing cu bandă limitată
Gramatici de tipul 0	mașină Turing

Arbori de analiză gramaticală

- Definiții: graf orientat, arbore, arbore sintactic
- Arbori sintactici descendenți
- Arbori sintactici ascendenți
- Analiza gramaticală

Arbori de analiză gramaticală

- Fie M o mulțime de noduri și R o relație binară peste M. Atunci $G = (M, R)$ se numește **graf orientat**.
- Dacă $(a, b) \in R$, atunci definim următoarele mulțimi:
  - $input(a) = \{b | (b, a) \in R\}$,
  - $output(a) = \{b | (a, b) \in R\}$.
- **Un arbore** este un graf orientat (M, R) cu următoarele restricții:
  - $\exists\ r \in M$ unic, astfel încât $input(r) = \emptyset$,
  - $\forall\ a \in M,\ a \neq r,\ card(input(a)) = 1$,
  - $\forall\ a \in M,\ (r, a) \in R^*$.

Arbori de analiză gramaticală

Fie o gramatică formală $G = (V_N, \Sigma, S, P)$. Un **arbore de derivare** (arbore sintactic) $A = (M, R)$ este un arbore etichetat ordonat de la stânga la dreapta cu proprietățile:

1. rădăcina arborelui este etichetată cu S,
2. pentru orice nod $a \in M$ cu descendenți, astfel încât A este eticheta nodului a, iar $X_1, X_2, \ldots, X_i$ sunt etichetele descendenților lui a, $\exists\ A \rightarrow X_1 X_2 \ldots X_i \in P$.

Arbori de analiză gramaticală

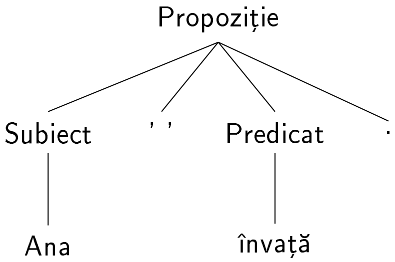
- arborii sintactici sunt o reprezentare arborescentă a succesiunii de derivări prin care se generează o anumită propoziție
- fiecare nod al arborelui care nu este frunză, este etichetat cu un neterminal
- fiecare frunză a arborelui este etichetată cu un terminal
- citind etichetele nodurilor arborelui (de la stânga la dreapta) de pe un nivel dat se obține o formă propozițională

Arbori de analiză gramaticală

- construcția arborelui sintactic pentru o propoziție dată se poate face în două moduri:
  - se poate porni de la rădăcină către frunze, adică de la simbolul de start către propoziție
    (construcție descendentă - **arbore sintactic descendent**)
  - se poate porni de la frunze către rădăcină, adică de la propoziție către simbolul de start
    (construcție ascendentă - **arbore sintactic ascendent**)
- evident, arborele sintactic descendent al unei propoziții este identic cu arborele sintactic ascendent al aceleiași propoziții

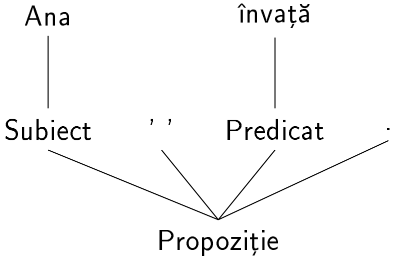
Arbori de analiză gramaticală

- arborele sintactic **descendent** corespunzător propoziției *Ana învață.* și gramaticii prezentate anterior:



Arbori de analiză gramaticală

- arborele sintactic **ascendent** corespunzător propoziției *Ana învață.* și gramaticii prezentate anterior:



Arbori de analiză gramaticală

- verificarea dacă o propoziție dată aparține limbajului definit de o gramatică formală se numește **analiză sintactică**
- ea constă în încercarea de a construi arborele sintactic al propoziției date, conform gramaticii care definește limbajul respectiv
  - dacă se poate construi un arbore sintactic - propoziția aparține limbajului
  - dacă nu se poate construi un arbore - propoziția nu aparține limbajului
- clasificare
  - **analiză sintactică descendentă** - încearcă construirea arborelui sintactic în sens descendent
  - **analiză sintactică ascendentă** - încearcă construirea arborelui în sens ascendent

Intrebari recapitulative

- Fie $G = (V_N, \Sigma, S, P)$, astfel încât $card(\Sigma) = m$, iar $card(V_N) = n$. Care ar trebui să fie valoarea minimă pentru $card(P)$ astfel încât definiția gramaticii să fie corectă? Explicați.
- Având în vedere mulțimile V_0, V_1, ..., V_i folosite în definiția mulțimii V*, să se scrie cine este $V_{i+1} \setminus V_i$ și să se explice.
- Care este restricția impusă numărului de elemente ale mulțimii terminalelor din definiția unei gramatici, așa cum a fost introdusă la curs?

Intrebari recapitulative

- Fie $G = (V_N, \Sigma, S, P)$, astfel încât $card(\Sigma) = m$, $card(VN) = n$, iar $card(P) = k$ și fie p un element din mulțimea $L(G)$. Câte frunze va avea arborele sintactic corespunzător lui p?

- Prezentați care sunt asemănările dintre o formă propozițională și o propoziție.

- Prezentați felul în care o gramatică G definește semantica unui limbaj.

Intrebari recapitulative

- Care este relația dintre alfabetul unei gramatici și alfabetul limbajului definit de către gramatica respectivă?
- Explicați care este diferența dintre relația de derivare directă și relația de derivare în k pași.
- Enumerați situațiile în care relația de derivare în k pași este o relație între forme propoziționale.

Intrebari recapitulative

- Explicați cauza diferențelor dintre derivarea stângă și derivarea dreapta pentru o propoziție care aparține unui limbaj regular, definit de o gramatică regulară.

- Pornind de la definiția formelor propoziționale, explicați de ce simbolul de start al unei gramatici este o formă propozițională.

Intrebari recapitulative

- Explicați semantica ierarhiei Chomsky.

- Câți arbori de derivare pot fi construiți pentru o propoziție dată, pe baza unei gramatici care definește limbajul căreia îi aparține propoziția? Explicați.

Intrebari recapitulative

- Fie următoarele două succesiuni de relații de derivare realizate pe baza unei gramatici G date, pentru o propoziție $p \in L(G)$:
  1. succesiunea de relații de derivare bazate pe derivare dreapta,
  2. succesiunea de relații de derivare bazate pe derivarea stânga.

  Scrieți care este numărul minim de forme propoziționale comune celor două succesiuni de relații de derivare și explicați.

- Fie G o gramatică. Se notează cu $F(G)$ mulțimea formelor propoziționale care se pot defini pe baza gramaticii G. Scrieți ce relație există între $F(G)$ și $L(G)$ și explicați.

Cuprins

- Automate cu stări finite
- Expresii regulare

Automate cu stări finite

Cuprins

- Automate finite deterministe
- Automate finite nedeterministe
- Automate finite ϵ
- Automate finite vs. Gramatici formale

Automate cu stări finite

- reprezintă un model matematic care permite descrierea proceselor de calcul
- un automat finit este gândit ca și o mașină abstractă, care, la un moment dat, se poate afla într-o singură stare, dintr-o mulțime finită de stări date
- starea în care se află automatul la un moment dat se numește **stare curentă**
- starea în care se află automatul înainte de începerea procesului de calcul se numește **stare inițială**
- automatul poate să treacă dintr-o stare în alta în funcție de anumite intrări
- schimbarea stării curente a automatului (trecerea din starea curentă într-o altă stare, care devine noua stare curentă) se numește **tranziție**

Clasificare

- dacă pentru o stare curentă și pentru o intrare dată automatul poate să facă o singură tranziție (există o singură stare în care poate să treacă, în baza intrării date), atunci el este un **automat finit determinist**
- dacă pentru o stare curentă și o intrare dată un automat poate efectua mai multe tranziții (există mai multe stări în care poate să treacă, în baza intrării date), atunci el este un **automat finit nedeterminist**

Automate cu stări finite

- în teoria limbajelor formale, automatele finite sunt utilizate pentru a defini sau pentru a recunoaște **limbaje regulare**
- cu alte cuvinte, expresivitatea unui anumit finit este aceeași cu cea a unei gramatici regulare
- intrarea pe baza căreia se realizează tranzițiile este dată de către următorul simbol neanalizat din șirul de la intrare
- mulțimea șirurilor de simboluri pe care le recunoaște un automat finit dat, se numește limbajul definit de către automat
- automatele finite sunt utilizate pentru a verifica dacă o propoziție dată aparține sau nu unui anumit limbaj regular, adică dacă este o propoziție corectă din punctul de vedere al limbajului regular avut în vedere

Automate finite deterministe

Un **automat finit determinist** este un cvintuplu $AFD = (\Sigma, Q, F, q_0, f)$, în care:

- Σ este o mulțime finită de simboluri și se numește alfabetul limbajului de intrare al automatului;
- Q este mulțimea finită de stări ale automatului;
- F este mulțimea stărilor finale ale automatului, $F \subseteq Q$;
- q_0 este starea inițială a automatului, $q_0 \in Q$;
- f este funcția de tranziție, $f : Q \times \Sigma \to Q$.

Automate finite deterministe

Reprezentare

- prin graful de tranziție
  - **stările** se reprezintă prin cercuri, care au înscrise în ele numele
  - **starea inițială** este marcată printr-o săgeată
  - **stările finale** sunt marcate prin dublarea cercurilor
  - **tranzițiile** sunt reprezentate prin săgeți între stări, deasupra cărora se scrie simbolul/simbolurile de la intrare în baza cărora se face schimbarea stării
- prin tabela de tranziție
  - stările finale se marchează printr-o steluță

Automate finite deterministe

Exemplu

- Fie automatul finit determinist $AFD = (\Sigma, Q, F, q_0, f)$, unde:
  - $\Sigma = \{0, 1\}$,
  - $Q = \{q_0, q_1, q_2\}$,
  - $F = \{q_0, q_2\}$
  - starea inițială este q_0,
  - iar funcția de tranziție f se deduce din graful de tranziție, sau respectiv din tabelul de tranziție.
- care acceptă limbajul următor:
  $L(AFD) = \{0^n 1^m 0 | n \geq 0,\ m \geq 1\}$,
  unde am notat cu a^t concatenarea simbolului a cu el însuși de un număr de t ori

Automate finite deterministe

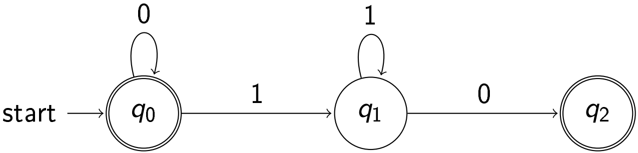


Figure: Graful de tranziție

f	0	1
$*q_0$	q_0	q_1
q_1	q_2	q_1
$*q_2$	-	-

Figure: Tabela de tranziție

Automate finite deterministe

- un automat citește un șir finit de simboluri $a_1 a_2 \ldots a_n,\ a_i \in \Sigma, \forall\, 1 \leq i \leq n$
- în funcție de fiecare simbolul citit și de starea curentă în care se află automatul la citirea simbolului respectiv, acesta execută o tranziție, în conformitate cu definiția funcției de tranziție
- automatul se oprește
  - după ce a citit toate simbolurile din propoziția de intrare
  - sau atunci când pentru simbolul următor de la intrare și pentru starea curentă în care se află, nu se poate executa nicio tranziție

Automate finite deterministe

- Fie $AFD = (\Sigma, Q, F, q_0, f)$ un automat finit. O pereche $(x,\ q), x \in \Sigma^*,\ q \in Q$
  se numește **configurație** a automatului finit.
- q este starea curentă a automatului,
- iar x este restul șirului de la intrarea automatului, care nu a fost încă analizat

Automate finite deterministe

- Fie $AFD = (\Sigma, Q, F, q_0, f)$ un automat finit. **Relația de mișcare** este o relație binară notată cu
  $\vdash$ $(\vdash \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q))$
  și definită astfel:
  $(ax, q) \vdash (x, q') \Leftrightarrow f(q, a) = q'$,
  unde $a \in \Sigma$, $x \in \Sigma^*$, $q, q' \in Q$.
- De exemplu:
  $(00, q_0) \vdash (0, q_0)$

Automate finite deterministe

- Fie $AFD = (\Sigma, Q, F, q_0, f)$ un automat finit. **Relația de mișcare în k pași** este o relație binară notată cu
  $\vdash^k$ $(\vdash^k \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q))$
  și definită astfel:
  $(a_1 a_2 \ldots a_k x, q) \vdash^k (x, q') \Leftrightarrow \exists\ k+1$ configurații astfel încât
  $(a_i a_{i+1} \ldots a_k x, q_i) \vdash (a_{i+1} \ldots a_k x, q_{i+1}),\ \forall\ 1 \leq i \leq k,$
  $q = q_1, q' = q_{k+1},$
  unde $a_1, a_2, \ldots, a_k \in \Sigma,\ x \in \Sigma^*,\ q, q', q_1, q_2, \ldots, q_{k+1} \in Q.$
- De exemplu:
  $(0001, q_0) \vdash^3 (1, q_0)$

Automate finite deterministe

- Fie $AFD = (\Sigma, Q, F, q_0, f)$ un automat finit. **Relația generală de mișcare** este o relație binară notată cu $\vdash^*$ $(\vdash^* \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q))$
și definită astfel:
$(x_1, q) \vdash^* (x_2, q') \Leftrightarrow (x_1 = x_2$ și $q = q')$ sau
$((x_1, q) \vdash^+ (x_2, q'))$,
unde $x_1, x_2 \in \Sigma^*, q, q' \in Q$.

Automate finite deterministe

- Fie $AFD = (\Sigma, Q, F, q_0, f)$ un automat finit. Se spune că $x \in \Sigma^*$ este o **propoziție acceptată** de către automat, dacă există următoarea relație de mișcare:
  $(x, q_0) \vdash^* (\epsilon, q')$, unde $q' \in F$.
- o propoziție acceptată de către un automat finit AFD este orice șir de simboluri din Σ, care, citit simbol cu simbol de la stânga spre dreapta, duce automatul din starea inițială într-o stare finală

Automate finite deterministe

- Fie $AFD = (\Sigma, Q, F, q_0, f)$ un automat finit. **Limbajul acceptat** de către automatul AFD se notează cu $L(AFD)$ și est definit în felul următor:
  $L(AFD) = \{x \in \Sigma^* | f^*(q_0, x) \in F\}$,
  unde f este extinsă la f^* astfel:
  - $f^*(q, \epsilon) = q$,
  - $f^*(q, ax) = f^*(f(q, a), x)$.
- cu alte cuvinte, limbajul acceptat de către un automat finit AFD este mulțimea formată din toate șirurile de simboluri peste Σ care sunt acceptate de către automat

Automate finite deterministe

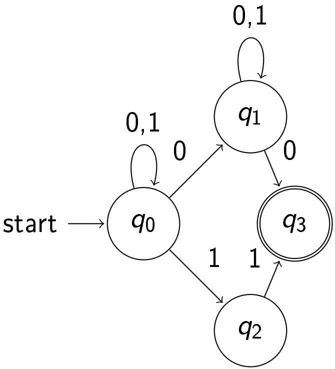
- automatele finite au un număr fix și limitat de stări
- există limbaje care nu pot fi recunoscute de către un astfel de automat finit - evident este vorba de limbajele care nu sunt regulare
- limbajul $L = \{a^n ba^n | n > 0\}$
  - un automat finit determinist ar trebui să se afle într-o stare diferită după citirea șirului a^k, pentru fiecare valoare a lui $k > 0$ (deoarece numai după k citiri de a, automatul trebuie să accepte șirul ba^k)
  - rezultă necesitatea unui număr infinit de stări, ceea nu este posibil

Automate finite nedeterministe

- Un **automat finit nedeterminist** este un cvintet $AFN = (\Sigma, Q, F, q_0, f)$, unde Σ, Q, F, q_0 sunt definite în același mod ca și pentru un automat finit determinist, iar funcția de tranziție f este definită astfel:
  $f : Q \times \Sigma \to 2^Q - \emptyset$,
  unde prin 2^Q s-a notat mulțimea tuturor submulțimilor mulțimii Q.
- pentru o aceeași stare și pentru un același simbol de la intrare, pot exista mai multe tranziții posibile, ceea ce înseamnă că automatul poate trece în mai multe stări
- evident, automatul nu are niciun criteriu suplimentar în baza căruia să facă alegerea.

Automate finite nedeterministe

- Fie automatul finit $M = (\Sigma, Q, F, q_0, f)$, unde:
  - $\Sigma = \{0, 1\}$
  - $Q = \{q_0, q_1, q_2, q_3\}$
  - $F = \{q_3\}$
  - iar funcția de tranziție f este definită prin următorul graf de tranziție:



Automate finite nedeterministe

- Pentru orice automat finit nedeterminist $M = (\Sigma, Q, F, q_0, f)$, există un automat finit determinist M' echivalent, adică $L(M') = L(M)$.
- Fie $M = (\Sigma, Q, F, q_0, f)$ un automat finit nedeterminist. Atunci automatul finit determinist echivalent M' este definit astfel:
  $M' = (\Sigma, Q', F', q_0', f')$,
  unde:

Automate finite nedeterministe

	AFN	AFD
Alfabetul	Σ	Σ
M. stărilor	$Q = \{q_0, q_1, q_2, \dots q_n\}$	$Q' = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \dots, \{q_n\}, \{q_0, q_1\}, \{q_0, q_2\}, \dots, \{q_0, q_n\}, \dots, \{q_{n-1}, q_n\}, \dots, \{q_0, q_1, \dots q_n\}\}$
Starea inițială	q_0	$\{q_0\}$
M. stărilor finale	$F \subset Q$	$F' = \{s \in Q' \mid s$ *conține cel puțin o stare finală a AFN* $\}$
Tranziții	f	$f'(\{q_{i_1}, q_{i_2}, \dots, q_{i_k}\}, a) = f(q_{i_1}, a) \cup f(q_{i_2}, a) \cup \dots \cup f(q_{i_k}, a)$

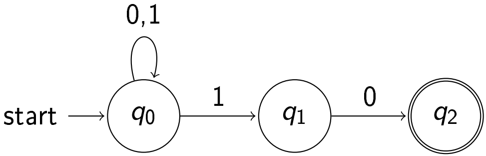
Automate finite nedeterministe

După generarea elementelor automatului finit determinist pornind de la cele ale automatului finit nedeterminist, se elimina stările și tranzițiile inutile:

- stările inutile sunt stările la care automatul nu poate ajunge pornind din starea inițială și făcând una sau mai multe tranziții
- tranzițiile inutile sunt tranzițiile care pleacă dintr-o stare inutilă

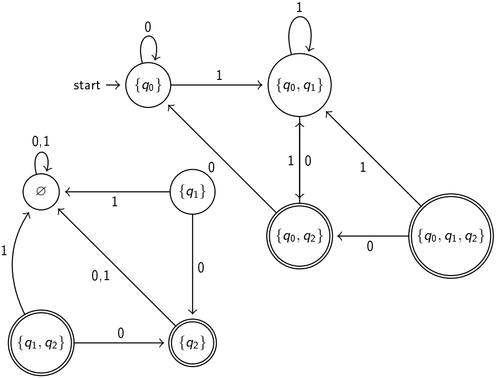
Automate finite nedeterministe - Exercițiu

AFN:

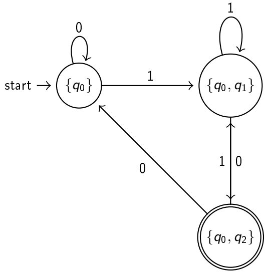


Automate finite nedeterministe - Exercițiu

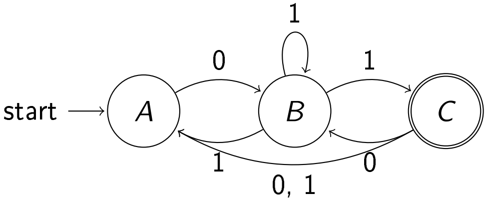
AFD echivalent (complet - inclusiv stările și tranzițiile inutile):



AFD echivalent (după eliminarea stărilor și tranzițiilor inutile):

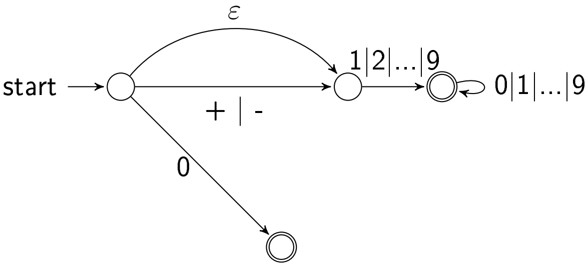


Automate finite nedeterministe - Exercițiu



Automate finite ϵ

- O tranziție ε este o tranziție dintr-o stare în alta, fără a "consuma" niciun simbol din șirul de intrare.
- Aceste tranziții sunt practic tranziții spontane, care se fac fără a privi la simbolul care urmează în șirul de intrare.



Automate finite ϵ

- Orice AFN este de fapt un AF-ϵ care nu are nicio tranziție ϵ.
- AFN și AF- ϵ sunt echivalente.
- Deducerea AFN echivalent unui AF- ϵ constă în construirea unui AFN care accepta același limbaj ca și AF- ϵ de la care am pornit.
- Construirea se face combinând fiecare tranziție ϵ cu următoarea tranziție care se face pe baza simbolului de la intrare.

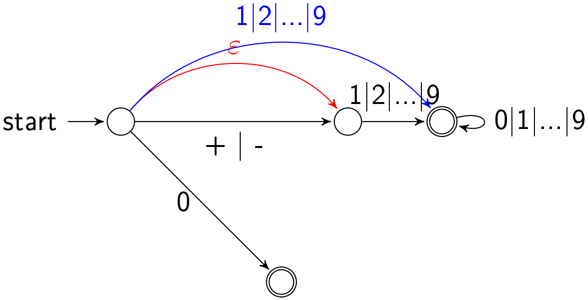
Automate finite ϵ

- Pentru orice automat finit ϵ $M = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$, există un automat finit nedeterminist M' echivalent, adică $L(M') = L(M)$.
- Fie $M = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$ un automat finit ϵ. Atunci automatul finit nedeterminist echivalent M' este definit astfel $M' = (\Sigma, Q', F', q_0, f')$, unde:

	AFϵ	AFN
Alfabetul	$\Sigma \cup \{\epsilon\}$	Σ
M. stărilor	$Q = \{q_0, q_1, q_2, \ldots q_n\}$	$Q' = \{q_0, q_1, q_2, \ldots q_n\}$
Starea inițială	q_0	q_0
M. stărilor finale	$F \subset Q$	$F' = F \cup \{q' \in Q' \mid \exists\, s \in CL(q')\|_{AF\epsilon}$ a.i. $s \in F\}$
Tranziții	f	$f'(q,a) = f(q,a)$ $f'(q,b) = f(f(\ldots f(q,\epsilon)\ldots,\epsilon),b)$ $\forall\, q \in Q'$ i $a, b \in \Sigma$

Automate finite ϵ

- Tranziția ϵ duce din starea inițiala în starea a doua.
- Din starea a doua se trece în starea a treia pe baza $1 \mid 2 \mid \cdots \mid 9$.
- Înlocuim tranziția ϵ (cea în roșu) cu o tranziție din starea inițială direct în starea a treia pe baza $1 \mid 2 \mid \cdots \mid 9$ (cea în albastru).



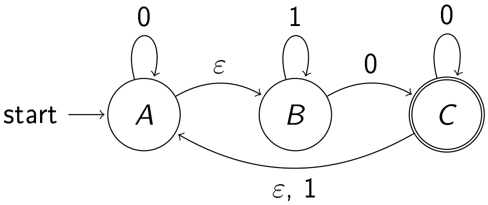
Automate finite ϵ

- După eliminarea tranzițiilor ϵ, toate stările ale căror închideri conțin o stare finală, devin stări finale.
- Dacă starea în care se trece prin tranziția ϵ nu are nicio tranziție pe baza intrării, atunci:
  - Dacă starea respectivă este stare finală, starea din care pleacă tranziția ϵ devine stare finală, iar tranziția ϵ este eliminată.
  - Altfel, se elimină pur și simplu tranziția ϵ (fiind oricum o tranziție inutilă).

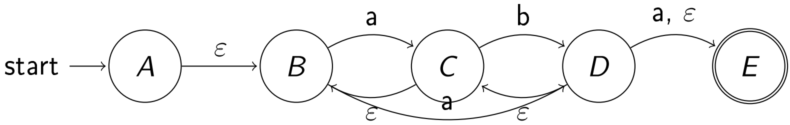
Automate finite ϵ

- Fie $AF - \epsilon = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$ un automat finit ϵ. **Închiderea unei stări (closure)** $q \in Q$, notată cu $CL(q)$, este mulțimea tuturor stărilor la care se poate ajunge pornind din starea q și făcând numai tranziții ϵ:
  $CL(q) = \{q' \in Q \mid (x, q) \vdash^{+} (x, q')\}$.
- Pentru a specifica faptul că închiderea unei stări q se calculează pentru un anumit automat finit AF, vom folosi notația $CL(q)|_{AF}$.

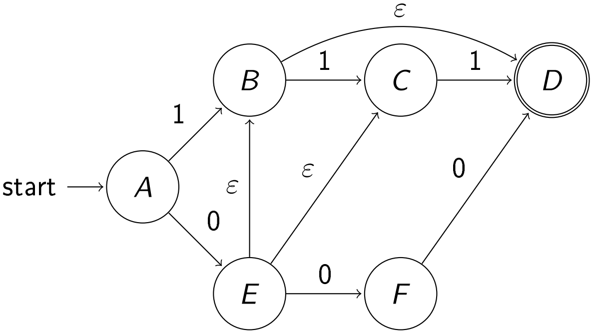
Automate finite ϵ - Exercițiu



Automate finite ϵ - Exercițiu



Automate finite ϵ - Exercițiu

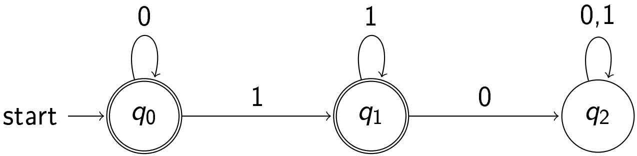


Automate finite vs. Gramatici formale

- automatele finite, indiferent de tip, accepta numai limbaje regulare
- **automatele finite, indiferent de tip, au aceeași expresivitate ca și gramaticile regulare (sunt echivalente)**
- este foarte simplu de observat acest lucru, încercând să construim gramatica formală care definește același limbaj ca și un AF dat
- se va observa că toate regulile de producție ale gramaticii respective vor avea, cel puțin în forma inițială, exact formatul cerut de definiția gramaticilor regulare

Automate finite vs. Gramatici formale - Exercițiu

De exemplu, pentru AFD de mai jos, mulțimea regulilor de producție a gramaticii care definește același limbaj ar putea avea elementele:



- $S \rightarrow 0S \mid 1P \mid \epsilon$
- $P \rightarrow 1P \mid 0Q \mid \epsilon$
- $Q \rightarrow 0Q \mid 1Q$

Automate finite vs. Gramatici formale

Aplicație facultativă:

- Implementarea într-un limbaj de programare oarecare a unei aplicații care să deducă gramatica formală echivalentă pentru un AFD dat.
- Intrare: Pentru AFD se vor da, prin intermediul unui fișier de intrare, alfabetul, mulțimea stărilor, starea inițială, mulțimea stărilor finale, tabelul de definiție a funcției de tranziție
- Ieșire: Elementele de definiție ale gramaticii formale echivalente.
- Observații:
  - Alfabetul AFD va fi mulțimea terminalelor pentru G.
  - Mulțimea stărilor AFD va reprezenta mulțimea neterminalelor pentru G.
  - Starea inițială va reprezenta simbolul de start pentru G.
  - Regulile de producție se vor construi pe baza tranzițiilor AFD.

Intrebari recapitulative

- Explicați la ce este folosită închiderea unei stări (closure) în cadrul algoritmului de eliminare a tranzițiilor epsilon ale unui automat finit epsilon.
- Explicați cum se folosește relația de mișcare pentru a defini limbajul acceptat de către un automat finit determinist.
- Explicați de ce este nevoie ca pentru automatele finite nedeterministe să se construiască automate finite deterministe echivalente.

Intrebari recapitulative

- Care este restricția impusă numărului de elemente ale mulțimii stărilor din definiția unui automat finit nedeterminist, așa cum a fost introdusă la curs?
- Prezentați situațiile în care un automat finit epsilon este un automat finit determinist.
- Scrieți care este ultima configurație a șirului de relații de mișcare care reprezintă analiza sintactică ascendentă în baza unei gramatici G, efectuată de către un automat finit cu stivă pentru o propoziție care aparține lui $L(G)$.

Intrebari recapitulative

- Fie automatul finit nedeterminist $A_1 = (\Sigma, Q, F, q_0, f)$ și fie $A2 = (\Sigma, Q', F', \{q_0\}, f')$ automatul finit determinist echivalent, obținut prin aplicarea algoritmului studiat. Scrieți ce relație există între Q și Q' și explicați.

Expresii regulare

Cuprins

- Expresii regulare
- Convenții de analiză pentru expresiile regulare
- De la expresii regulare la automate finite

Expresii regulare

- Constituie un limbaj care permite descrierea altor limbaje (descrierea tuturor atomilor lexicali care pot aparea într-un limbaj).
- O expresie regulara R definește recursiv un limbaj L(R), astfel:
  - Șirul vid ϵ este o expresie regulara și reprezinta șirul care nu conține niciun caracter.

    $$L(\epsilon) = \{""\}$$

  - Orice caracter singular c este o expresie regulara și reprezintă șirul de caractere, care conține un singur caracter, și anume pe c.

    $$L(c) = \{"c"\}$$

  - **Concatenarea:** Daca R_1 și R_2 sunt două expresii regulare, atunci $R1R2$ (citit R_1 concatenat cu R_2) este o expresie regulară și reprezintă limbajul:

    $$L(R_1R_2) = \{s_1s_2 | s_1 \in R1 \wedge s_2 \in R_2\}$$

Expresii regulare

- **Reuniunea:** Dacă R_1 și R_2 sunt două expresii regulare, atunci $R_1|R_2$ este o expresie regulară și reprezintă limbajul:

$$L(R_1|R_2) = L(R_1) \cup L(R_2).$$

- **Închiderea:** Dacă R este o expresie regulară, atunci R^* este o expresie regulară și reprezintă limbajul format din mulțimea de propoziții obținute prin concatenarea a zero sau mai multe șiruri, fiecare din $L(R)$.

$$L(R^*) = L(\epsilon) \ \cup \ L(R) \ \cup \ L(RR) \ \cup \ L(RRR) \ \cup \ldots$$

- **Închiderea nereflexivă:** Dacă R este o expresie regulară, atunci R^+ este o scriere prescurtată pentru RR^*.

$$L(R^+) = L(R) \cup L(RR) \cup L(RRR) \cup \ldots$$

- **Opțional:** Dacă R este o expresie regulară, atunci $R?$ este o scriere prescurtată pentru $R|\epsilon$.

Alte prescurtari permisei

- $[c_1 c_2 \ldots]$ este o prescurtare pentru $c_1|c_2|\ldots$.
- $a_1 - a_n$ folosita intr-o prescurtare de tipul anterior reprezinta toate caracterele intre a_1 si a_n.
- $[\hat{}c_1 c_2 \ldots]$ este o prescurtare pentru $d_1|d_2|\ldots$, unde$d_1, d_2, \ldots$ sunt toate acele caractere care nu apar printre $c_1, c_2, \ldots$.
- . este o prescurtare pentru toate șirurile de caractere care au un singur caracter, iar acesta nu este un terminator de linie.

Expresii regulare

- Operatorii inchidere și opțional au prioritate față de concatenare, care are prioritate față de reuniune.
- Pentru o scriere clară și fără ambiguități se pot utiliza parantezele rotunde.
- Exemplu: Expresia regulara

  $$ab?cd*ef|g$$

  este echivalenta cu

  $$(a(b?)c(d*)ef)|g$$

  și reprezintă limbajul format din propozițiile : acef, g, acefg, abcef, abcefg, acdddef, acdddefg, abcdddefg, abcdddddef, s.a.m.d.

Expresii regulare

- Potrivirea unui text
  - **Text**: Anna Jones and a friend went to lunch
  - **Regex**: went
  - **Matches**: Anna Jones and a friend **went** to lunch
- Potrivirea unui singur caracter oarecare (punct)
  - **Text**: abc def ant cow
  - **Regex**: a..
  - **Matches**: **abc** def **ant** cow
- Potrivirea unui singur caracter dintr-un cuvânt (backslash w mic)
  - **Text** abc anaconda ant cow apple
  - **Regex**: a\w\w
  - **Matches**: **abc** **ana**conda **ant** cow **app**le

Expresii regulare

- Potrivirea unui singur caracter care nu este într-un cuvânt (backslash W mare)
  - **Text**: abc anaconda ant cow apple
  - **Regex**: a\W
  - **Matches**: abc anacond**a** ant cow apple
- Potrivirea unui spațiu alb (backslash s mic)
  - **Text**: abc anaconda ant
  - **Regex**: a\w\w\s
  - **Matches**: **abc** anaconda ant

  Spațiu alb este definit ca fiind caracterul spațiu (\0x0020), new line (\n), form feed (\f), carriage return (\r), tab (\t) și vertical tab (\v).

Expresii regulare

- Potrivirea unei cifre
  - **Text:** 123 12 843 8472
  - **Regex:** \d\d\d
  - **Matches:** **123** 12 **843** **847**2
- Potrivirea unui singur caracter dintr-un anumit set
  - **Text:** abc def ant cow
  - **Regex:** [da]..
  - **Matches:** **abc** **def** **ant** cow
- Adăugarea lui ^ la începutul setului de caractere cere ca nici unul dintre caracterele din setul specificat să nu fie căutate
  - **Text:** abc def ant cow
  - **Regex:** [^da]..
  - **Matches:** abc def ant cow

Expresii regulare

- Potrivirea unui caracter dintr-o serie de caractere
  - **Text**: abc pen nda uml
  - **Regex**: .[a-d].
  - **Matches**: **abc** pen **nda** uml

  - **Text**: abc no 0aa i8i
  - **Regex**: [a-z0-9]\w \w
  - **Matches**: **abc** no **0aa i8i**
- Specificarea numărului de potriviri - de zero sau de mai multe ori
  - **Text**: Anna Jones and a friend owned an anaconda
  - **Regex**: a \w*
  - **Matches**: **Anna** Jones **and a** friend owned **an anaconda**
- Specificarea numărului de potriviri - o data sau de mai multe ori
  - **Text**: Anna Jones and a friend owned an anaconda
  - **Regex**: a \w+
  - **Matches**: **Anna** Jones **and** a friend owned **an anaconda**

Expresii regulare

- Specificarea numărului de potriviri - de zero ori sau o data
  - **Text:** Anna Jones and a friend owned an anaconda
  - **Regex:** $an?$
  - **Matches:** **An**n**a** Jones **and** **a** friend owned **an** **a**cond**a**
- Specificarea exactă a numărului de potriviri
  - **Text:** Anna Jones and Anne owned an anaconda
  - **Regex:** $an\{2\}$
  - **Matches:** **Ann**a Jones and **Ann**e owned an anaconda

  - **Text:** Anna and Anne lunched with an anaconda annnnnex
  - **Regex:** $an\{2,3\}$
  - **Matches:** **Ann**a and **Ann**e lunched with an anaconda **annn**nnex

Expresii regulare

- Potrivirea începutului unui șir
  - **Text:** an anaconda ate Anna Jones
  - **Regex:** ^a
  - **Matches:** **a**n anaconda ate Anna Jones
- Potrivirea sfârșitului unui șir
  - **Text:** an anaconda ate Anna Jones
  - **Regex:** \w+$
  - **Matches:** an anaconda ate Anna **Jones**

Convenții de analiză pentru expresiile regulare

- În unele cazuri, secvențe de caractere din codul sursă se pot potrivi la mai multe expresii regulare dintre cele folosite de către analizor.
- Exemplu: Fie expresiile regulare:

  (ab)*

  și

  [a-c]*

  Secvența de caractere "ab" se potrivește ambelor expresii.
- Deci **trebuie convenit dacă expresiile au prioritate în ordinea apariției sau în ordinea inversă a apariției.**

Convenții de analiză pentru expresiile regulare

- Pentru aceeași expresie regulară pot exista secvențe de caractere diferite din codul sursă, care să se potrivească la un moment dat.
- Exemplu: Fie expresia regulară

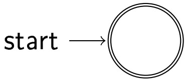
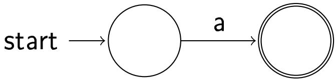
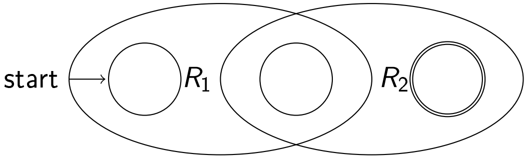
  $$ab|abcd$$

  și șirul de intrare "abcd": se va extrage atomul "ab" sau atomul "abcd"?
- Convenție in acest sens : **leftmost longest** - cea mai lungă secvență de caractere, citită de la stânga spre dreapta:
  - Pentru operatorii ?,*,+ se va lua secvența cea mai mare de caractere care se potrivește subexpresiei din care fac parte și care permite și restului codului sursă să fie analizat.
  - Dacă pentru o subexpresie $R_1|R_2$ se potrivește o aceeași secvență de caractere, atunci se va lua în considerare R_1.

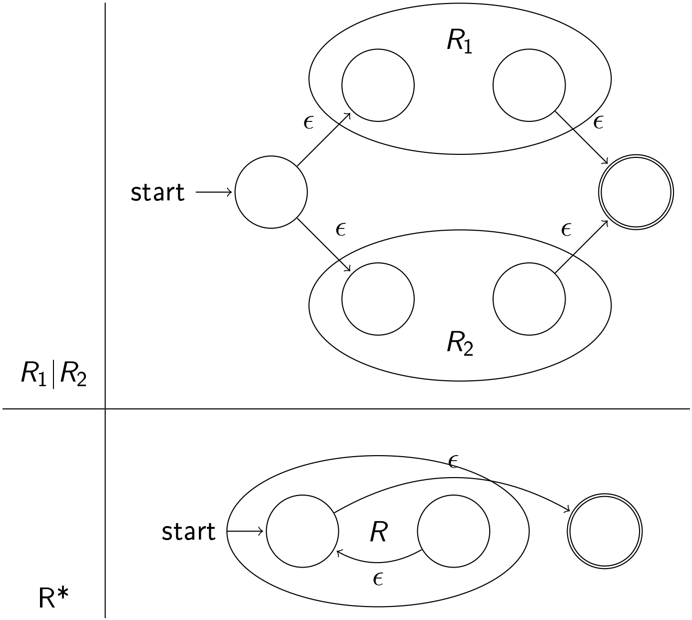
De la expresii regulare la automate finite

- **automatele finite și expresiile regulare au aceeași expresivitate (sunt echivalente)**
- orice expresie regulară poate fi convertită într-un automat finit
- construcția automatului finit se face recursiv, pe baza definiției expresiei regulare
- pentru fiecare dintre expresiile regulare de bază, va fi prezentat automatul finit echivalent, cu o singură stare finală, folosind conventiile:
  - o elipsă reprezintă un automat, care recunoaște expresia regulară R, sau aceeași mașină, dar cu toate stările componente transformate în stări nefinale
  - o elipsă este o colecție de stări și tranziții
  - cele două stări marcate reprezintă starea inițială și respectiv starea finală, sau starea inițială și starea finală, transformată într-o stare nefinală

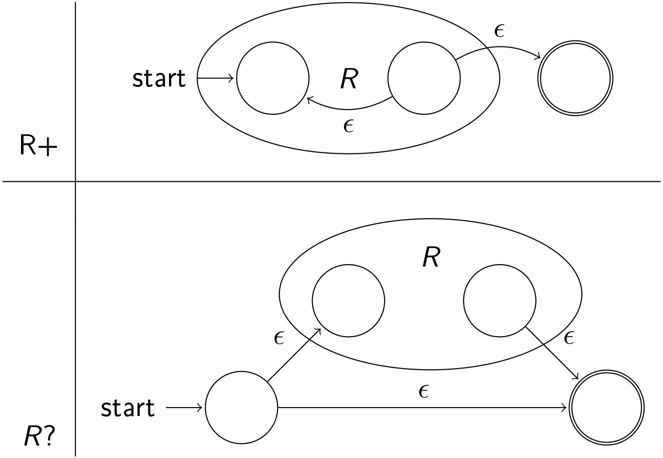
De la expresii regulare la automate finite

ϵ	
a	
$R_1 R_2$	

De la expresii regulare la automate finite



De la expresii regulare la automate finite



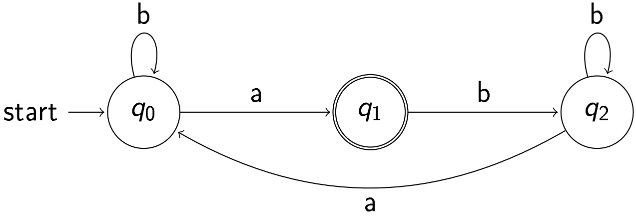
De la expresii regulare la automate finite

Aplicație facultativă

- Implementarea într-un limbaj de programare oarecare a unei aplicații care să construiască AFD care acceptă același limbaj ca și o expresie regulară dată.
- Intrare: Expresia regulară.
- Iesire:
  - Elementele de definiție ale AF-ϵ echivalent: alfabetul, mulțimea stărilor, starea inițială, mulțimea stărilor finale, graful de tranziție.
  - Codul care implementează funcționarea AFD echivalent pentru AF-ϵ de mai sus.

Automate finite vs. Gramatici formale vs. Expresii regulare - Exercițiu

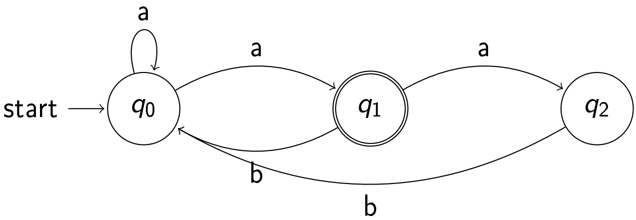
Fie automatul finit reprezentat prin graful de tranziție:



- Construiți o gramatică echivalentă.
- Construiți o expresie regulară echivalentă.

Automate finite vs. Gramatici formale vs. Expresii regulare - Exerciţiu

Fie automatul finit reprezentat prin graful de tranziţie:



- Construiţi o gramatică echivalentă.
- Construiţi o expresie regulară echivalentă.

Automate finite vs. Gramatici formale vs. Expresii regulare - Exercițiu

Fie gramatica pentru care mulțimea regulilor de producție conține:
$S \rightarrow abA$
$A \rightarrow baB$
$B \rightarrow aA \mid bb$

- Construiți un automat finit determinist echivalent.
- Construiți o expresie regulară echivalentă.

Automate finite vs. Gramatici formale vs. Expresii regulare - Exercițiu

Fie gramatica pentru care mulțimea regulilor de producție conține:

$S \rightarrow aA \mid bB$

$A \rightarrow aA \mid a$

$B \rightarrow c \mid bS$

- Construiți un automat finit determinist echivalent.
- Construiți o expresie regulară echivalentă.

Intrebari recapitulative

- Explicați ambiguitățile care pot să apară în utilizarea expresiilor regulare pentru a recunoaște atomii lexicali dintr-un text dat.
- Fie R o expresie regulară și fie expresiile regulare R^* și R^+. Scrieți care este rezultatul operației $L(R^+) \setminus L(R^*)$ și explicați.
- Explicați care este diferența dintre un automat finit și o expresie regulară relativ la definirea unui limbaj regular.

Demonstrarea/infirmarea regularității unui limbaj

Cuprins

- Demonstrarea regularității unui limbaj
  - Construirea unui automat cu stări finite sau a unei expresii regulare
- Infirmarea regularității unui limbaj
  - Principiul cuiburilor de porumbei
  - Lema pompării pentru limbajele regulare
  - Teorema Myhill-Nerode à la Brzozowski

Principiul cuiburilor de porumbei

- dacă n porumbei zboară spre m cuiburi și dacă $n>m$, atunci cel puțin un cuib va avea doi sau mai mulți porumbei
- deoarece automatele finite au un număr finit de stări și deoarece, în general, există un număr infinit de propoziții care sunt acceptate de către automat, atunci trebuie să existe cel puțin o stare la care automatul, în analiză, se întoarce de două sau de mai multe ori
- stările automatului sunt cuiburile, iar simbolurile propozițiilor de la intrare sunt porumbeii

Principiul cuiburilor de porumbei

- fie limbajul format din orice succesiune de 0 și 1 care începe cu un 1, continuă apoi cu unul sau mai mulți de zero, iar la sfârșit are un 1
- mulțimea propozițiilor acestui limbaj este infinită, cuprinzând succesiunile de simboluri: 101, 1001, 10001, $10\ldots01$
- mulțimea stărilor automatului este finită
- conform principiului cuiburilor de porumbei, trebuie să există o stare q_m astfel încât, în analiza a două succesiuni de simboluri 10^p și 10^q, cu $p \neq q$, automatul va ajunge în aceeași stare q_m

Principiul cuiburilor de porumbei

Exemplu

- $L = \{a^n b^n | n \geq 1\}$
- presupunem că limbajul L este un limbaj regular
- înseamnă că există un automat finit $AF = (\{a, b\}, Q, F, q_0, f)$ care acceptă acest limbaj
- considerăm $f^*(q_0, a^i)$ pentru i egal cu $1, 2, \ldots$
- deoarece i, numărul de a, poate lua un număr infinit de valori, și deoarece mulțimea de stări din automat este finită, atunci, conform principiului cuiburilor de porumbei, există o stare q_k astfel încât:
  $f^*(q_0, a^m) = q_k$ și $f^*(q_0, a^n) = q_k$, cu $m \neq n$.

Principiul cuiburilor de porumbei

Exemplu

- deoarece automatul accepta șirurile de forma $a^n b^n$, atunci este obligatoriu să fie adevărat că:
  $f^*(q_k, b^n) = q_f \in F$.
- pe baza relațiile scrise mai sus, se poate deduce că:
  $f^*(q_0, a^m b^n) = f^*(f^*(q_0, a^m), b^n) = f^*(q_k, b^n) = q_f \in F$
- dar acest lucru înseamnă că automatul accepta și șirurile de forma $a^m b^n$, cu $m \neq n$, ceea ce contrazice presupunerea inițială cum că el accepta limbajul L
- deci presupunerea inițială este falsă și deci L este un limbaj neregular

Lema pompării pentru limbajele regulare

Dacă L este un limbaj regular, atunci există un număr natural $N \geq 1$, astfel încât $\forall$ propoziție w $\in$ L, unde |w| $\geq$ N, $\forall$ x, y, z astfel încât w=xyz, și :

- $|xy| \leq N$
- y $\neq \epsilon$
- $\forall$ q $\geq$ 0, $xy^qz \in$ L.

Lema pompării pentru limbajele regulare

- dacă L este un limbaj regular, atunci înseamnă că există un automat finit determinist care îl definește
- presupunem că automatul are un număr $n + 1$ de stări și că acestea sunt etichetate cu $q_0, q_1, \ldots, q_n$.
- fie atunci o propoziție $w \in L$, astfel încât $|w| \geq m = n + 1$
- fie $q_0, q_i, q_j, \ldots, q_f$ succesiunea de stări prin care automatul trece în analiza propoziției w (evident q_0 este starea inițială, iar q_f este o stare finală)

Lema pompării pentru limbajele regulare

- numărul de stări din cadrul acestei succesiuni de stări este egal cu lungimea șirului w plus 1, deoarece pentru a accepta un număr de simboluri egale cu lungimea lui w, este nevoie de același număr de tranziții, iar numărul de stări necesare pentru a forma aceste tranziții este mai mare cu 1
- implică faptul că cel puțin o stare din succesiunea $q_0, q_i, q_j, \ldots, q_f$ se repetă
- prin urmare, succesiune de stări poate fi rescrisă $q_0, q_i, q_j, \ldots, q_r, \ldots, q_r, \ldots, q_f$

Lema pompării pentru limbajele regulare

- această scriere a succesiuni de stări prin care trece automatul în analiza și acceptarea propoziției w, indică că aceasta poate fi scrisă ca o concatenare a trei subșiruri x, y, z, astfel încât:
  - șirul x este prima parte a propoziției w, pentru analiza căreia automatul trece prin succesiunea $q_0, q_i, q_j, \ldots, q_r$, și se poate scrie:
    $(q_0, x) \vdash^* (q_r, \epsilon)$ sau $f^*(q_0, x) = q_r$
  - șirul y este partea de mijloc a propoziției w, pentru analiza căreia automatul trece prin succesiunea $q_r, \ldots, q_r$, și se poate scrie:
    $(q_r, y) \vdash^+ (q_r, \epsilon)$ sau $f^*(q_r, y) = q_r$
  - iar șirul z este ultima parte a propoziției w, pentru analiza căreia automatul trece prin succesiunea $q_r, \ldots, q_f$, și se poate scrie:
    $(q_r, z) \vdash^+ (q_f, \epsilon)$ sau $f^*(q_r, z) = q_f$

Lema pompării pentru limbajele regulare

- rezultă că:
  - $|xy| \leq n+1 = m$ deoarece am presupus ca singura stare care se repetă în analiza propoziției w este q_r și deci, în rest, stările sunt distincte
    prin urmare, pentru șirurile x, y se pot analiza un număr maxim de simboluri egal cu numărul maxim de stări care pot fi implicate în analiză, adică $n+1$;
  - $|y| \geq 1$, deoarece s-a arătat că starea q_r trebuie să se repete, și deci între cele două apariții ale acestei stări va fi acceptat cel puțin un simbol (în cazul în care a doua apariție a lui q_r urmează imediat după prima)

Lema pompării pentru limbajele regulare

- având în vedere relațiile de mai sus, se poate scrie că:
  1. $(q_0, xz) \vdash^* (q_r, z) \vdash^+ (q_f, \epsilon)$, ceea ce înseamnă că $xz \in L$;
  2. $(q_0, xy^2z) \vdash^* (q_r, y^2z) \vdash^+ (q_r, yz) \vdash^+ (q_r, z) \vdash^+ (q_f, \epsilon)$, ceea ce înseamnă că $xy^2z \in L$
  3. în același mod se poate arăta că $xy^3z \in L$ și $xy^4z \in L$, și, în general, $xy^qz \in L$, pentru orice număr natural q

Exemplul 1

- $L=\{a^n\ b^n \mid n \geq 1\}$
- presupunem că limbajul este unul regular
- înseamnă că există un număr natural $N \geq 1$ astfel încât pentru orice propoziție $w \in L$ care are lungimea cel puțin egală cu N se îndeplinesc afirmațiile lemei
- fie $w = a^N\ b^N$
- atunci, indiferent de modul în care propoziția w este scrisă ca și o concatenare de trei șiruri xyz, se îndeplinesc afirmațiile teoremei

Lema pompării pentru limbajele regulare

Exemplul 1

- având în vedere
  - că în conformitate cu lema pompării, $|xy| \leq N$
  - felul în care l-am ales pe w

  înseamnă că y poate lua următoarele forme:
  y=a^i, $1 \leq |y| \leq N$
- însă, pentru că $|xy| \leq N$:
  $x = a^j$, cu condiția că $i + j \leq N$
- cu aceste observații, rezultă că șirul w are forma:
  $w = a^j \ a^i \ a^{N-(i+j)} \ b^N$

Lema pompării pentru limbajele regulare

Exemplul 1

- în aceste condiții, lema afirmă că pentru orice număr natural $q \geq 0$, $xy^q z \in L$
- căutăm o valoare a lui q astfel încât:
  $w^{'} = a^j \; a^{i^q} \; a^{N-(i+j)} \; b^N \nexists$ L,
  unde $w^{'}$ este șirul pompat.
- pentru $q = 0$:
  $w^{'} = a^j \; a^{N-(i+j)} \; b^N = a^{N-i} \; b^N \notin L$
- contradicție: prin urmare, limbajul L nu este un limbaj regular

Lema pompării pentru limbajele regulare

Exemplul 2

- lema pompării nu poate fi folosită pentru a demonstra că un limbaj este regular, deoarece reciproca ei nu este adevărată: există limbaje care pot fi pompate, dar care, totuși, nu sunt regulare
- L = { uvwxy : u,y $\in$ {0,1,2,3}* ; v, w, x $\in$ {0,1,2,3} $\wedge$ (v = w $\vee$ v=x $\vee$ x=w) } $\cup$ { w : w $\in$ { 0,1,2,3 }* $\wedge$ exact 1/7 simboluri din w să fie 3 }
- limbajul nu este regular (spre exemplu $(013)^{3m}(012)^{i} \in L$ dacă și numai dacă $i = 4m$), dar poate fi pompat pentru N având valoarea 5

Lema pompării pentru limbajele regulare

Exemplul 2

- fie o propoziție w care are lungimea cel puțin egală cu 5
- deoarece alfabetul limbajului are numai 4 simboluri, înseamnă că cel puțin două dintre primele 5 simboluri sunt identice, iar aceste două simboluri identice sunt separate de cel mult trei simboluri

Lema pompării pentru limbajele regulare

Exemplul 2

- dacă cele două simboluri identice nu sunt separate de niciun simbol sau sunt separate de un singur simbol, atunci se poate pompa oricare dintre celelalte două simboluri din propoziție
  - $\mathbf{aa}bcd$, $\mathbf{a}b\mathbf{a}cd$, $b\mathbf{aa}cd$, $b\mathbf{a}c\mathbf{a}d$, $bc\mathbf{aa}d$, $bc\mathbf{a}d\mathbf{a}$, $bcd\mathbf{aa}$, etc.
  - pentru prima varianta se poate pompa simbolul c sau simbolul d
  - $\mathbf{aa}bc^q d$ sau $\mathbf{aa}bcd^q$
- dacă cele două simboluri identice sunt separate de două sau de trei simboluri, atunci pot fi pompate două dintre simbolurile care le separă
  - $\mathbf{a}bc\mathbf{a}d$, $\mathbf{a}bcd\mathbf{a}$, $b\mathbf{a}cd\mathbf{a}$, etc.
  - pentru prima varianta se poate pompa grupul bc
  - $a(bc)^q ad$

Lema pompării pentru limbajele regulare - Exercițiu

Să se demonstreze că următoarele limbaje nu sunt limbaje regulare:

- L=$\{ab^nc^n|$ n$\geq$ 1$\}$
- L=$\{a^nb^lc^{n+l}|$ n, l$\geq$ 0$\}$
- L = $\{ a^ib^j$ | i $>$ j $\}$
- L = $\{ a^{n!}|n \geq 0 \}$

Teorema Myhill-Nerode à la Brzozowski

- **derivata unui limbaj** L **în raport cu un simbol** c este un nou limbaj obținut prin aplicarea următoarelor două operații:
  1. se rețin toate propozițiile limbajului L care încep cu simbolul c.
  2. se elimină simbolul c din poate aceste propoziții.
- derivata unui limbaj L în raport cu un simbol c se definește astfel:
  $\frac{d}{dc}\mathrm{L} = \{\ \mathrm{v} \mid \mathrm{cv} \in \mathrm{L}\ \}$
- exemple:
  - $\frac{d}{da}$ ab*a = b*a
  - $\frac{d}{db}$ ab*a $=\emptyset$
  - $\frac{d}{db}$ b*a = b*a

Teorema Myhill-Nerode à la Brzozowski

- **derivata unui limbaj regular este tot un limbaj regular**
- orice limbaj regular poate fi definit prin intermediul unui automat cu stări finite
- dar, operația de derivare, prin care se obține limbajul derivat, corespunde schimbării stării inițiale a automatului finit care definește limbajul supus operației de derivare
- prin urmare și derivata unui limbaj este definită printr-un automat cu stări finite, ceea ce înseamnă că este un limbaj regular

Teorema Myhill-Nerode à la Brzozowski

- **un limbaj regular are un număr finit de derivate distincte**
- având în vedere că
  - operația de derivare a unui limbaj înseamnă de fapt schimbarea stării inițiale a automatului cu stări finite care definește limbajul supus derivării
  - numărul de stări ale unui automat este finit
- rezultă că numărul de derivate posibile este maxim egal cu numărul de stări ale automatului și deci este un număr finit
- reciproca afirmației este și ea adevărată: **dacă un limbaj are un număr finit de derivate distincte, atunci el este un limbaj regular**

Teorema Myhill-Nerode à la Brzozowski

Exemplul 1

- L = { $a^i b^i$ | $\forall$ i $\in \mathbb{N}$ }
- pentru a arăta că limbajul nu este regular, se va arăta că numărul de derivate distincte ale limbajului este infinit
- de exemplu, se va deriva limbajul dat în raport cu a^n
- $\frac{d}{da^n}\ a^{i+n}\ b^{i+n} = a^i\ b^{i+n}, \forall i, n \in \mathbb{N}$
- depinzând de n, este evident că numărul de derivate distincte obținute este infinit
- prin urmare, limbajul nu este regular

Exemplul 2

- L $= \{ a^i b \mid \forall$ i $\in \mathbb{N} \}$
- $\frac{d}{da^n}\ a^{i+n}\mathrm{b} = a^i b$
- numărul de derivate este tot infinit, însă ele nu sunt distincte
- prin urmare limbajul este unul regular

Exemplul 3

- L = { uvwxy :
  u,y $\in$ {0,1,2,3}*;
  v, w, x $\in$ {0,1,2,3} $\wedge$ (v = w $\vee$ v=x $\vee$ x=w) }
  $\cup$ { w $\in$ {0,1,2,3}*| ($n_{\{0,1,2\}}(w) = 6 * n_{\{3\}}(w)$ }
- notând cu C a doua submulțime a propozițiilor din limbajul L, vom calcula derivata:
- $\frac{d}{d(0123)^i}$C

Exemplul 3

- $= \{w \in \{0123\}^* \mid n_{\{0,1,2\}}((0123)^i w) = 6 * n_{\{3\}}((0123)^i w)\ \}$
- $= \{w \in \{0123\}^* \mid n_{\{0,1,2\}}((0123)^i) + n_{\{0,1,2\}}(w) = 6 * (n_{\{3\}}((0123)^i) + n_{\{3\}}(w))\}$
- $= \{w \in \{0123\}^* \mid 3 * i + n_{\{0,1,2\}}(w) = 6 * i + 6 * n_{\{3\}}(w)\ \}$
- $= \{w \in \{0123\}^* \mid n_{\{0,1,2\}}(w) = 3 * i + 6 * n_{\{3\}}(w)\ \}$
- aceste limbaje sunt distincte
  - spre exemplu, pentru fiecare valoare a lui i derivata $\frac{d}{d(0123)^i} C$ conține șirul $(012)^i$
  - însă niciuna dintre celelalte derivate nu conține acest șir
- prin urmare, L are un număr infinit de derivate distincte și deci nu este un limbaj regular

Intrebari recapitulative

- De ce nu se poate folosi lema pompării pentru a demonstra că un limbaj este regular?
- Explicați care este diferența dintre reciproca lemei pompării și reciproca teoremei Myhill-Nerode à la Brzozowski.

Intrebari recapitulative

- Explicați legătura dintre caracterul finit al mulțimii stărilor automatelor finite și Teorema Myhill-Nerode à la Brzozowski.
- Explicați rolul principiului cuiburilor de porumbei în înțelegerea limbajelor regulare.

Intrebari recapitulative

- Fie Σ alfabetul unui limbaj L, astfel încât $a, b \in \Sigma$ și $x \in \Sigma^*$. Cunoscând că limbajul L are proprietățile:
  - $ax \in L \Leftrightarrow bx \in L$,
  - numărul de propoziții de forma ax din L este k.

  Să se scrie care este rezultatul operației:
  card(L) – card(d/da L) – card(d/db L).

Implementarea automatelor finite deterministe

Cuprins

- Exemplificare la tablă

Bibliografie

- Halvor Eifring, Rolf Theil, Linguistics for Students of Asian and African Languages, Oslo, 2005 https://www.uio.no/studier/emner/hf/ikos/EXFAC03-AAS/h05/larestoff/linguistics/
- Paul Kay, Willett Kempton, What is the Sapir-Whorf hypothesis?, 1984 http://www.blutner.de/color/Sapir-Whorf.pdf
- Umberto Eco, “Limbajul Pământului Austral”, “De la arbore la labirint”, editura POLIROM, 2009.
- Vlad Tarko, The Metaphysics of Object Oriented Programming http://news.softpedia.com/news/The-Matephysics-of-Object-Oriented-Programming-24906.shtml
- Wikipedia, Programming paradigm http://en.wikipedia.org/wiki/Programming_paradigm

Bibliografie

- Irina Athanasiu, Limbaje formale și translatoare, 2002 http://www.scribd.com/doc/22056333/Limbajeformale-%C5%9Fi-automate
- Lila Ghemri, The Pigeonhole principle with proof, 2007 http://cs.tsu.edu/ghemri/CS248/ClassNotes/Non_Reg_Lang.pdf
- Ben Wiedermann, How to Use The Pumping Theorem, 2004 https://www.cs.hmc.edu/~benw/teaching/cs341_fa04/pumping.pdf
- Bosker Blog, I hate the Pumping Lemma https://bosker.wordpress.com/2013/08/18/i-hate-the-pumping-lemma/

Bibliografie

- Costas Busch, More Applications of the Pumping Lemma, 2006 `https://slideplayer.com/slide/5322383/`
- N. Murugesan, O. V. Sundaram, "Some Properties of Brzozowski Derivatives of Regular Expressions", arXiv preprint arXiv:1407.5902, 2014 `https://arxiv.org/abs/1407.5902`
- Matthew Might, David Darais, Daniel Spiewak, "Parsing with Derivatives: a Functional Pearl", ICFP'11, September 19–21, 2011, Tokyo, Japan `https://dl.acm.org/doi/abs/10.1145/2034574.2034801`