VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA GRAMATIKÁCH ŘÍZENÝCH STROMY

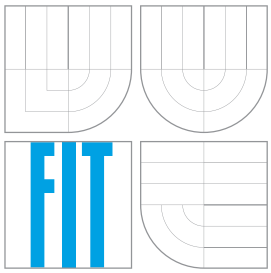
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTĚPÁN GRANÁT

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA GRAMATIKÁCH ŘÍZENÝCH STROMY

PARSING BASED ON TREE-CONTROLED GRAMMARS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTĚPÁN GRANÁT

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. RNDr. ALEXANDR MEDUNA

BRNO 2016

Abstrakt

Výtah (abstrakt) práce v českém jazyce.

Abstract

Výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Klíčová slova v českém jazyce.

Keywords

Klíčová slova v anglickém jazyce.

Citace

Štěpán Granát: Syntaktická analýza založená
na gramatikách řízených stromy, bakalářská práce, Brno, FIT VUT v Brně, 2016

Syntaktická analýza založená na gramatikách řízených stromy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandra Meduny, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Štěpán Granát
14. března 2016

Poděkování

Zde je možné uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc.

Obsah

Kapitola 1

Úvod

Kapitola 2

Definice a pojmy

2.1 Teorie množin

2.2 Teorie grafů

Definice 2.2.1. (Strom). [1, str. 12]
Strom je orientovaný acyklický graf, $G = (\Sigma, R)$, který má tyto tři vlastnosti:

G má právě jeden uzel, do něhož nevstupují žádné hrany; tento uzel se nazývá kořen G označovaný jako kořen(G).

Jestliže $a \in \Sigma$ a $a \neq$ kořen(G), potom a je potomkem kořenu(G) a vstupuje do něj právě jedna hrana.

Každý uzel $a \in \Sigma$ který není listem má svého přímého potomka, b_1 až b_n, řazené zleva doprava tak, že b_1 je nejlevějším přímým potomkem a a b_n je nejpravějším přímým potomkem a.

Definice 2.2.2. (Úroveň, cesta, řez, hranice, hloubka, elementární strom a podstrom). [1, str. 12]
Nechť $G = (\Sigma, R)$ je stromem.

Úroveň l, stromu G, je posloupnost s, všech uzlů se stejnou vzdáleností od kořene(G). Jinými slovy, úroveň l, je posloupnost, $s = n_1 n_2 ... n_k$, taková, že existuje cesta v grafu o délce ℓ v G pro všechny posloupnost od kořene(G) $...n_i$, pro $1 \leq i \leq k$ a $l \geq 1$.

Cesta p, stromu G, je posloupnost s, uzlů, kde první uzel je kořen(G), poslední je listem a mezi každými dvěma následnými uzly v s existuje hrana v G. Jinými slovy, cesta p stromu G, je posloupnost, $s = n_1 n_2 ... n_k$, taková, že s je cesta grafem o délce k v G, kde $n_1 =$ kořen(G) a n_k je listem v G, pro $k \geq 1$.

Řez c, stromu G je posloupnost s, uzlů takových, že každá cesta v G má právě jeden uzel v c. Jinými slovy, řez c je posloupnost, $s = n_1 n_2 ... n_k$, taková, že pro každou cestu $p = m_1 m_2 ... m_\ell$ stromu G, $|\{n_1, n_2, ..., n_k\} \cap \{m_1, m_2, ..., m_\ell\}| = 1$, pro $k, \ell \geq 1$.

Hranice stromu G, hranice(G), je posloupnost listů G řazených zleva do prava.

Hloubka stromu G, hloubka(G), je délka nejdelší cesty v G; jestliže platí hloubka(G)$= 1$, potom je G elementární strom.

Jestliže $G' = (\Sigma', R')$ představuje strom vyhovující těmto čtyřem podmínkám: $\Sigma' \neq \emptyset$; $\Sigma' \subseteq \Sigma$; $R' = (\Sigma' \times \Sigma')$; a jestliže v G není žádný z uzlů v $\Sigma - \Sigma'$ potomkem uzlu v Σ', potom je G' podstromem G.

Kapitola 3

Formální jazyky

Teorie formálních jazyků si bere za cíl formalizovat obecně jazyky (přirozené, programovací, matematické, ...) tak, abychom se mohli zabývat jejich automatizovaným zpracováním. Pojmy, které známe z lingvistiky jsou zde zobecněny a přesně definovány, takže nemusí přesně odpovídat naší dosavadní představě.

Abeceda

Základem jazyka je *abeceda*. V teorii formálních jazyků je obvykle značena Σ (sigma) a je definována jako konečná neprázdná množina, jejíž objekty se nazývají *symboly*.

Řetězec

Konečná posloupnost symbolů patřících do Σ je *řetězec* nad Σ. Zvláštním případem je ε (epsilon), značící *prázdný řetězec* - tedy takový, že neobsahuje žádný symbol.

Jazyk

Σ^* značí množinu všech řetězců, které je možné sestrojit nad abecedou Σ. Jakákoliv podmnožina $L \subseteq \Sigma^*$ je *jazykem* nad abecedou Σ. Jestliže *jazyk* představuje konečnou množinu řetězců, potom jej nazýváme *konečným jazykem*, v opačném případě *jazykem nekonečným*.

Gramatika

Pokud se zabýváme nekonečnými jazyky, nemůžeme je vyjádřit jednoduchým výčtem jejich řetězců. Místo toho definujeme *gramatiku*, která stanovuje pravidla pro generování řetězců patřících do daného jazyka.

Gramatika obecně obsahuje 4 části:

- Množinu *neterminálních symbolů* N (neterminálů), které slouží k označení syntaktických celků.
- Množinu *terminálních symbolů* Σ (terminálů) - symboly, které jsou konečným výstupem. (abeceda)
- Množinu přepisovacích pravidel P.
- Počáteční (startovací) symbol $S \in N$.

Přepisovací pravidla

Přepisovací pravidlo je složeno ze dvou řetězců (α, β), složených z terminálů a neterminálů, přičemž α obsahuje alespoň jeden neterminál. Zapisují se jako $\alpha \to \beta$.
Pravidla se aplikují od startovacího symbolu, kdy postupně přepisujeme řetězec tak že nahradíme jakoukoliv část řetězce, která se nachází na levé straně některého pravidla za pravou stranu tohoto pravidla. Řetězec upravujeme podle pravidel tak dlouho, až se v něm nacházejí pouze neterminály.

řetězec	pravidlo	výsledek
a**Q**e	`Q -> bcd`	a**bcd**e

Ekvivalence gramatik

Dvě gramatiky označujeme jako *ekvivalentní*, pokud generují stejný jazyk.

Hierarchie jazyků

Protože obecná gramatika může obsahovat pravidla, která lze těžko zpracovávat, v praxi se gramatiky dále omezují - přesněji omezujeme tvar pravidel gramatiky. Omezením gramatiky získáme výhodu rychlejšího (a jednoduššího) zpracování, ovšem na úkor vyjadřovací síly.

Gramatiky se nejčastěji rozdělují podle Chomského hierarchie:

- **Gramatiky typu 0** (frázové/neomezené gramatiky)
  Zahrnují všechny formální gramatiky.
  Model pro zpracování se nazývá *Turingův stroj*.
  Tvoří třídu *rekurzivně spočetných jazyků*, zkratka **RE**.
- **Gramatiky typu 1** (kontextové gramatiky)
  Tyto gramatiky se skládají z pravidel typu $\alpha A \beta \to \alpha \gamma \beta$, kde A je neterminál a α, β, γ jsou řetězce terminálů i neterminálů, přičemž γ je neprázdný.
  Model pro zpracování se nazývá *lineárně ohraničený Turingův stroj*.
  Tvoří třídu *kontextových jazyků*, zkratka **CS**.
- **Gramatiky typu 2** (bezkontextové gramatiky)
  Skládají se z pravidel typu $A \to \gamma$, kde A je neterminál a γ řetězec terminálů a neterminálů.
  Model pro zpracování se nazývá *nedeterministický zásobníkový automat*.
  Tvoří třídu *bezkontextových jazyků*, zkratka **CF**.
- **Gramatiky typu 3** (regulární gramatiky)
  Skládají se z pravidel typu $A \to a$ a $A \to aB$, kde A, B jsou neterminály a a je terminál.
  Model pro zpracování se nazývá *konečný automat*.
  Tvoří třídu *regulárních jazyků*, zkratka **REG**.

V této práci se budeme zabývat skupinami 2 a 3, tedy bezkontextovými a regulárními jazyky. Tyto jazyky jsou prodrobněji popsány v následujících kapitolách.

3.1 Regulární jazyky

Regulární jazyky jsou nejjednodušší formální jazyky v Chomského hierarchii. I přesto si však našly široké využití v různých oblastech informačních technologií. Využívají se např. pro pokročilé vyhledávání v textu nebo pro rozdělení programovacího jazyka na základní jednotky (tokeny). V implementační části této práce jsou využity ke kontrole úrovní derivačního stromu, také proto se jimi budeme hlouběji zabývat.

Definice 3.1.1. (Regulární jazyk)
Regulární jazyk nad abecedou Σ lze definovat následovně:

- Prázdný jazyk $\emptyset$ je regulární.
- Pro každé a z Σ je $\{a\}$ regulární.
- Jestliže A a B jsou regulární jazyky, poté všechny tyto jazyky jsou také regulární: $A \cup B$ (sjednocení), AB (konkatenace) a A^* (iterace).

Konečné automaty

Každý Regulární jazyk lze zpracovávat konečným automatem a každý konečný automat lze vyjádřit regulárním jazykem.

Automat je pětice $M = (Q, \Sigma, R, s, F)$, kde:

- Q je množina stavů
- Σ je *vstupní abeceda*
- R je množina *přechodových pravidel*
- s je *počáteční stav*
- F je množina *konečných stavů*

Přechodová pravidla jsou ve tvaru $qa \to p$, kde q, p jsou stavy a a je vstupní symbol. Pravidlo nám říká, že jsme-li ve stavu q a na vstupu máme symbol a, poté může automat přejít do stavu p. Začínáme vždy v počátečním stavu a aby vstupní řetězec patřil do jazyka, musíme skončit v jednom z konečných stavů. Velkou výhodou je, že práce konečného automatu je paměťově velmi nenáročná, jelikož obsahuje pouze informaci o aktuálním stavu.

Příklad 3.1.2. Mějme konečný automat M1:

```
M1 = (
  {s, q, f},
  {a, b, c},
  {
    sa -> q1,
    qb -> q,
    qc -> f
  },
  s,
  {f}
)
```

A řetězec:

abbc

Při kontrole vstupního řetězce budeme postupovat následovně:

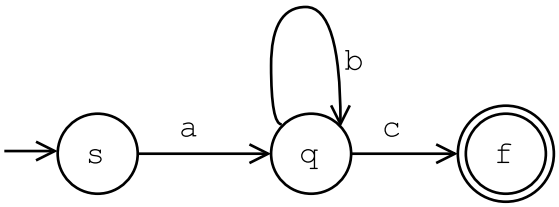
1. Nastavíme počáteční stav s
2. Vstupním symbolem je 'a' - podle prvního pravidla přejdeme do stavu q
3. Vstupním symbolem je 'b' - zůstáváme ve stavu q (pr. 2)
4. Vstupním symbolem je 'b' - zůstáváme ve stavu q (pr. 2)
5. Vstupním symbolem je 'c' - přejdeme do stavu f (pr. 3)
6. Jsme na konci řetězce - zkontrolujeme, zdali jsme v konečném stavu - řetězec byl automatem přijat, takže řetězec patří do jazyka generovaného automatem.

Během činnosti konečného automatu mohou nastat tyto chyby:

- vstupní symbol nepatří do abecedy
- neexistuje pravidlo pro vstupní symbol a aktuální stav
- po přečtení posledního znaku se nenacházíme v konečném stavu

Ve všech těchto případech není vstupní řetězec přijat konečným automatem, tudíž nepatří do jazyka generovaného automatem.

Tento konečný automat lze také zobrazit pomocí následujícího grafu:



Uveďme další příklad, který už nebude tak jednoduchý:

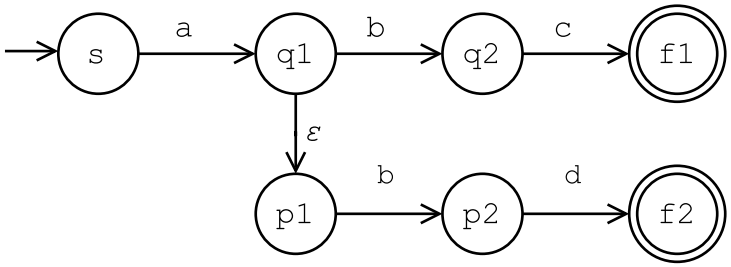
Příklad 3.1.3. Mějme konečný automat M2:

```
M2 = (
  {s, q1, q2, p1, p2, f1, f2},
  {a, b, c, d},
  {
    sa -> q1,
    q1b -> q2,
    q2c -> f1,
    q1ε -> p1,
    p1b -> p2,
    p2d -> f2
  },
```

```
    s1 ,
    {f1}
)
```

Za pozornost stojí hlavně čtvrté pravidlo s ε přechodem. Toto pravidlo značí, že lze bez přijetí jakéhokoliv znaku přejít z jednoho stavu do druhého. Tyto pravidla se bohužel mohou v obecných gramatikách vyskytovat a způsobují potíže při zpracovávání automatu, jak bude vysvětleno dále.

Pro větší názornost budeme nyní pracovat se schématem automatu:



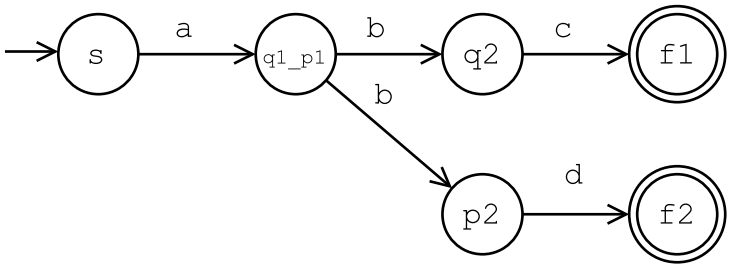
Pokud se při zpracování ocitneme ve stavu $q1$ a vstupním symbolem bude 'b', není jasné, jestli máme použít 2. nebo 4. pravidlo. Museli bychom vyzkoušet obě možnosti a až zpětně bychom zjistili, která možnost byla správná. Tomuto jevu se v teorii formálních jazyků říká *nedeterminismus* a setkáme se s ním ještě několikrát.

Nutno ale podotknout, že *nedeterminismus* neznamená to, že bychom nebyli schopni určit, jestli řetězec patří do daného jazyka. Ve skutečnosti totiž můžeme vyzkoušet všechna možná pravidla a zjistit, jestli z nich některé povede k úspěchu. Tato technika se také pro některé jazyky využívá a při zpracování přirozených jazyků se jí většinou nelze úplně vyhnout. Slepé zkoušení pravidel však vede k velkému zpomalení vyhodnocování, může totiž dojít k tomu, že se program bude větvit opakovaně a zpracování může dojít až k exponencionální složitosti. Např. programovací jazyky bývají navrženy tak, aby šly zpracovávat deterministicky, protože rychlost překladu je velmi důležitým faktorem pro jejich nasazení.

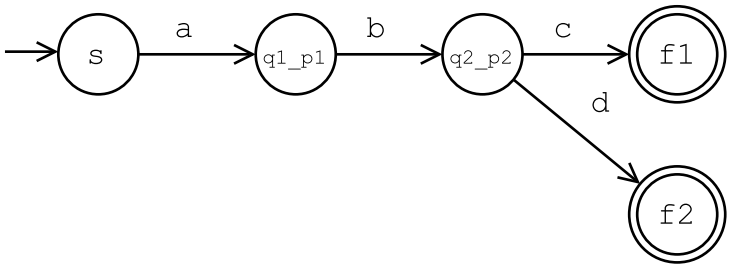
Determinizace konečného automatu

Z výše uvedených odstavců je zjevné, že je lepší se preventivně nedeterminismu zbavit. Nejprve ukážeme demonstraci na tomto konkrétním příkladě a poté uvedeme obecný algoritmus.

Abychom se zbavili ε přechodů musíme vytvořit nový automat, který ale generuje stejný jazyk (přijímá stejné řetězce) jako ten původní. Takovéto dva automaty se označují jako *ekvivalentní*. Protože můžeme z přechodu $q1$ kdykoliv přejít do $q2$ intuitivním řešením je oba stavy spojit do jednoho. Pro zachování *ekvivalence*, musíme do nového stavu přidat všechna pravidla, která vycházela z původních dvou stavů. Nový automat bude vypadat takto:



Pokud si nové schéma dobře prohlédneme, odhalíme další zádrhel, který se nám objevil v novém pravidle $q1\_p1$. Pokud je v tomto stavu na vstupu symbol 'b', nevíme které pravidlo použít a máme tu opět *nedeterminismus*. I tento problém lze naštěstí vyřešit obdobným způsobem:



Tento automat můžeme označit jako *deterministický* a lze jej s klidem implementovat. V tomto konkrétním případě jsme si pomohli intuicí, uveďme ale obecné algoritmy:

Algoritmus 1: Stavy dostupné bez čtení ze stavů E (*ε-uzávěr(E)*)

Input: Konečný automat $M = (Q, \Sigma, R, s, F)$ a $E \subseteq Q$
Output: *ε-uzávěr(E)*

```
1 begin
2     ε-uzávěr(E) := E;
3     repeat
4         ε-uzávěr(E) := ε-uzávěr(E) ∪ {p|q → p ∈ R a q ∈ ε-uzávěr(E)}
5     until ε-uzávěr(E) byl změněn;
```

Algoritmus 2: Odstranění ε pravidel

Input: Konečný automat $I = (Q_I, \Sigma_I, R_I, s_I, F_I)$ a $E \subseteq Q$
Output: Konečný automat bez ε pravidel O, ekvivalentní s I

```
1 begin
2     Q_O := Q_I;
3     Σ_O := Σ_I;
4     s_O := s_I;
5     F_O := {q|q ∈ Q_I, ε-uzávěr(q) ∩ F_I ≠ ∅};
6     R_O := {qa → p|q ∈ Q_I, a ∈ Σ_I, oa → p ∈ R_I pro všechna o ∈ ε-uzávěr(q) v I}
```

Algoritmus 3: Odstranění nedeterminismu

Input: Konečný automat bez ε-přechodů $M = (Q, \Sigma, R, s, F)$
Output: Deterministický KA: $D = (Q_D, \Sigma, R_D, s_D, F_D)$ ekvivalentní s M

begin
  $s_D := \{s\}$;
  $Q_{new} := \{s_D\}$;
  $R_D := \emptyset$;
  $Q_D := \emptyset$;
  $F_D := \emptyset$;
  repeat
    nechť $Q' \in Q_{new}$;
    $Q_{new} := Q_{new} - \{Q'\}$;
    $Q_D := Q_D \cup Q'$;
    forall the $a \in \Sigma$ **do**
      $Q'' := \{q | p \in Q', pa \to q \in R\}$;
      if $Q'' \neq \emptyset$ **then**
        $R_D := R_D \cup \{Q'a \to Q''\}$;
      if $Q'' \notin Q_D \cup \{\emptyset\}$ **then**
        $Q_{new} := Q_{new} \cup \{Q''\}$;
    if $Q' \cap F \neq \emptyset$ **then**
      $F_D := F_D \cup \{Q'\}$
  until $Q_{new} = \emptyset$;

Tyto algoritmy jsou definovány pro jakýkoliv Konečný automat, tedy jakýkoliv KA lze převést na Deterministický KA [2, str. 39]. Z toho vyplývá, že jakýkoliv Regulární jazyk lze zpracovávat deterministicky, což u jazyků z ostatních tříd *Chomského hierarchie* neplatí.

Existují také další transformace Konečného automatu, jako například: minimalizace a odstranění nedostupných stavů. Ty slouží ale spíše k jakémusi uhlazení automatu - při práci konečného automatu nám nemusí vadit, že jsou některé stavy nedostupné, protože se k nim ani nemůžeme dostat. Bez nich by sice zabíral méně paměťi, ale odstraňování těchto stavů, stojí zase výpočetní čas.
Podobné je to i s minimalizací - to že uživatel zadá více stavů, než je nutné, můžeme brát jako jeho osobní věc a to že bychom nepotřebné vypustili, by ho mohlo spíše poplést. Dopad na výkon automatu je zde zanedbatelný.

3.2 Bezkontextové jazyky

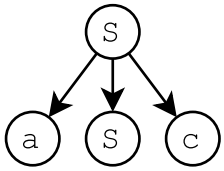
3.2.1 Derivační strom

Při aplikaci gramatiky na řetězec jde vlastně o ověření, jestli lze z počátečního symbolu, postupnou derivací (aplikací pravidel gramatiky) získat daný řetězec. Mějme například gramatiku:

```
G = (
  {S, a, b, c},
```

```
    {a, b, c},
    {
      S -> aSc,
      S -> b
    },
    S
)
```

Pravidla této gramatiky lze vyjádřit pomocí elementárního stromu, kde levá strana je kořen a pravá strana představuje jeho potomky. Např. první pravidlo z gramatiky G lze znázornit stromem takto:



Pro menší velikost grafu budeme ale pravidla zjednodušeně znázorňovat takto:

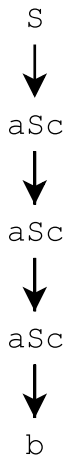
Pro příklad postupné derivace mějme řetězec s:

```
aaabccc
```

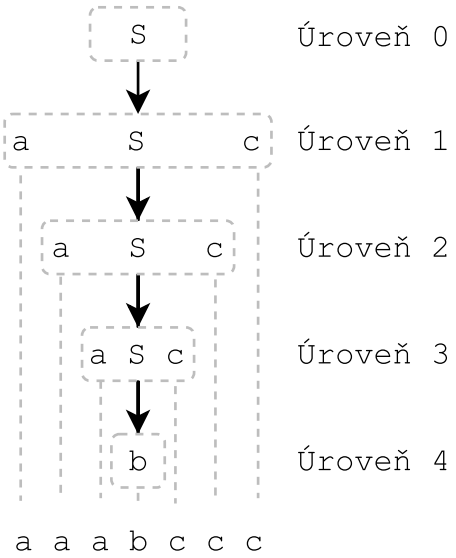
Nyní zkusíme postupně aplikovat pravidla na počáteční symbol, tak abychom získali řetězec s (v komentáři jsou uvedena pravidla, která byla použita):

```
S
aSc                    //S -> aSc
aaScc                  //S -> aSc
aaaSccc                //S -> aSc
aaabccc                //S -> b
```

Touto posloupností derivací jsme byli schopni dosáhnout kontrolovaného řetězce s, daný řetězec tedy patří do gramatiky. Výše zobrazené derivace lze zobrazit i jinak a to pomocí tzv. `Derivačního stromu`:



Pro větší názornost ukažme ještě stejný strom s přiřazenými symboly k původnímu řetězci a vyznačenými jednotlivými úrovněmi.



Definice 3.2.1. (Derivační strom). [2, str. 92]
Nechť $G = (\Sigma, R)$ je Bezkontextová gramatika.

1. Pro l: $A \to x \in R, A\langle x \rangle$ je strom pravidla, které reprezentuje l.
2. Derivační strom reprezentující derivace v G je definován rekurzivně:
   (a) Strom s jedním uzlem X je derivační strom odpovídající $X \Rightarrow^0 X$ v G, kde $X \in \Sigma$.
   (b) Nechť d je derivační strom reprezentující $A \Rightarrow^0 uBv\ [\rho]$ s hranicí$(d) = uBv$, a nechť $l : B \to z \in R$. Derivační strom, který reprezentuje

$$\begin{aligned} A &\Rightarrow^* uBv[\rho] \\ &\Rightarrow\ uzv[l] \end{aligned}$$

je získán nahrazením $(|u|+1)$-tého listu v d, B, stromem pravidla odpovídajícho l, $B\langle z\rangle$

3. Derivační strom v G je jakékoliv t, pro které existuje derivace odpovídající t (viz 2.).

Kapitola 4

Zpracování jazyků řízených stromy

4.1 Omezení úrovní derivačního stromu

V této části práce je vysvětleno, jak lze omezit derivační stromy a tím zvýšit sílu dané gramatiky. Teoretická část této sekce vychází převážně z práce Ing. Koutného[1].

4.1.1 Definice

Definice 4.1.1. (Gramatiky řízené stromy) [1, str. 28]
Gramatika řízená stromem je dvojice (G, R), kde $G = (V, T, P, S)$ je řízená gramatika a $R \subseteq V^*$ kontrolní jazyk.

Definice 4.1.2. (Jazyky řízené stromy) [1, str. 28]
Nechť (G, R) je gramatika řízená stromem. Jazyk generovaný (G, R) je značen jako $L(G, R)$ a definován jako:

$L(G, R) = \{x : x \in L(G, R)$ a existuje derivační strom t pro každé x v G takový, že každé slovo získané konkatenací všech symbolů na kterékoliv úrovni t (kromě poslení) zleva doprava, patří do $R\}$.

V této práci se zabýváme především gramatikami bezkontextovými, protože je lze relativně snadno zpracovávat a kontrolní jazyk bude regulární. Jinými slovy, zabýváme se gramatikami (G, R), kde G je bezkontextová gramatika a R je regulární jazyk.

4.1.2 Příklady

Pro lepší porozumění následuje podrobný praktický příklad, jak lze ověřovat příslušnost řetězce do dané gramatiky řízené stromem.

Příklad 4.1.3. Mějme stromem řízený jazyk $L(G, R)$, kde:

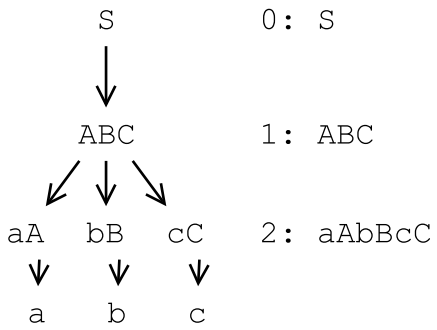
```
G = (
  {S, A, B, C, a, b, c},
  {a, b, c},
  {
    S -> ABC,
    A -> aA,
```

```
      A -> a,
      B -> bB,
      B -> b,
      C -> cC,
      C -> c
    },
    S
),
R = {S, ABC, aAbBcC}
```

A řetězec:

```
aabbcc
```

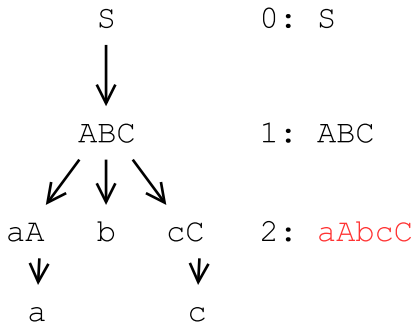
Nejprve sestavíme derivační strom pro daný řetězec, poté projdeme všechny jeho úrovně (kromě poslední, viz. Definice 4.1.2) a ověříme, že patří do kontrolního jazyka R.



Z grafu je zřejmé, že v tomto případě řetězec patří do jazyka L. Zkusme však ještě jiný případ pro tento řetězec:

```
aabcc
```

A jemu odpovídající derivační strom:



V tomto případě úroveň 2 derivačního stromu nepatří do kontrolního jazyka R, proto tento řetězec nepatří do jazyka L.

4.2 Syntaktická analýza

V předchozích kapitolách jsme se vždy zabývali pouze tím, jak pracovat s derivačním stromem, ale ne jakým způsobem ho sestavit pro daný řetězec a gramatiku. Metody řešící tento problém jsou již podrobně prozkoumané a obecně známé, proto není tolik vysvětlován jejich princip, ale spíše jejich vlastnosti, mající vliv na podobu derivačního stromu.

Při konstrukci syntaktického analyzátoru se vždy objeví otázka, zdali je lepší při konstrukci derivačního stromu postupovat od kořene (shora dolů) nebo od zkoumaného řetězce (zdola nahoru). Odpověď na tuto otázku není jednoznačná, což je vidět i na široce používaných analyzátorech programovacích jazyků, kdy se tato technika liší projekt od projektu.

V případě tohoto projektu je tomu nejinak, a proto v této části budeme rozebírat obě alternativy, aby byly zřejmé jejich výhody i nevýhody.

4.2.1 Shora dolů

4.2.2 Zdola nahoru

Kapitola 5

Implementace

Literatura

[1] Koutný, J.: *Gramatiky s omezenými derivačními stromy, phd thesis*. Brno, FIT VUT v Brně, 2002.

[2] Meduna, A.: *Formal languages and computation: models and their applications*. Taylor Francis, New York, 2014, ISBN 978-1-4665-1345-7.

[3] Rábová, Z.; Hanáček, P.; Peringer, P.; aj.: Užitečné rady pro psaní odborného textu [online]. `http://www.fit.vutbr.cz/info/statnice/psani_textu.html`, 2008-11-01 [cit. 2008-11-28].