Univerzita Komenského, Bratislava

Fakulta Matematiky, Fyziky a Informatiky

# Detekcia nejednoznačnosti bezkontextových gramatík

Diplomová práca

Bc. Ján Ruhalovský, 2014


Univerzita Komenského, Bratislava

Fakulta Matematiky, Fyziky a Informatiky

# Detekcia nejednoznačnosti bezkontextových gramatík

Diplomová práca

**Študijný program:** Informatika
**Študijný odbor:** 2508 Informatika
**Školiace pracovisko:** Katedra informatiky
**Vedúci práce:** RNDr. Richard Ostertág, PhD.

Bratislava, 2014

Bc. Ján Ruhalovský


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc. Ján Ruhalovský
**Študijný program:** informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
**Študijný odbor:** 9.2.1. informatika
**Typ záverečnej práce:** diplomová
**Jazyk záverečnej práce:** slovenský

**Názov:** Detekcia nejednoznačnosti bezkontextových gramatík
*Detecting ambiguity of context-free grammars*

**Cieľ:** Ciele práce sú nasledovné:
* spracovanie prehľadu metód detekcie nejednoznačnosti bezkontextových gramatík (úplné preberanie, aproximačné metódy, kombinované metódy)
* hľadanie a popis zlepšení metód detekcie (najmä zlepšením aproximácie)
* implementácia navrhnutých zlepšení
* empirické porovnanie existujúcich a zlepšených metód

Pre implementáciu bude použitý programovací jazyk Java.

**Vedúci:** RNDr. Richard Ostertág, PhD.
**Katedra:** FMFI.KI - Katedra informatiky
**Vedúci katedry:** doc. RNDr. Daniel Olejár, PhD.

**Spôsob sprístupnenia elektronickej verzie práce:**
bez obmedzenia

**Dátum zadania:** 31.10.2012

**Dátum schválenia:** 19.11.2012 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že som túto prácu vypracoval sám za pomoci uvedenej literatúry a konzultácií s vedúcim práce.

. . . . . .

# Poďakovanie

Ďakujem vedúcemu mojej diplomovej práce, RNDr. Richardovi Ostertágovi, PhD., za vedenie tejto práce, cenné rady pri konzultáciách, jeho čas a ústretovosť a za zadanie témy. Moja vďaka patrí aj pánovi Andersovi Møllerovi za poskytnutie zdrojových kódov k frameworku ACLA, taktiež rodine a kamarátom, predovšetkým za trpezlivosť a podporu pri písaní.

# Abstract

In this thesis we deal with ambiguity detection of context-free grammars. We will show the existing different approaches to ambiguity detection. They will be divided into three basic categories: exhaustive methods, approximation methods and combined methods. Other methods which do not fall clearly into one of these categories will be categorized as "others". This thesis focuses mostly on approximation methods and ACLA framework. We implement cycle-breaking approximation strategy into this framework and this new strategy will be compared with previous implementation. In the end, we will show that this new cycle-breaking strategy provides better results in ambiguity detection of context-free grammars.

**KEYWORDS:** ambiguity detection of context-free grammars, regular approximation of context-free grammars, approximation strategy, methods of ambiguity detection

# Abstrakt

V tejto práci sa zaoberáme detekciou nejednoznačnosti bezkontextových gramatík. Ukážeme aké rôzne prístupy na zisťovanie nejednoznačnosti existujú. Rozdelíme ich do troch základných kategórií: úplné metódy, aproximačné metódy a kombinované metódy. Metódy, ktoré nebudú výrazne spadať do žiadnej z týchto kategórii označíme ako ďalšie. Práca sa sústreďuje predovšetkým na aproximačné metódy a framework ACLA, do kotrého implementujeme aproximačnú stratégiu rozbitia cyklov. Túto novú stratégiu porovnáme s doterajšou implementáciou. Na záver ukážeme, že pomocou stratégie rozbitia cyklov vieme dosiahnuť lepšie výsledky v zisťovaní nejednoznačnosti bezkontextových gramatík.

**KĽÚČOVÉ SLOVÁ:** detekcia nejednoznačnosti bezkontextových gramatík, regulárna aproximácia bezkontextových gramatík, aproximačná stratégia, metódy detekcie nejednoznačnosti

# Zoznam obrázkov

# Obsah

# Úvod

Bezkontextové gramatiky sú využívané v rôznych oblastiach ako napríklad v softvérovom inžinierstve pre vývoj programovacieho jazyka, spracovanie prirodzeného jazyka, ale aj v oblasti bioinformatiky. Sú vhoné pre definovanie veľkého množstva jazykov, no ich prípadná nejednoznačnosť môže nepriaznivo ovplyvniť ich široké použitie. Preto je stála potreba presnejších a podrobnejších detekčných metód, respektíve schém, ktoré poukazujú na zdroje týchto nejednoznačností v danej gramatike. Nejednoznačnosť bezkontextových gramatík je opakujúci sa problém ako pri navrhovaní jazykov, tak aj pri generovaní parserov a pri využívaní gramatík ako modeloch štruktúr reálneho sveta.
Určiť, či bezkontextová gramatika je nejednoznačná je vo všeobecnosti nerozhodnuteľný problém, avšak existujú metódy, ktoré sa rôznym spôsobom snažia o správnu detekciu, čo najväčšej množiny gramatík. Je snaha tieto metódy zlepšovať, zjemňovať presnosť detekcie a prichádzať na iné vhodné metódy a spôsoby. Nejednoznačnosť je častokrát ťažké odhaliť ručne, preto automatizované nástroje na jej detekciu sú veľmi vítané, ba priam žiadúce.

Metódy detekcie môžu byť rozdelené do troch základných kategórií: úplné, aproximačné a kombinované metódy.
V našej práci taktiež priblížime a popíšeme tieto jednotlivé kategórie techník a ich základné princípy. V súvislosti s týmito metódami poukážeme aj na konkrétne nástroje, ktoré ich používajú, zároveň tieto nástroje stručne charakterizujeme, zhodnotíme a popíšeme.

Cieľom tejto diplomovej práce je spracovanie základného prehľadu metód detekcie nejednoznačnosti bezkontextových gramatík, konkrétnejšie úplných metód, aproximačných metód a kombinovaných metód. Budeme sa snažiť hľadať a poukázať na zlepšenie hlavne pri aproximačnej metóde. Zároveň našou snahou bude implementovať zlepšenie aproximačnej metódy, pričom ho empiricky porovnáme s existujúcou implementáciou a zhodnotíme naše výsledky v porovnaní s existujúcim riešením. Pri implementácii bude použitý programovací jazyk Java.

V 1. kapitole s názvom Prehľad problematiky sa budeme venovať vysvetleniu jednotlivých základných pojmov súvisiacich s touto témou. V 2. kapitole, ktorej názov je Prehľad metód, uvedieme aké sú kategórie metód detekcie a popíšeme teoreticky ich základné princípy. 3. kapitola s pomenovaním Exisujúce nástroje nám priblíži konkrétne nástroje, ktoré využívajú metódy detekcie uvedené v kapitole číslo 2. Témou 4. kapitoly s názvom Implementácia je opis podrobností samotnej implementácie a postupoch pri jej tvorbe. Zároveň v tejto kapitole zhrnieme dosiahnuté výsledky a provnáme ich s doterajším existujúcim riešením.

# 1

# Prehľad problematiky

V úvodnej kapitole tejto práce sa budeme venovať základným pojmom používaným v tejto práci. Zároveň si uvedieme ich definície, aby bola zrejmá notácia, ktorú budeme v práci používať.

## 1.1 Bezkontextová gramatika

**Definícia 1.1.1.** (**Bezkontextová gramatika**)
*Bezkontextová gramatika* je štvorica $G=(N,\Sigma,P,S)$, kde

$N$ je konečná množina *neterminálov*

$\Sigma$ je konečná množina *terminálov*

$P = \{ A \to \alpha \mid A \in N, \alpha \in E^*, E = N \cup \Sigma \}$ je konečná množina *prepisovacích pravidiel*

$S \in N$ je *začiatočný neterminál*

Budeme hovoriť, že $\alpha X \beta \Rightarrow \alpha\theta\beta$ je *krokom odvodenia*, práve vtedy keď $X \to \theta \in P$ a $\alpha,\beta \in E^*$. Symbol $\Rightarrow^*$ nám označuje *reflexívny tranzitívny uzáver* kroku odvodenia $\Rightarrow$.

*Vetnou formou* budeme nazývať reťazec $\alpha \in E^*$.
Jazyk vetnej formy $\alpha \in E^*$ je $\mathcal{L}_G(\alpha) = \{x \in \Sigma^* \mid \alpha \Rightarrow^* x \}$ a jazyk gramatiky $G$ je $\mathcal{L}(G) = \mathcal{L}_G(S)$.
Postupnosť vetných foriem $\alpha \Rightarrow \cdots \Rightarrow \gamma \Rightarrow \cdots \Rightarrow \beta$ budeme nazývať *odvodenie* z $\alpha$ do $\beta$. Takúto postupnosť nám môže určovať aj *strom odvodenia*.
Je to strom, ktorého vrcholy sú označené symbolmi z $E$: koreň je označený symbolom $S$, a synovské vrcholy vrcholu $X$ sú označené symbolmi z vetnej formy $\gamma$, kde pre niektoré pravidlo $X \to \gamma \in P$ [BGM07],[Kru08].

## 1.2 Nejednoznačnosť gramatiky

Bezkontextové gramatiky generujú jazyky pomocou rekurzie a vkladania, čo z nich robí vhodný formalizmus pre opis programovacích jazykov. Avšak nejednoznačnosť je častokrát ich nežiadúca vlastnosť.

**Definícia 1.2.1.** (**Nejednoznačnosť**)
Bezkontextová gramatika $G$ je *nejednoznačná* práve vtedy, ak existuje slovo $x$ v jazyku $\mathcal{L}(G)$, pre ktoré existuje viacero (aspoň 2) rôznych stromov odvodenia. Zároveň hovoríme, že v tomto prípade je $x$ *nejednoznačné* vzhľadom ku gramatike $G$ [BGM07]. V opačnom prípade je gramatika *jednoznačná*.

Táto definícia nám hovorí, že reťazec môže byť analyzovaný (parsovaný) aspoň dvomi rôznymi spôsobmi. V tejto práci si ukážeme rôzne techniky pre detekciu nejednoznačnosti v bezkontextových gramatikách.

**Príklad 1.2.1.** Nasledujúca gramatika je dobre známym príkladom nejednoznačnej gramatiky $G$:

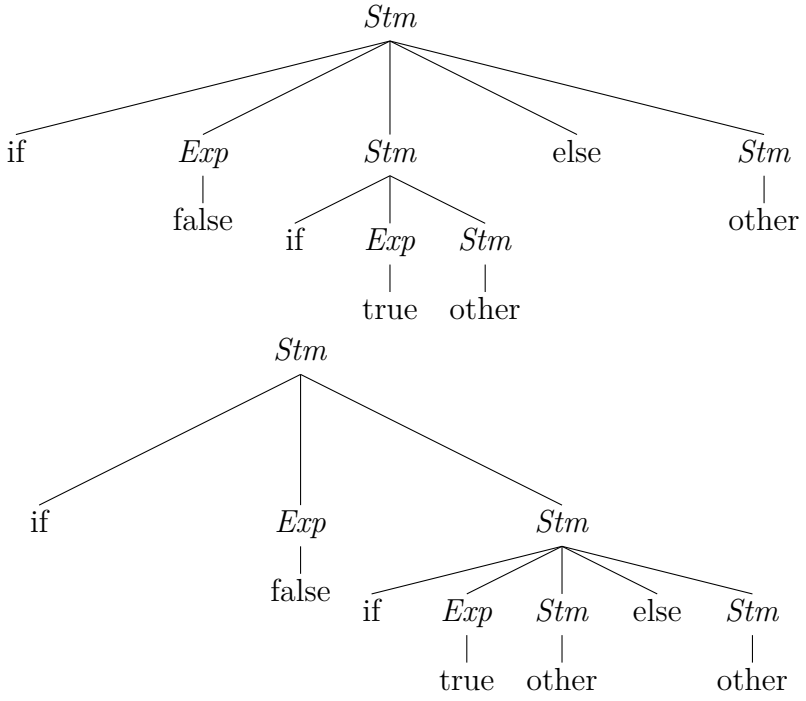
$N = \{Stm, Exp\}$

$\Sigma =$ {if, else, other, true, false }

$Stm$ je počiatočný neterminál

$P =$ { $Stm \rightarrow$ if $Exp$ $Stm$
$Stm \rightarrow$ if $Exp$ $Stm$ else $Stm$
$Stm \rightarrow$ other
$Exp \rightarrow$ true
$Exp \rightarrow$ false }

Ak túto gramatiku použjeme na parsovanie výrazu "if flase if true other else other" tak nie jasné, ktoré pravidlo máme použiť ako prvé ako to vidieť na Obr. 1.1.




Obr. 1.1: Rôzne stromy odvodenia pre výraz "if false if true other else other"

### 1.2.1 Nejednoznačnosť – nerozhodnuteľný problém

**Definícia 1.2.2.** (**Nerozhodnuteľný problém**)
Problém nazývame *nerozhodnuteľný*, ak neexistuje žiadny algoritmus, ktorý preň na každom vstupe skončí s korektnou odpoveďou "áno" alebo "nie" [Poo].

Už od roku 1962 je známe, že problém nejednoznačnosti bezkontextových gramatík

je nerozhodnuteľný, čo ukázali Cantor [Can62], Floyd [Flo62] a Chomsky so Shützenbergom [CS].
To, že problém nejednoznačnosti bezkontextových gramatík je nerozhodnuteľný, znamená, že nie je možné pre všetky bezkontextové gramatiky rozhodnúť, či sú alebo nie sú nejednoznačné, respektíve jednoznačné. Avšak existujú metódy, ktoré sa snažia na čo možno najväčšej množine bezkontextových gramatík túto otázku zodpovedať.
Ak s určitosťou zistíme, že gramatika neobsahuje nejednoznačnosti, tak môžeme prehlásiť, že je jednoznačná. V opačnom prípade, kým nenájdeme príklad nejednoznačného odvodenia, tak nevieme prehlásiť, či gramatika je nejednoznačná.

## 1.3 Ďalšie pojmy

V tejto časti uvediem ďalšie pojmy, s ktorými sa stretneme v našej práci, a ktoré s ňou súvisia.

### 1.3.1 Aproximácia gramatiky

**Definícia 1.3.1.** (**Aproximácia gramatiky**)
*Aproximácia* bezkontextovej gramatiky $G$ je funkcia $\mathcal{A}_G : E^* \to \mathcal{P}(\Sigma^*)$.
Ak $\mathcal{L}_G(\alpha) \subseteq \mathcal{A}_G(\alpha)$ pre každé $\alpha \in E^*$, budeme hovoriť o *hornej aproximácii*.
Ak $\mathcal{L}_G(\alpha) \supseteq \mathcal{A}_G(\alpha)$ pre každé $\alpha \in E^*$, budeme hovoriť o *dolnej aproximácii*.

Inými slovami, $\mathcal{A}_G$ dáva pre každú vetnú formu $\alpha$ aproximáciu pôvodného jazyka vygenerovaného z $\alpha$ [BGM07]. My budeme v tejto práci pracovať s horonou aproximáciou. Čiže pod aproximáciou budeme chápať nadmnožinu pôvodného jazyka, čo znamená že aproximácia obsahuje všetky slová pôvodného jazyka, no môže obsahovať aj nejaké navyše.

### 1.3.2 Transformácia gramatiky

**Definícia 1.3.2.** (**Transformácia gramatiky**)
*Transformáciou gramatiky* budeme rozumieť proces alebo postup, vďaka ktorému sa zápis gramatiky zmení na iný, čím vznikne nová gramatika.

Niektoré transformácie nemenia množinu slov vygenerovaných pôvodnou a transformovanou gamatikou, to znamená, že jazyk generovaný oboma gramatikami je rovnaký (takýto prípad nastane, ak napríklad transformujeme gramatiku do Chomského normálneho tvaru).
Avšak niektoré transformácie zapríčinia, že novovzniknutá gramatika generuje iný jazyk ako pôvodná. Vtedy môžu nastať tri prípady, keď generovaný jazyk je podmnožinou, nadmnožinou pôvodného alebo ani jedno z predošlých.

### 1.3.3 Aproximačná stratégia

**Definícia 1.3.3.** (**Aproximačná stratégia**)
Nech $I = \{G_1, G_2, \dots\}$ je množina bezkontextových gramatík.
Nech $J = \{A_{G_1}, A_{G_2}, \dots\}$ je množina aproximácií príslušných gramatík.
*Aproximačná stratégia* $\mathcal{A} : I \to J$ je funkcia, ktorá vracia aproximáciu gramatiky $\mathcal{A}_G$ danej bezkontextovej gramatiky $G$.

Jednou z možných aproximačných stratégií je stratégia, ktorá vracia konštantú aproximáciu gramatiky $\Sigma^*$. Ako neskôr zistíme, je to príklad aproximačnej stratégie, ktorá v našom kontexte (rozhodovania nejednoznačnosti) nie je veľmi užitočná, lebo pri analýze vyhodnotí, že každá gramatika môže byť nejednoznačná.
Naopak, aproximačná stratégia $\mathcal{A}$, ktorá by pre každú gramatiku $G$ vrátila $\mathcal{A}_G$ také, že $\mathcal{A}_G(\alpha) = \mathcal{L}_G(\alpha)$ pre každé $\alpha$, je stopercentne presná [BGM07].

# 2

# Prehľad metód

V tejto kapitole si priblížime 3 základné skupiny metód, ktoré sa pri detekcii nejednoznačnsti bezkontextových gramatík používajú. Uvedieme ich hlavné princípy a spôsoby ako túto nejednoznačnosť odhaľujú a ich prípadné vylepšenia.

## 2.1 Úplné metódy – Exhaustive methods

Nosným princípom metód, ktoré spadajú do tejto kategórie, je úplné prehľadávanie jazyka vygenerovaného gramatikou s cieľom nájsť nejednoznačné slová. Existujú na to rôzne techniky, ktoré môžeme taktiež nazývať spoločným pomenovaním ako *generovanie vetných foriem alebo výrazov*. Samotné techniky ako aj ich obmeny sú podrobnejšie popísané napríklad v [AHL08], [CU95], [Gor63], [Jam05], [Sch01].

Metódy tohto typu sú naozaj presné, no problémom je, že nie vždy pri detekovaní, respektíve generovaní skončia. A to kvôli tomu, že jazyk, ktorý gramatika generuje, môže byť nekonečný. Ak sme v tomto prípade pri generovaní doposiaľ nenašli nejednoznačnosť, algoritmus pokračuje v generovaí ďalších vetných foriem, no to môže trvať do nekonečna. Takáto situácia v mnohých praktických prípadoch aj nastáva. Jedným z riešní tohto problému môže byť generovanie vetných foriem do nami vopred určenej dĺžky generovaných vetných foriem. Vtedy algoritmus s určitosťou zastaví.

Ak nájdeme nejednoznačnosť do určitej dĺžky, môžeme s určitosťou prehlásiť, že gramatika je nejednoznačná. Avšak, ak sa nejednoznačnosť do danej dĺžky nenájde, môžeme len prehlásiť, že gramatika je jednoznačná iba do danej dĺžky. O nejednoznačnosti / jednoznačnosti celej gramatiky nevieme vyvodiť v tomto prípade žiaden záver.
Úplné metódy produkujú presné a užitočné správy o nejdnoznačnosti. Vedia nájsť presné nejednoznačné slová a ich rôzne stromy odvodenia [Bas11].

## 2.2 Aproximačné metódy

Aproximačné metódy, narozdiel od úplných metód, upustia z presnosti, aby mohli vždy skončiť v konečnom čase. To znamená, že hľadajú nejednoznačnosti nie na pôvodných gramatikách a jazykoch, ale na ich aproximáciách alebo pomocou ich aproximácií.

Do tejto kategórie v súčasnosti patria metódy [Sch07b] a [BGM07], ktoré sa taktiež nazývajú *konzervatívne metódy*. Princíp konzervatívnych metód spočíva v tom, že ak určite vieme, že nejednoznačnosť neexistuje, tak prehlásime gramatiku za jednoznačnú. V opačnom prípade, keď nájdeme nejednoznačnosť, nevieme povedať s istotou, že gramatika je nejednoznačná, lebo pri aproximácii mohli vzniknúť takzvané falošné nejednoznačnosti (nejednoznačnosti, ktoré v pôvodnej gramatiky v skutočnosti nie sú).

V nasledujúcej časti stručne popíšeme RU Test (Regular Unambiguity Test) a NU Test (Noncanonical Unambiguity Test), ktorých autorom je Sylvain Schmitz [Sch07a], [Sch07b] a najväčší dôraz budeme dávať na metódu ACLA (Ambiguity Checking with Language Approximation) od autorov Brabranda, Giegericha a Møllera [BGM07].

### 2.2.1 RU Test – Regular Unambiguity Test

RU test je aproximačný test na existenciu dvoch stromov odvodenia, pričom môže obsahovať aj takzvané falošné nejednoznačnosti, t.j. nejednoznačnosti, ktoré vznikli z dôvodu aproximácie a v pôvodnej gramatike sa nevyskytujú. V tejto časti popíšeme pojmy, na ktorých je RU Test založený a popíšeme si ako detekuje nejednoznačnosti.

#### 2.2.1.1 Uzátvorkovaná gramatika

**Definícia 2.2.1.** (**Uzátvorkovaná gramatika**)
Z gramatiky $G = (N, \Sigma, P, S)$ môže byť zkonštruovaná *uzátvorkovaná gramatika* $G_b$ pridaním unikátnych terminálnych symbolov na začiatok a koniec každého pravidla [GH67]. Uzátvorkovaná gramatika $G_b$ je definovaná ako štvorica $(N, \Sigma_b, P_b, S)$, kde:

$$\Sigma_b = \Sigma \cup \Sigma_\langle \cup \Sigma_\rangle$$

$$\Sigma_\langle = \{\ \langle_i \mid \exists p \in P : i = \mathrm{pid}(p)\ \}$$

$$\Sigma_\rangle = \{\ \rangle_i \mid \exists p \in P : i = \mathrm{pid}(p)\ \}$$

$$P_b = \{\ A \to \langle_i \alpha \rangle_i \mid A \to \alpha \in P, i = \mathrm{pid}(A \to \alpha)\ \}.$$

$\mathrm{pid}(p)$ je jednoznačný identifikátor pravidla $p \in P$. $E_b$ je definovaná ako $\Sigma_b \cup N$. Symbol $\Rightarrow_b$ označuje krok odvodenia, ktorý používa pravidlá z $P_b$.
Homomorfizmus $h : {E_b}^* \to E^*$ mapuje každú *uzátvorkovanú vetnú formu* na prislúchajúcu vetnú formu v pôvodnej gramatike. Je definnovaný nasledovne

$$h(\langle_i) = \epsilon$$
$$h(\rangle_i) = \epsilon$$
$$h(X) = X, \text{ pre } X \in E_b \setminus (\Sigma_\langle \cup \Sigma_\rangle).$$

Vetné formy vygenerované garmatikou $G_b$ sú jednoznačné a to vďaka pridaniu zátovriek $\langle_i$ pred a zátovriek $\rangle_i$ za každé pravidlo pôvodnej gramatiky [Bas11]. Túto

uzátvorkovanú gramatiku budeme používať pri konštruovaní pozičného automatu.

#### 2.2.1.2 Pozičný automat

Základnom RU Testu je *pozičný automat*, ktorý popisuje všetky vetné formy vygenerované uzátvorkovanou gramatikou. Stavy pozičného automatu sú totožné s možnými pozíciami vo vetných formách gramatiky $G_b$, v ktorých sa môžeme nachádzať pri ich odvodzovaní. Prechody medzi jednotlivými stavmi automatu sú označené prvkami z $E_b$.
Gramatika $G$ je nejednoznačná práve vtedy, keď existujú dve cesty automatom (dva vetné formy gramatiky $G_b$), ktoré opisujú rôzne stromy odvodenia jednej vetnej formy v $G$. Takúto dvojicu ciest nazývame *nejednoznačná dvojica ciest* [Sch07a], [Bas11].

#### 2.2.1.3 Aproximovaný pozičný automat

Ak má gramatika $G$ nekonečný počet stromov odvodenia, pozičný automat je nekonečnej veľkosti. Vtedy hľadanie dvojíc ciest nejednoznažnosti bude trvať do nekonečna. Preto pozičný automat spomínaný vyššie musíme aproximovať, kedy využijeme ekvivalentné vzťahy medzi jednotlivými pozíciami.
Nový, aproximovaný pozičný automat má stavy, ktoré zodpovedajú jednotlivým triedam ekvivalencie pôvodného pozičného automatu. Pre každý prechod medzi dvoma stavmi v pôvodnom automate, ktorý má svoje označenie, je v aproximovanom automate vytvorený prechod s rovnakým označením a to medzi triedami ekvivalencie (stavmi v aproximovanom automate), v ktorých sa nachádzajú stavy pôvodného automatu.
Ak použijeme vzťah ekvivalencie medzi stavmi pôvodného automatu, tak získame konečný počet tried ekvivalencie, čiže konečný počet stavov aproximovaného automatu. V takto aproximovanom automate môžeme teraz hľadať dvojice ciest nejednoznačnosti v konečnom čase [Bas11].

#### 2.2.1.4 Párový pozičný automat

Existenciu dvojíc ciest nejednoznačnosti v pozičnom automate môžeme zistiť pomocou *párového pozičného automatu*. V ňom je každý stav vlastne dvojicou stavov z pozičného automatu. Prechody medzi jednotlivými stavmi sú určené špeciálnymi vzťahmi popísanými v [Bas11].
Každá cesta z počiatočného do koncového stavu v tomto párovom pozičnom automate nám vlastne popisuje dve cesty v pozičnom automate (aproximovanom pozičnom automate). Jazykom, ktorý generuje párový automat je množina dvojíc slov. Cesta nám indikuje dvojicu ciest nejednoznačnosti ak jej dve zátvorkované slová sú rôzne, ale rovnaké pri homomorfizme $h$.
Keďže aproximovaný pozičný automat je horná aproximácia (môže generovať aj niečo naviac oproti pôvodnému automatu), tak párový pozičný automat obsahuje aspoň všetky dvojice ciest nejednoznačnosti pôvodného pozičného automatu.
Z tohto vieme vyvodiť, že ak neexistuje nejdnoznačná dvojica ciest v párovom pozičnom automate, tak neexistuje ani v pôvodnom párovom automate, čo hovorí o jednoznačnosti pôvodnej gramatiky $G$.
Čiže ak nenájdeme nejednoznačnosť, vieme určiť výsledok. No ak zistíme prítomnosť potenciálnej nejednoznačnosti, tak nevieme s určitosťou povedať, či je naozaj nejednoznačná a to kvôli možným falošným nejednoznačnostiam [Bas11].

### 2.2.2 NU Test – Noncanonical Unambiguity Test

V tejto časti popíšeme NU Test [Sch07b], ktorý je precíznejší než RU Test [Bas11]. Umožňuje identifikovať širšiu množinu irelevantných stromov odvodenia, presnejšie tie, ktoré nepatria do jadra nejednoznačnosti gramatiky $G$.

#### 2.2.2.1 Jadro nejednoznačnosti

**Definícia 2.2.2.** (**Jadro nejednoznačnosti**)
Množinu najmenších možných nejednoznačných vetných foriem gramatiky $G$ budeme

nazývať *jadro nejednoznačnosti* a označovať $C^a(G)$.
Sú to vlasnte nejednoznačné vetné formy, ktoré nemôžu byť odvodené z inej vetnej formy, ktorá je nejednoznačná. Ich stromy odvodenia sú najmenší indikátor nejednoznačnosti v $G$.

Pozičný párový automat pri RU Teste kontroloval všetky dvojice ciest na prítomnosť nejednoznačnosti. Avšak nie všetky je nutné kontrolovať na identifikovanie nejednoznačnosti v gramatike.
Tie, ktoré nie je potrebné kontrolovať, sú dvojice ciest odvodzujúce rovnaké jednoznačné podslová pre istý neterminál ako predošlá dvojica ciest. Tieto cesty môžeme úplne ignorovať. Pre presnejšie pochopenie, o aké dvojice ciest ide, a ktoré dvojice ciest môžeme pri detekovaní vynechať, uvádzame nasledujúci príklad:

**Príklad 2.2.1.** Uvažujme cesty $\langle_1\langle_2\langle_3 a\rangle_3\alpha_b\rangle_2\rangle_1$ a $\langle_1\langle_2\langle_3 a\rangle_3\beta_b\rangle_2\rangle_1$. Ak tieto tvoria dvojicu ciest v párovom pozičnom automate, tak potom aj kratšie cesty $\langle_1\langle_2 A\alpha_b\rangle_2\rangle_1$ a $\langle_1\langle_2 A\beta_b\rangle_2\rangle_1$ (predpokladáme, že $A \to \langle_3 a\rangle_3 \in P_b$) tvorili dvojicu ciest v párovom pozičnom automate. Ak prvá dvojica ciest tvorí nejednoznačnú dvojicu ciest, potom aj druhá dvojica, lebo $\langle_3 a\rangle_3$ neprispieva k nejednoznačnosti (je to jednoznačné podlsovo). V tomto prípade budeme teda preferovať druhú dvojicu, lebo ona má menší strom odvodenia, čo vlastne chceme dosiahnuť [Bas11].

#### 2.2.2.2 Nekánonický párový pozičný automat

Nekánonický párový pozičný automat (NPPA) je úprava párového pozičného automatu z RU Testu (budeme označovať RPPA).
K dvojiciam stavov sa pridajú špeciálne booleovké príznaky $c_0$ a $c_1$, ktoré nám hovoria pre každý stav v páre, či jeho cesta je v konflikte s cestou druhého stavu. Booleovské príznaky nám spôsobia, že sa nebudú kontrolovať zbytočné dvojice ciest. Podrobnejší popis tohto automatu nájdeme v [Sch07b] a [Bas11].
Tak ako RPPA aj NPPA opisuje nejednoznačné cesty aproximovaného pozičného automatu. Rozdiel je iba v tom, že jazyk, ktorý vygeneruje NPPA neobsahuje dvojice

ciest bez konfliktov. Preto $\mathcal{L}(\text{NPPA}) \subseteq \mathcal{L}(\text{RPPA})$.
Zároveň NPPA opisuje určite aspoň všetky stromy odvodenia vetných foriem z jadra nejednoznačnosti gramatiky $G$.

### 2.2.3 ACLA – Ambiguity Checking with Language Approximation

Ďalšou metódou, ktorá patrí do skupiny aproximačných metód je metóda ACLA – zisťovanie nejednoznačnosti pomocou aproximovania jazyka [BGM07].
Namiesto toho, aby sa táto metóda zaoberala problémom nejednoznačnosti ako celkom, rozdelí ho na dva menšie podproblémy nazývané *vertikálna* a *hotizontálna* nejednoznačnosť. Budeme ich označovať aj ako *čiastočné nejednoznačnosti*.
Najprv si definujeme horizontálnu a vertikálnu nejednoznačnosť, potom si ukážeme ako aproximácia pomáha pri detekovaní nejednoznačnosti, a nakoniec ako aproximáciu vytvoriť.

**Definícia 2.2.3.** (**Prekryv** $\boxtimes$)
Prekryv dvoch jayzkov $X$ a $Y$ je definovaný nasledovne:

$$X \boxtimes Y = \{\ xay \mid x, y \in \Sigma^* \wedge a \in \Sigma^+ \wedge x, xa \in X \wedge y, ay \in Y\ \}$$

Prekryv je vlasnte množina slov z $XY$, ktoré nevedia byť rozdelené unikátne na časť prislúchajúcu množine $X$ a časť prislúchajúcu množine $Y$. Uvedieme príklad:

**Príklad 2.2.2.** Nech $X = \{x, xo\}$ a $Y = \{y, oy\}$, potom $X \boxtimes Y = \{xoy\}$

#### 2.2.3.1 Horizontálna a vertikálna nejednoznačnosť

**Definícia 2.2.4.** (**Horizontálna nejednoznačnosť**)
Gramatika $G$ je *horizontálne nejednoznačná* práve vtedy, ak

$$\exists A \in N,\ \alpha,\beta \in E^*,\ (A \to \alpha\beta) \in P : \mathcal{L}_G(\alpha) \boxtimes \mathcal{L}_G(\beta) \neq \emptyset$$

V opačnom prípade je gramatika *horizontálne jednoznačná.*

To znamená, že gramatika $G$ je horizontálne nejednoznačná práve vtedy, ak existuje v $G$ pravidlo $A \to \alpha\beta$, pre ktoré majú jazyky $\mathcal{L}_G(\alpha)$ a $\mathcal{L}_G(\beta)$ neprázdny prekryv (čiže exituje postupnosť terminálov, ktorými končí slovo z $\mathcal{L}_G(\alpha)$ a začína slovo z $\mathcal{L}_G(\beta)$ ).

**Definícia 2.2.5.** (**Vetikálna nejednoznačnosť**)
Gramatika $G$ je *vertikálne nejednoznačná* práve vtedy, ak

$$\exists A \in N,\ \alpha,\beta \in E^*,\ (A \to \alpha),(A \to \beta) \in P,\ \alpha \neq \beta : \mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\beta) \neq \emptyset$$

V opačnom prípade je gramatika *vertikálne jednoznačná.*

Gramatika je teda vertikálne nejednoznačná práve vtedy, ak má dve rôzne pravidlá $A \to \alpha$, $A \to \beta$, ktoré môžu odvodiť to isté slovo.

**Tvrdenie 2.2.1.** (Horizontálna nejednoznačnosť $\Rightarrow$ nejednoznačnosť)
Ak je gramatika $G$ horizontálne nejednoznačná, potom je nejednoznačná aj podľa definície 1.2.1.

*Dôkaz.* [Kru08], [BGM07] Nech máme danú gramatiku $G$, ktorá je horizontálne nejednoznačná kvôli pravidlu $A \to \alpha\beta$ ($\mathcal{L}_G(\alpha) \between \mathcal{L}_G(\beta) \neq \emptyset$). Nech $w \in \mathcal{L}_G(\alpha) \between \mathcal{L}_G(\beta)$. Z definície 2.2.3 $w = xvy$, kde $|v| > 0$, $x$, $xv \in \mathcal{L}_G(\alpha)$ a $y$, $vy \in \mathcal{L}_G(\beta)$. Prepokladáme, že každý neterminál je odvoditeľný z počiatočného neterminálu $S$ a odvodí neprázdnu množinu slov. A keďže prekryv je neprázdny, môžeme skonštruovať dva stromy odvodenia pre slovo $pxvyq$, pre nejaké $p, q \in \Sigma^*$, zodpovedajúce nasledujúcim odvodeniam:
$S \Rightarrow^* pAq \Rightarrow p\alpha\beta q \Rightarrow^* px\beta q \Rightarrow^* pxvyq$
$S \Rightarrow^* pAq \Rightarrow p\alpha\beta q \Rightarrow^* pxv\beta q \Rightarrow^* pxvyq$

□

**Tvrdenie 2.2.2.** (Vetikálna nejednoznačnosť $\Rightarrow$ nejednoznačnosť)
Ak je gramatika $G$ vertikálne nejednoznačná, potom je nejednoznačná aj podľa definície 1.2.1.

*Dôkaz.* [Kru08], [BGM07] Nech máme danú gramatiku $G$, ktorá je vertikálne nejednoznačná kvôli pravidlám $A \to \alpha$ a $A \to \beta$ ($\mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\beta) \neq \emptyset$). Nech $w \in \mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\beta)$, $w \in \Sigma^*$, t.z. $\alpha \Rightarrow^* w$ a $\beta \Rightarrow^* w$. Podobne ako v predošlom tvrdení môžeme skonštruovať dva stromy odvodenia pre slovo $pwq$ pre nejaké $p, q \in \Sigma^*$ zodpovedajúce nasledujúcim odvodeniam:
$S \Rightarrow^* pAq \Rightarrow p\alpha q \Rightarrow^* pwq$
$S \Rightarrow^* pAq \Rightarrow p\beta q \Rightarrow^* pwq$

□

**Tvrdenie 2.2.3.** (Nejednoznačnosť ⇒ horizontálna ALEBO vertikálna nejednoznačnosť)
Ak je gramatika $G$ nejednonzačná podľa definície 1.2.1, potom je buď horizontálne alebo vertikálne nejednoznačná (alebo oboje).

*Dôkaz.* Dôkaz je založený na matematickej indukcii vzhľadom na výšku stromov odvodenia nejednoznačného slova $\omega \in \mathcal{L}(G)$, pre ktoré existujú apoň dva takéto stromy odvodenia. Matematickou indukciu zistíme, že nejednoznačná gramatika je buď horizontálne alebo vertikálne nejednoznačná (alebo oboje). Pre rozsiahlosť celý dôkaz neuvádzame, ale je popísaný v [Kru08], [BGM07]. □

Z tvrdenia 2.2.3 ľahko vyplýva fakt, že ak zistíme o gramatike $G$, že je vertikálne a súčasne aj horizontálne jednoznačná, tak ju môžeme s určitosťou prehlásiť za jednoznačnú, čo nám vraví aj nasledujúce tvrdenie:

**Tvrdenie 2.2.4.** (Jednoznačnosť ⇔ horizontálna a vertikálna jednoznačnosť)
Gramatika $G$ je jednonzačná práve vtedy, keď je horizontálne a súčasne vertikálne jednoznačná.

*Dôkaz.* Toto tvrdenie vyplýva jednoducho z obmeny tvrdenia 2.2.3 a tvrdení 2.2.2, 2.2.1. □

#### 2.2.3.2 Aproximovaná horizontálna a vertikálna nejednoznačnosť

V tejto časti si priblížime aproximovanú horizontálnu a vertikálnu nejednoznačnosť, pri ktorých budeme využívať aproximáciu gramatiky $G$ ako ju máme danú v definícii 1.3.1. Aproximovanú nejednoznačnosť potrebujeme kvôli tomu, že vďaka nej budeme rozhodovať o jednoznačnosti, respektíve nejednoznačnosti gramatiky.

**Definícia 2.2.6.** (**Aproximovaná horizontálna nejednoznačnosť**)
Nech je daná bezkontextová gramatika $G = (N,\Sigma,P,S)$ a aproximácia gramatiky $\mathcal{A}_G(\gamma) \supseteq \mathcal{L}_G(\gamma)$, pre každé $\gamma \in E^*$. Gramatika $G$ je *horizontálne nejednoznačná vzhľadom na aproximáciu* $\mathcal{A}_G$ práve vtedy, keď:

$$\exists A \in N,\ \alpha,\beta \in E^*,\ (A \to \alpha\beta) \in P : \mathcal{A}_G(\alpha) \between \mathcal{A}_G(\beta) \neq \emptyset$$

V opačnom prípade je gramatika $G$ *horizontálne jednoznačná vzhľadom na aproximáciu* $\mathcal{A}_G$.

**Definícia 2.2.7.** (**Aproximovaná vertikálna nejednoznačnosť**)
Nech je daná bezkontextová gramatika $G = (N,\Sigma,P,S)$ a aproximácia gramatiky $\mathcal{A}_G(\gamma) \supseteq \mathcal{L}_G(\gamma)$, pre každé $\gamma \in E^*$. Gramatika $G$ je *vertikálne nejednoznačná vzhľadom na aproximáciu* $\mathcal{A}_G$ práve vtedy, keď:

$$\exists A \in N,\ \alpha,\beta \in E^*,\ (A \to \alpha),(A \to \beta) \in P,\ \alpha \neq \beta : \mathcal{A}_G(\alpha) \cap \mathcal{A}_G(\beta) \neq \emptyset$$

V opačnom prípade je gramatika $G$ *vertikálne jednoznačná vzhľadom na aproximáciu* $\mathcal{A}_G$.

Tieto aproximované nejednoznačnosti budeme taktiež nazývať *potenciálne vertikálne* a *horizontálne nejednoznačnosti* ako sú označované v [Kru08]. Je to kvôli tomu, že prieniky a prekryvy jednotlivých aproximácií môžu vygenerovať aj takzvané falošné nejednoznačnosti, ktoré v pôvodnej gramatike nemusia vôbec existovať. To je vidieť z nasledujúceho tvrdenia:

**Tvrdenie 2.2.5.** (Nerovnosť nadmnožín)
Nech $A$ a $B$ sú množiny terminálnych slov a $A' \supseteq A$, $B' \supseteq B$ sú ich nadmnožiny. Potom

$$A' \cap B' \supseteq A \cap B$$
$$A' \between B' \supseteq A \between B$$

*Dôkaz.* Dôkaz tohto tvrdenia nie je zložitý, dá sa povedať, že je intuitívny. My ho neuvádzame, ale môžeme ho nájsť v [Kru08].

□

Ďalej si ukážeme ako súvisia spolu čiastočné nejednoznačnosti s aproximovanými čiastočnými nejednoznačnosťami.

**Tvrdenie 2.2.6.** (Horizontálna nejednoznačnosť ⇒ aproximovaná horizontálna nejednoznačnosť)
Nech je daná bezkontextová gramatika $G = (N,\Sigma,P,S)$ ktorá je horizontálne nejednoznačná. Potom je $G$ horizontálne nejednoznačná vzhľadom na všetky aproximácie $\mathcal{A}_G$ (má potenciálnu horizontálnu nejednoznačnosť pre všetky aproximácie).

*Dôkaz.* [Kru08] Nech $A \to \alpha\beta \in P$ je pravidlo, ktoré zapríčiňuje horizontálnu nejednoznačnosť v $G$, t.j. $\mathcal{L}_G(\alpha) \between \mathcal{L}_G(\beta) \neq \emptyset$. Z toho vyplýva $\mathcal{A}_G(\alpha) \between \mathcal{A}_G(\beta) \neq \emptyset$ podľa tvrdenia 2.2.5 a preto $A \to \alpha\beta$ spĺňa definíciu 2.2.6 o aproximovanej horizonatálnej nejednoznačnosti gramatiky. □

**Tvrdenie 2.2.7.** (Vertikálna nejednoznačnosť ⇒ aproximovaná vertikálna nejednoznačnosť)
Nech je daná bezkontextová gramatika $G = (N,\Sigma,P,S)$ ktorá je vertikálne nejednoznačná. Potom je $G$ vertikálne nejednoznačná vzhľadom na všetky aproximácie $\mathcal{A}_G$ (má potenciálnu vertikálnu nejednoznačnosť pre všetky aproximácie).

*Dôkaz.* [Kru08] Nech $A \in N$ je neterminál, ktorý zapríčiňuje vertikálnu nejednoznačnosť v gramatike $G$, t.j. existujú dve pravidlá $A \to \alpha$, $A \to \beta \in P$ také, že $w \in \mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\beta)$. Potom podľa tvrdenia 2.2.5 $w \in \mathcal{A}_G(\alpha) \cap \mathcal{A}_G(\beta)$, takže tento prienik nie je prázdny. Podľa definície 2.2.7 je gramatika aproximovane vertikálne

nejednoznačná.

□

Lahko z týchto tvrdení vidieť, že ak nenájdeme žiadne aproximované vertikálne a horizontálne nejednoznačnosti (t.j. potenciálne nejednoznačnosti), je gramatika $G$ jednoznačná, čo hovorí aj nasledujúce tvrdenie:

**Tvrdenie 2.2.8.** Ak je $G$ aproximovane (potenciálne) horizontálne a vertikálne jednoznačná, potom je $G$ jednoznačná [BGM07].

*Dôkaz.* [BGM07] Dôkaz tohto tvrdenia jednoducho vyplýva z predchádzajúcich tvrdení.

□

**Definícia 2.2.8.** Vtedy, keď pre $\forall\alpha,\beta \in \mathcal{L}(G)$ vieme rozhodnúť, či množiny $\mathcal{A}_G(\alpha) \cap \mathcal{A}_G(\beta)$ a $\mathcal{A}_G(\alpha) \between \mathcal{A}_G(\beta)$ sú prázdne alebo nie, hovoríme o *konzervatívnej aproximácii* pre problém nejednoznačnosti.

Ešte je vhodné si uvedomiť, že predošlé tvrdenia, nám otvárajú priestor, pre rôzne prístupy vytvorenia aproximácií pre zistenie nejednoznačnosti, respektíve jednoznačnosti gramtiky $G$.
Môžeme vytvoriť *hornú* ($\mathcal{A}_G \supseteq \mathcal{L}_G$) alebo *dolnú aproximáciu* ($\mathcal{A}_G \subseteq \mathcal{L}_G$). Pri oboch je zaujímavé sledovať detekovanie nejednoznačnosti. Avšak, my sa v práci budeme zaoberať len hornou aproximáciou, kde ak zistíme jednoznačnosť, vieme s určitosťou povedať, že aj pôvodná gramatika bola jednoznačná. No v tomto prípade nevieme vyvodiť záver pri nejednoznačnosti. Niektoré z možností ako vytvoriť rôzne aproximácie môžme nájsť napríklad v [Ned00].

#### 2.2.3.3 Kombinovanie aproximácií

Ak máme dve rôzne aproximácie (horné) danej gramatiky $\mathcal{A}_G$ a $\mathcal{A}'_G$, môžeme ich pokojne kombinovať. Nová funkcia $\mathcal{A}''_G$ bude definovaná ako $\mathcal{A}''_G(\alpha) = \mathcal{A}_G(\alpha) \cap$

$\mathcal{A}'_G(\alpha)$ je tiež hornou aproximáciou gamatiky, ktorá akoby zahŕňa obe. Takáto kombinácia môže viesť k vyššej presnosti než spraviť dve aproximácie nezávisle, keďže jedna z aproximácií môže byť dobrá v jednej časti pôvodnej gramatiky a druhá v inej časti.

Doteraz sme vo väčšine definícii používali nejednoznačnosť, no v nasledujúcom odseku sa nám bude lepšie pracovať s jednoznačnosťou.

**Definícia 2.2.9.** (**Aproximovaná horizontálna a vertikálna jednoznačnosť vetných foriem**)
Nech je daná aproximácia gramatiky $\mathcal{A}_G$ a dve vetné formy $\alpha,\beta \in E^*$.
Vetné formy $\alpha$ a $\beta$ sú *horizontálne jednoznačné v závislosti k* $\mathcal{A}_G$, práve vtedy ak

$$\mathcal{A}_G(\alpha) \between \mathcal{A}_G(\beta) = \emptyset, \qquad \text{ozn. } \vDash_{\mathcal{A}_G} \alpha,\beta.$$

Vetné formy $\alpha$ a $\beta$ sú *vertikálne jednoznačné v závislosti k* $\mathcal{A}_G$, práve vtedy ak

$$\mathcal{A}_G(\alpha) \cap \mathcal{A}_G(\beta) = \emptyset, \qquad \text{ozn. } \Vdash_{\mathcal{A}_G} \alpha,\beta.$$

Aproximácie gramatiky môžu byť ešte kombinované ďalším spôsobom, ktorý je v praxi užitočný:

**Tvrdenie 2.2.9.** (Kombinovanie aproximácií)
Nech je daná množina aproximácií gramatiky $\mathcal{A}^1_G,\ldots,\mathcal{A}^k_G$.
Ak $\forall A \in N$, $(A \to \alpha\beta) \in P$ kde $\alpha,\beta \in E^*$ :

$$\exists j \in \{\ 1,\ldots,\ k\ \} : \vDash_{\mathcal{A}^j_G} \alpha,\beta,$$

potom je $G$ horizontálne jednoznačná ($\vDash G$).

Ak $\forall A \in N$, $(A \to \alpha),(A \to \beta) \in P$ kde $\alpha \neq \beta$ :

$$\exists j \in \{\ 1,\ldots,\ k\ \} : \Vdash_{\mathcal{A}^j_G} \alpha,\beta,$$

potom je $G$ vertikálne jednoznačná ($\Vdash G$).

*Dôkaz.* [BGM07] Dôkaz je iba jednoduchou obmenou dôkazu tvrdenia 2.2.8. Intuitívne, gramatika je jednoznačná, ak zistíme vertikálnu a horizontálnu jednoznačnosť všetkých vetných foriem pravých strán pravidiel, pričom môžeme použiť rôzne aproximácie. □

#### 2.2.3.4 Aproximácie pomocou regulárnych gramatík

Ako sme už vyššie spomínali, aproximácie používané pri detekovaní nejednoznačnosti môžu byť rôzne konštruované. Potrebujeme takú aproximáciu, s ktorou sa bude dobre pracovať, a ktorú vieme rozumne skonštruovať. Preto veľmi vhodnou aproximáciu je aproximácia pomocou regulárnych gramatík.
Regulárne gramatiky umožňujú počítanie prienikov a prekryvov spomínaných vyššie, a súčasne vedia byť aj hornou aproximáciou, čo zabezpečuje fakt, že $\Sigma^*$ je regulárny jazyk a nadmnožinou každého jazyka. Preto sa aj v tejto práci budeme zaoberať regulárnou aproximáciou jazyka.

Teraz načrtneme akým spôsobom môžeme počítať prieniky a prekryvy regulárnych jazykov.
Keďže regulárne gramatiky sú zvyčajne reprezentované *konečným stavovým automatom* (ozn. $KA$), budeme využívať ten. O tomto modeli sa viac môžeme dozvedieť napríklad v [RF13].

Prienik dvoch regulárnych jazykov môžeme vďaka konečným stavovým automatom počítať v kvadratickom čase v závislosti od počtu stavov obidvoch automatov. Postup akým to docielime je popísaný v [HMU06].
Prekryv je ale o čosi komplikovanejšia operácia.
Nasledujúce tvrdenie nám naznačuje, ako ho môžeme implementovať:

**Tvrdenie 2.2.10.** (Operácia prekryvu pre regulárne jazyky)
Nech $KA_1$ a $KA_2$ sú dva konečné stavové automaty. Nech $\Sigma$ je množina symbolov používaných v $KA_1$ a $KA_2$. Zavedieme dva nové terminálne symboly $<XV>$ a $<VY>$, ktoré sa nenachádzajú v $\Sigma$.
Potom $\mathcal{L}(KA_1) \between \mathcal{L}(KA_2)$ je rovný prieniku nasledovných troch množín:

$$\{\ x_1{<}VY{>}y_1 \mid x_1 \in \mathcal{L}(KA_1 \text{ rozšírený o } {<}XV{>}),\ y_1 \in \mathcal{L}(KA_2)\ \}$$
$$\{\ x_2{<}XV{>}y_2 \mid x_2 \in \mathcal{L}(KA_1),\ y_2 \in \mathcal{L}(KA_2 \text{ rozšírený o } {<}VY{>})\ \}$$
$$\{\ x_3{<}XV{>}v_3{<}VY{>}y_3 \mid x_3,y_3 \in \Sigma^*,\ v_3 \in \Sigma^+\ \} \quad ,$$

v ktorom budú terminály $<XV>$ a $<VY>$ vymazané.

*Dôkaz.* Dôkaz tohto tvrdenia nebudme uvádzať, lebo je popísaný v [Kru08]. □

Označenie $\mathcal{L}(KA$ rozšírený o $t)$ určuje zmenu jazyka rozpoznávaného automatom $\mathcal{L}(KA)$, a to takú, že $\mathcal{L}(KA$ rozšírený o $t)$ zahŕňa všetky slová, kde ľubovoľný počet terminálov $t$ je vložených na ľubovoľnú pozíciu do slova $w \in \mathcal{L}(KA)$.

### 2.2.4 Aproximačné stratégie pre metódu ACLA

V tejto časti budeme pokračovať v metóde ACLA a to tak, že si priblížime rôzne aproximačné stratégie, vďaka ktorým môžeme detekovať možné (horizontálne a vertikálne) nejednoznačnosti. Pre prehľad základných aproximačných metód pozri predošlú časť.

#### 2.2.4.1 EmptyString – aproximačná stratégia

Ako prvú spomenieme stratégiu prázdneho slova (*EmptyString*), ktorá je založená na odvodení prázdneho slova ($\epsilon$) z vetnej formy $\alpha$ a používa sa pri detekovaní vertikálnej nejednoznačnosti.*EmptyString* je aproximačná stratégia, ktorá vráti pre gramatiku $G$ aproximáciu gramatiky $EmptyString_G$ danú predpisom:

$$EmptyString_G(\alpha) = \begin{cases} \Sigma^0 & \text{ak } \alpha = \epsilon \\ \Sigma^+ & \text{ak } \alpha \Rightarrow^* \epsilon \\ \Sigma^* & \text{inak} \end{cases}$$

Ak nastane prvý prípad, t.j. vetná forma $\alpha$ je prázdne slovo, stratégia odvodí jazyk $\Sigma^0 = \{\epsilon\}$, ktorý je vlastne disjunktný s druhou možnosťou. Tá nastane vtedy, ak

z $\alpha$ nevieme odvodiť prázde slovo. Prvý prípad môžeme zistiť triviálne, na detekciu druhého môžeme použiť algoritmus z [HU79]. Tretí prípad sa použije vtedy, ak nevieme o vetnej forme zistiť prvé dva prípady. Táto stratégia je účinná, ak z dvojice analyzovaných pravidiel (vetných foriem) jedno odvodí iba prázdne slovo a druhé ho nedokáže odvodiť (čiže ak nastanú prvé dva prípady), lebo vtedy je prienik prázdny a nie je zistená nejednoznačnosť, čiže dané pravidlá sú vertikálne jednoznačné [BGM07], čo si uložíme na nasledujúcom príklade:

**Príklad 2.2.3.** Nech máme gramatiku G:

$S \rightarrow$ a $S$ a
$S \rightarrow \epsilon$

Potom $EmptyString_G$(a $S$ a) $= \Sigma^+$ a $EmptyString_G(\epsilon) = \Sigma^0$, čo sú navzájom disjunktné množiny. Teda $EmptyString_G$(a $S$ a) $\cap$ $EmptyString_G(\epsilon) = \emptyset$ a nie je zistená vertikálna nejednoznačnosť, takže vieme, že pravidlá sú vertikálne jednoznačné a môžeme pokračovať v detekcii ďalších pravidiel.

#### 2.2.4.2 MayMust – aproximačná stratégia

MayMust aproximačná stratégia je založená na vytvorení množín terminálov, ktoré sa môžu (May), a ktoré sa musia (Must) vyskytovať v odvodenom slove.
Množina $MAY_G(\alpha) \subseteq \Sigma$ je množina terminálov zo $\Sigma$, ktoré sa vyskytujú v nejakom slove odvodenom z $\alpha$.
Naopak, množina $MUST_G(\alpha) \subseteq \Sigma$ je množina terminálov z $\Sigma$, ktoré sa vyskytujú v každom slove odvodenom z $\alpha$.

$MayMust$ aproximačná stratégia nám vráti aproximáciu gramatiky, ktorú definujeme nasledovne:
$MayMust_G(\alpha) =$
$\{\ \sigma \in (MAY_G(\alpha))^* \mid \exists m : \sigma = \sigma_1 \cdots \sigma_m, \forall \delta \in MUST_G(\alpha) : \exists i \in \{1,\ldots,m\} : \sigma_i = \delta\ \}$

Obidve množiny $MAY_G(\alpha)$ a $MUST_G(\alpha)$ môžu byť vypočítané pre každé $\alpha$ vyskytujúce sa v $G$ v lineárnom čase v závislosti od veľkosti $G$. Zároveň $MayMust_G(\alpha)$ môže byť účinne reprezentovaná ako dvojica ($MAY_G(\alpha)$, $MUST_G(\alpha)$).
Táto aproximácia môže zistiť vertikálnu jednoznačnosť dvoch pravidiel (vetných foriem) $\alpha$ a $\alpha$', keď každé slovo odvoditeľné z $\alpha$ obsahuje terminál, ktorý sa nikdy nevyskytne v slovách odvoditeľných z $\alpha$', a naopak [BGM07].

**Príklad 2.2.4.** Nech máme gramatiku G:

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow$ x
$F \rightarrow$ ( y )

Pozrieme sa na pravidlá pre neterminál $E$:
$MUST_G(E + T) = \{+\}$ $\quad$ $MAY_G(E + T) = \{+, *, \text{x}, (, ), \text{y}\}$
$MUST_G(T) = \emptyset$ $\quad$ $MAY_G(T) = \{*, \text{x}, (, ), \text{y}\}$
Potom
$MayMust_G(E + T) = \{\alpha+\beta \mid \alpha,\beta \in (MAY_G(E + T))^*\}$ a
$MayMust_G(T) = \emptyset$.
Odtiaľ dostávame, že $MayMust_G(E + T) \cap MayMust_G(T) = \emptyset$, takže v tomto prípade nie je zistená žiadna vertikálna nejednoznačnosť, čo znamená, že pravidlá sú vertikálne jednoznačné.

#### 2.2.4.3 FirstLast – aproximačná stratégia

Aproximačná stratégia $FirstLast$ je oproti dvom predošlým stratégiám účinnejšia pri zisťovaní horizontálnej nejednoznačnosti. $FirstLast$ nám vráti aproximáciu gramatiky definovanú nasledovne:

$$FirstLast_G(\alpha) = (MAY_G(\alpha))^* \cap FIRST_G(\alpha)\Sigma^* \cap \Sigma^* LAST_G(\alpha) \quad .$$

Množinu $FIRST_G$ budeme počítať a chápať ako v [AU72]. Množina $LAST_G$ je počítaná ako $FIRST_G$, ale s otočenými pravými stranami pravidiel.
Pri vertikálnej nejednoznačnosti je táto stratégia účinná pre dvojicu pravidiel, ktoré majú disjunktné množiny $FIRST$ alebo množiny $LAST$, vďaka čomu zistíme, že kontolované pravidlá sú vertikálne jednoznačné.
Pri horizontálnej nejednoznačnosti táto stratégia využíva fakt, že prekryv dvoch jazykov $A$ a $B$ ($A \bowtie B$) je prázdny (čo značí horizontálnu jednoznačnosť pravidla), ak sa first-symboly (symboly z množin $FIRST$) slov z množiny $B$ nevyskytujú vobec v slovách množiny $A$. Alebo naopak, že last-symboly (symboly vyskytujúce sa v množine $LAST$) slov množiny $A$ sa nevyskytujú vôbec v slovách z množiny $B$ [BGM07].

**Príklad 2.2.5.** Nech máme gramatiku G:

$S \rightarrow B$
$S \rightarrow$ a b $B$
$A \rightarrow$ c d c
$B \rightarrow A$
$B \rightarrow B$ b $A$

Detekcia vertikálnej nejednoznačnosti pre pravidlá neterminálu $S$:
$FIRST_G(B) = \{\text{c}\}$
$FIRST_G(\text{a b } B) = \{\text{a}\}$
Množiny sú disjunktné preto $FirstLast_G(B) \cap FirstLast_G(\text{a b } B) = \emptyset$, čo znamená, že pri týchto pravidlách nebola zistená žiadna vertikálna nejednoznačnosť.

Detekcia horizontálnej nejednoznačnosti pre pravidlo $S \rightarrow$ a b $B$ pri rozdelení pravej strany na dva terminály a jeden neterminál:
$MAY_G(\text{a b}) = \{\text{a,b}\}$
$FIRST_G(B) = \{\text{c}\}$
Množiny sú tiež disjunktné preto $FirstLast_G(\text{a b}) \bowtie FirstLast_G(B) = \emptyset$, čo znamená, že pri tomto pravidle pri danom rozdelení pravej strany nebola zistená žiadna horizontálna nejednoznačnosť.

#### 2.2.4.4 Mohri-Nederhof – aproximačná stratégia

Je založená na Mohri-Nederhofovej transformácii garmatiky, ktorá vlastne danú bezkontextovú gramatiku $G$ pretransformuje na gramatiku $G'$, ktorá je *silno regulárna* (strongly regular). Definícia silno regulárnej gramatiky sa zhoduje s *gramatikou bez vnorených cyklov* (non-self-embedding grammar) z [Cho59b].

**Definícia 2.2.10.** (**Gramatika s vnoreným cyklom**)
Bezkontextová gramatika $G$ je *gramatika s vnoreným cyklom* alebo *self-embedding gramatika*, ak v nej existuje odvodenie $A \Rightarrow^* \alpha A \beta$, pričom $\alpha \neq \epsilon$ a súčasne $\beta \neq \epsilon$.

**Definícia 2.2.11.** (**Silno regulárna gamatika**)
Bezkontextová gramatika $G$ je *silno regulárna* alebo *non-self-embedding gramatika*, ak to nie je gramatika s vnoreným cyklom.

Z [Cho59a] vieme, že silno regulárne gramatiky generujú regulárny jazyk.
Pri Mohri-Nederhofovej transformácii je dôležité rozdeliť množinu neterminálov $N$ na množiny *vzájomne rekurzívnych neterminálov*. Na to nám bude nápomocná relácia ekvivalencie $\mathcal{R}$ medzi dvoma neterminálmi $A$ a $B$ definovaná nasledovne:

$$A \mathcal{R} B \Leftrightarrow (\exists \alpha,\beta \in E^* : A \Rightarrow^* \alpha B \beta) \wedge (\exists \alpha,\beta \in E^* : B \Rightarrow^* \alpha A \beta)$$

Vo všeobecnosti je nerozhodnuteľné, či bezkontextová gramatika generuje regulárny jazyk [Ull67]. Avšak je rozhodnuteľné, či bezkontextová gramatika je silno regulárna [AGV02]. Z [Cho59a] vieme, že silno regulárna gramatika generuje regulárny jazyk. Takže to, čo budeme teraz chcieť dosiahnuť je zistiť, či daná bezkontextová gramatika $G$ je silno regulárna. Ak je silno regulárna, tak vieme, že bude generovať regulárny jazyk, čo aj chceme. No ak nie je silno regulárna, tak z nej budeme chcieť spraviť silno regulárnu gramatiku.

**Mohri-Nederhofova tranformácia**
Teraz si popíšeme konkrétnu transformáciu, ktorá vytvorí z bezkontextovej gramatiky silno regulárnu gramatiku ako je popísaná v [MjN00]. Nech $M$ je označenie pre jednotlivé množiny vzájomne rekurzívnych neterminálov.
Transformácia je definovaná nasledovne:

1. Pre každý neterminál $A \in M$, zavedieme nový neterminál $A' \notin N$ a pridáme

nasledujúce pravidlo do gramatiky:

$$A' \to \epsilon$$

2. Ak máme pravidlo s ľavou stranou $A \in M$ tvaru
$$A \to \alpha_0\ B_1\ \alpha_1\ B_2 \cdots\ B_m\ \alpha_m$$
(neterminály, ktoré nepatria do aktuálnej množiny $M$, považujeme v danom prípade za terminály),
pričom $m \geq 0$, $B_1$,...,$B_m \in M$, $\alpha_0 \ldots \alpha_m \in (\Sigma \cup (N$ - $M))$*, tak ho nahradíme množinou nasledujúcich pravidiel:

$$\begin{aligned} A &\to \alpha_0 B_1 \\ B'_1 &\to \alpha_1 B_2 \\ B'_2 &\to \alpha_2 B_3 \\ &\cdots \\ B'_{m-1} &\to \alpha_{m-1} B_m \\ B'_m &\to \alpha_m A' \end{aligned}$$

V prípade, že $m = 0$, tak pridáme iba pravidlo $A \to \alpha_0 A'$.

Takáto transofrmácia spraví aproximáciu gramatiky, kotrá je nadmnožinou pôvodného jazyka.

#### 2.2.4.5 Rozbitie cyklov (CycleBreaking) – aproximačná stratégia

Ďalším spôsobom, ako získať silno regulárnu gramatiku je rozbitie cyklov, ktoré si popíšeme. Najprv si však musíme zaviesť pojem *hranovo zafarbený graf pravidiel* $CP(G)$.
Predpokladajme, že gramatika G je v Chomského normálovej forme (všetky pravidlá tvaru $A \to BC$ alebo $A \to a$ ). Neterminály gramatiky budú vrcholmi grafu. Pre každé pravidlo $A \to BC$ budeme mať v $CP(G)$ hranu z vrcholu $A$ do $B$ zafarbenú

farbou $l$ a z vrcholu $A$ do $C$ farbou $r$.
Nasledujúce tvrdenie využijeme na zistenie, či bezkontextová gramatika je silno regulárna. Ak nie, tak zároveň získavame vhodnú informáciu na to, aby sme z nej mohli spraviť silno regulárnu gramatiku.

**Tvrdenie 2.2.11.** Gramatika G je silno regulárna práve vtedy, ak všetky cykly v $CP(G)$ sú monochromatické.

Dôkaz tohto tvrdenia nájdeme v [Ege09].

Teraz vieme, že z bezkontextovej gramatiky by sme vedeli spraviť silno regulárnu gramatiku, ak by sme vedeli spraviť všetky cykly monochromatické. Čiže chceme nemonochromatické cykly zmeniť na monochromatické. Preto budeme cykly "rozbíjať" a to tak, že odstránime hrany, ktoré nám kazia monochromatickosť.
Teraz si stručne popíšeme princíp, ako môžeme dané hrany odstrániť. Nech máme gramatiku $G$ a gramatiku $G'$, ktorá vznikla z $G$ Mohri-Nederhofovou transformáciou. Predpokladajme, že hrana medzi vrcholmi $A$ a $B$ v $CP(G)$ nám kazí monochromatickosť. Túto hranu odstránime tak, že ju zrušíme a nahradíme hranou z $A$ do vrcholu v $CP(G')$, ktorý zodpovedá vrcholu $B$ z $CP(G)$. Je to vlastne prepojenie dvoch grafov. Ak túto zmenu aplikujeme na gramatiky, tak ide vlastne o zmenu pravidla $A \to BC$ na pravidlo $A \to B_{MN}C$, premenovanie neterminálov $X$ v $G'$ na neterminály $X_{MN}$, kvôli odlíšeniu, a doplnenie premenovaných pravidiel z $G'$ do $G$. Týmto spôsobom vytvoríme silno regulárnu aproximáciu gramatiky $G$, lebo zmeníme všetky cykly na monochromatické a pri daných odstráneniach hrán sa odkážeme na MN aproximáciu gramatiky, ktorá je tiež silno regulárna.

**Tvrdenie 2.2.12.** (Užšia aproximácia)
Regulárna aproximácia $G''$, ktorá vznikla rozbíjaním nemonochromatických cyklov grafu $CP(G)$ s použitím MN-transformácie, je užšou hornou aproximáciou než samotná MN-aproximácia $G'$ gramatiky $G$.
Inak povedané, $\mathcal{L}(G) \subseteq \mathcal{L}(G'') \subseteq \mathcal{L}(G')$.

Odôvodnenie tohto tvrdenia ako aj podrobnejší popis stratégie rozbitia cyklov je popísaný v [Ege09].

## 2.3 Kombinované metódy

Tretia skupina metód je skupinou, do ktorej patria metódy, ktoré vznikli spojením viacerých metód, čiže ich kombináciou. V tejto časti si spomenieme spojenie aproximačnej metódy s úplnou metódou, ktorej tvorcom je Basten [Bas11], ktorú implementoval do nástroja s názvom AmbiDexter. Do tejto kategórie by sme taktiež mohli zaradiť konkrétnu implementáciu metódy ACLA z [BGM07], kde je mierna obmena pôvodnej aproximačnej metódy, no nie je to také výrazné spojenie dvoch metód, preto sa budeme sústrediť iba na prvú metódu.

**Rozšírenie RU a NU Testu**

Kombinovaná metóda od Bastena je založená na rozšírení RU Testu a NU Testu. Toto rozšírenie umožňuje detekovanie *neškodných pravidiel*. Sú to pravidlá, ktoré neprispievajú k nejednoznačnosti gramatiky a preto ich môžeme ignorovať, respektíve odfiltrovať. Pri rozšírení RU Testu vieme pomocou tohto prístupu nájsť pravidlá, ktoré odvodzujú jednoznačné slová, pri NU Teste vieme nájsť pravidlá, ktoré odvodzujú jednoznačné podslová nejednonzačných slov. Po odfiltrovaní neškodných pravidiel využíva táto metóda úplnú metódu, už na gramatike, ktorá je zbavená neškodných pravidiel a detekuje nejednoznačnosť na pravidlách, ktoré môžu túto nejednoznačnosť zapríčiniť. Keďže úplnú metódu používa táto metóda na gramatike bez neškodných pravidiel, je toto prehľadávanie rýchlejšie.

Teraz si popíšeme myšlienku tejto kombinovanej metódy.

**Nepotrebné stavy**

Prvým krokom pri tejto metóde je detekovať stavy v aproximovanom pozičnom automate z časti 2.2.1.3, ktoré nie sú použité pri dvojiciach ciest nejednoznačnosti. Tieto stavy sú teda stavmi jednoznačných odvodení. Ako tieto stavy nájsť je popísané v [Bas11].

**Filtrovanie neškodných pravidiel**

Ak máme nájdené stavy v pozičnom automate, ktoré sú stavmi pre jednoznačné odvo-

denia, potrebujeme zistiť, ktoré pravidlá im zodpovedajú. Tieto pravidlá nazývame *neškodlivé pravidlá* a množinu nešodlivých pravidiel označíme $P_{hl}$. Tie nájdeme tak, že si najprv určíme, ktoré pravidlá sú potenciálne škodlivé (tie ktoré môžu prispievať k nejdnoznačnosti), označíme $P_{hf}$) a zvyšné pravidlá budú tie, ktoré sú určite neškodlivé ($P_{hl} = P \setminus P_{hf}$). Ako zistiť, ktoré pravidlá sú neškodlivé je popísané v [Bas11].

**Zostavenie gramatiky**

Ak máme identifikované neškodné pravidlá, tak ich môžeme z gramatiky vynechať. Tak nám vzniká nová gramatika, ktorá obsahuje iba pravidlá, ktoré môžu prispievať k nejednoznačnosti. Môže sa nám zdať, že niektoré pravidlá v takto filtrovanej gramatike nebudú dosiahnuteľné z počiatočného neterminálu $S$ po vynechaní neškodných pravidiel. Avšak takéto pravidlá sú taktiež neškodné, lebo ak sa k nim neviem dopracovať z počiatočného neterminálu, to znamená, že sú súčasťou iba jednoznačných stromov odvodenia zostavených z odstránených neškodných pravidiel, čiže sú tiež neškodné. Takto filtrovanú gramatiku musíme ešte upraviť a doplniť nové pravidlá a terminály, ktoré zabezpečia celistvosť gramatiky. Z množiny $P_{hf}$ vieme vytvoriť novú gramatiku nasledovne, pričom $\alpha, \beta, \gamma \in E^*$:

1. množina definovaných neterminálov: $N_{def} = \{A \mid A \to \alpha \in P_{hf}\}$
2. používané ale nedefinované neterminály: $N_{undef} = \{B \mid A \to \alpha B \beta \in P_{hf}\} \setminus N_{def}$
3. neproduktívne neterminály:
   $N_{unpr} = \{A \mid A \in N_{def}, \nexists u : A \Rightarrow^* u$ používajúc iba pravidlá z $P_{hf}\}$
4. nové terminály $t_A$ pre každý neterminál $A \in N_{undef} \cup N_{unpr}$
5. pravidlá na doplnenie neproduktívnych a nedefinovaných neterminálov:
   $P' = P_{hf} \cup \{\ A \to t_A \mid A \in N_{undef} \cup N_{unpr}\ \}$
6. nová množina terminálov: $T' = \{a \mid A \to \beta a \gamma \in P'\}$
7. nová gramatika: $G' = (N_{def} \cup N_{undef}, T', P', S)$

**Úplná metóda**

Ak máme vytvorenú takto filtrovanú gramatiku, posledným krokom pri zisťovaní nejednoznačností je použitie metódy úplného prehľadávania. Takto použité prehľadávanie je vcelku užitočné, keďže môže nájsť nejednoznačnosti, ktoré sa v gramatike určite nachádzajú. Prehľadávnie je použité na filtrovanej gramatike, takže sa zaoberá iba okresanou množinou pravidiel, čo podstatne zníži jeho prehľadávaný priestor. Nevýhodou však je exponenciálna zložitosť tejto metódy.
Podrobnejší popis tejto kombinovanej metódy nájdeme v [Bas10].

## 2.4 Ďalšie metódy

Okrem týchto troch základných skupín metód si spomenieme metódu, ktorá charakterovo nepatrí ani do jednej z troch spomínaných skupín. Je to metóda $LR(k)$ Testu podľa [Knu65].

### 2.4.1 $LR(k)$ Test

$LR(k)$ parsovanie je parsovacia technika, ktorí spraví rozhodnutie na základe $k$ vstupných symbolov vo výhľade. Množina gramatík, ktoré sú deterministicky parsovateľné týmto algoritmom sa nazýva množina $LR(k)$ gramatík. To znamená, že pre tieto gramatiky existuje hodnota $k$, pre ktorú vie byť zostrojené parsovacia tabuľka danej gramatiky bez konfliktov. Konflikt nastáva vtedy, ak v danom stave sa nám naskytuje možnosť viacerých akcií. Takýto konflik znamená, že nemáme deterministické rozhodnutie, ktorú akciu urobiť pri parsovaní.
Takže ak je daná gramatika $LR(k)$ gramatikou, potom je jednoznačná. Ak gramatika nie je $LR(k)$ gramatikou, tak nevieme s istotou prehlásiť či je jednoznačná.

Zisťovanie, či daná gramatika patrí do množiny $LR(k)$ gramatík, môže byť uskutočňované tak, že budeme generovať jej parsovaciu tabuľku pre hodnoty $k$. Ak parsovacia tabuľka neobsahuje žiadne konflikty, tak gramatika patrí do množiny $LR(k)$ gramatík. Takýto test môže byť použitý ako metóda na zisťovanie nejednoznačnosti.

Pri danej gramatike môžeme postupne zvyšovať hodnotu $k$ a generovať jej parsovaciu tabuľku. Ak nájdeme takú parsovaciu tabuľku, v ktorej nebudú žiadne konflikty, tak ju prehlásime za jednoznačnú.

Ak vstupná gramatika bude naozaj patriť do množiny $LR(k)$ gramatík, tak algoritmus skončí. Ale ak daná gramatika nepatrí do tejto množiny, algoritmus bude pokračovať do nekonečna.

Takže táto metóda vie vrátiť výsledok iba ak gramatika je jednoznačná. V opačnom prípade, ak gramatika je nejednoznačná algoritmus neskončí.

# 3

# Existujúce nástroje

V predošlej kapitole sme sa zaoberali popisom jednotlivých metód na detekciu nejednoznačnosti. V tejto kapitole si spomenieme konkrétne nástroje, ktoré sa snažia o zisťovanie nejednoznačnosti v bezkontextových gramatikách a uvedieme si aj to, ktorú metódu využívajú. Pokúsime sa ich stručne popísať a vystihnúť ich podstatu.

## 3.1 AMBER

AMBER je nástroj na detekciu nejednoznačností gramatík, ktorého autorom je Schröer [Sch01]. Je nástrojom, ktorého spôsob zisťovania nejednoznačnosti patrí k úplným metódam. Na generovanie všetkých možných slov gramatiky a zisťovanie ich duplicít využíva Earleyho parser [Ear70]. V podstate sa jedná o Gornovu metódu, avšak tento nástroj ponúka možnosť použiť vstupné parmetre, pomocou ktorých môžeme využiť rôzne varianty prehľadávania. Jednou z možností je porovnávať všetky vygenerované vetné formy narozdiel od porovnávania iba terminálnych slov. Takéto vylepšenie môže viesť k rýchlejšiemu odhaleniu nejednoznačného slova.
AMBER ponúka možnosť určiť dĺžku generovaných slov, alebo maximálny počet neterminálov, ktoré bude prepisovať. Ďalej môžeme špecifikovať aj percento expanzie, použité v jednotlivých krokoch detekcie.
Ak máme na vstupe rekurzívnu gramatiku AMBER nevie skončiť generovanie, avšak

pri zadaní dĺžky generovaných slov zastaví.
Amber je dostupný online na [amb], kde sa nachádza aj stručný popis tohto nástroja.

## 3.2 CFG Analyzer

CFG ANALYZER je nástroj na detekciu nekednoznačnosti, ktorý využíva SAT-solver na úplné prehľadávanie všetkých vetných foriem gramatiky so zväčšujúcou sa dĺžkou. Takže patrí do kategórie nástrojov využívajúcich úplnú metódu hľadania nejednoznačností. Okrem problému nejednoznačnosti sa tento nástroj zaoberá aj ďalšími problémami ako je napríklad problém ekvivalencie. Tento nástroj je dostupný na [cfga], kde je aj jeho bližší popis. Jeho teoretické pozadie je popísané v [AHL08].

## 3.3 ACLA framework

Tento framework využíva metódu, ktorá je popísaná v časti 2.2.3. Využíva aproximačné stratégie ako $MayMust$, $FirstLast$, $EmptyString$ a $Mohri - Nederhofovu$ aproximačnú stratégiu a detekuje nejednoznačnosť pomocou horizontálonej a vertikálnej nejednoznačnosti. Tento framework narozdiel od bežnej aproximačnej metódy využíva Earleyho parser, ktorý kontroluje, či vzniknté nejednoznačnosti sa dajú naozaj vygenerovať pôvodnou gramatikou.
Ak nastane prípad, že pri aproximácii nájdeme nejednoznačnosť, môžeme ju zatiaľ nazývať potenciálna nejednoznačnosť, lebo v pôvodnej gramatike táto nejednoznačnosť vôbec nemusí nastať. Pomocou parsera tento framework zistí, či potenciálne nejdnoznačné slovo sa dá naozaj vygenerovať aj v pôvodnej gramatike z daného pravidla. Je rozdiel pri kontrole potenciálne horizontálnej aj vertikálnej nejednoznačnosti. Pri vertikálnej paser kontroluje, či sa dá potenciálne nejednoznačné slovo vygenerovať z oboch kontrolovaných pravidiel v pôvodnej gramatike. Pri horizontálnej kontroluje, či jednotlivé prekrývajúce sa časti potenciálne nejednoznačného slova sa dajú vygenerovať v pôvodnej gramatike. Ak to platí, tak sa potenciálna nejednoznačnosť zmení na stopercentnú nejednoznačnosť. V opačnom prípade, zostáva

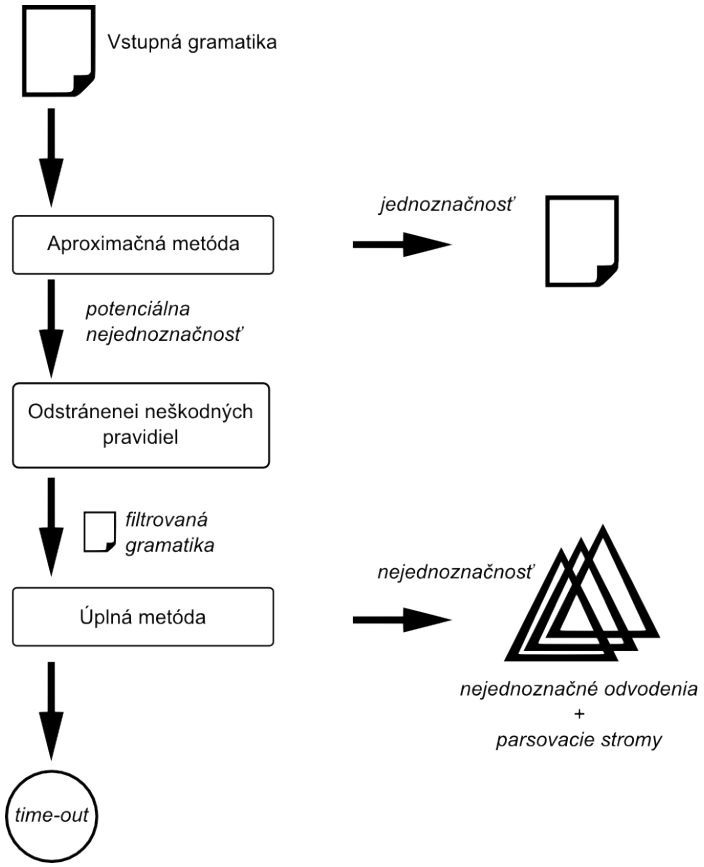
táto nejednoznačnosť potenciálnou.
Tento nástroj je dostupný na [acl], kde je možné vyskúšať si aj online verziu. Viac detailnejšieho popisu tohto nástroja sa dá zísať v [BGM07].

## 3.4 AmbiDexter

Framework AmbiDexter kombinuje rozšírenie NU Testu [Sch07b] s úplnou metódou generovania vetných foriem podobnej metóde [Sch01]. Ako sme už spomínali v časti 2.3 tento nástroj je založený na princípe rozdelenia pravidiel na neškodlivé (tie, ktoré s určitosťou neprispievajú k nejednoznačnosti) a tie, ktoré môžu nejednoznačnosť zapríčiniť. Potom využíva generovanie vetných foriem (úplnú metódu) na zisťovanie, či dané potenciálne škodlivé pravidlá naozaj generujú nejednoznačné slová. Toto generovanie robí na gramatike, ktorá je filtrovaná, to znamená, že neobsahuje v sebe pravidlá, ktoré určite neprispievajú k nejednoznačnosti (neškodlivé pravidlá). Na obrázku 3.1 môžeme vidieť ako nástroj AmbiDexter pracuje.

## 3.5 CFG Tool

Nástrojom, ktorý nie je vyslovene iba nástrojom na detekciu nejednoznačnosti, ale robí ďalšie operácie na gramatikách (zostrojovanie parsovacích tabuliek, výpočet množín First a Follow, a ďalšie) je CONTEXT FREE GRAMMAR TOOL. Tento nástroj je online dostupný na [CFGb]. K tomuto nástroju nemáme podrobnú dokumentáciu, ktorá by popisovala akú z metód na detekovanie nejednoznačností využíva, ale pripravuje sa nová verzia tohto nástroja s názvom GRAMMOPHONE [gra]. Spomíname ho, lebo môže byť užitočný pre lepšie pochopenie našej problematiky.




Obr. 3.1: Popis ako pracuje AmbiDexter

## 3.6 Dr. Ambiguity

Ďalším nástrojom vhodným pre oblasť nejednoznačnosti je nástroj Dr. AMBIGUITY. Tento nástroj nie je výslovne nástrojom, ktorý by detekoval nejednoznačnosti, ale snaží sa tieto nejednoznačnosti odstrániť, čo je zaujímavé a preto ho okrajovo spomíname. Nezaradíme ho teda, do žiadnej z našich kategórií metód na zisťovanie nejednoznačnosti. Presnú metódu odstraňovania nejednoznačností v tejto práci nebudeme spomínať, ale podrobnejší popis, ako daný nástroj pracuje môžeme nájsť v [BV12].

# 4

# Implementácia

V tejto časti si popíšeme našu implementačnú prácu. Popíšeme aké nástroje a aké metódy sme pri nej použili a zároveň prezentujeme naše dosiahnuté výsledky. Zároveň ich porovnáme s doterajším riešením.

## 4.1 Popis implementácie

Pri implementácii budme využívať framework ACLA [acl], ktorý je implementovaný v programovacom jazyku Java. Do tohto frameworku implementujeme novú aproximačnú stratégiu, stratégiu rozbitia cyklov, pričom využijeme aj obmenu pôvodnej ACLA metódy rozšírenú o kontrolovanie možných nejednoznačných slov parserom.

### 4.1.1 ACLA framework

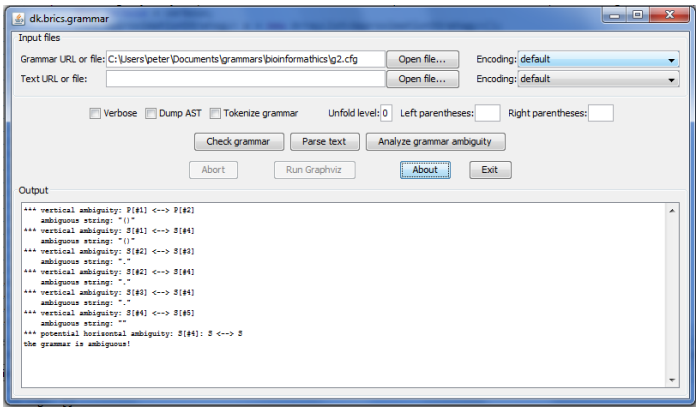
ACLA framework (implementovaný ako balík `dk.brics.grammar` [acl]) využíva k svojej funkcionalite ďalšie dve javovské implementácie.
A to konkrétne balík `dk.brics.automaton` [aut] a nástroj *Java String Analyzer*, ktorý je implementovaný ako balík `dk.brics.string` [jsa]. Oba tieto balíky sú od rovnakých autorov ako ACLA framework. Framework poskytuje grafické rozhranie 4.1, s možnosťou výberu gramatiky zo súboru alebo z url adresy. Po analýze zobrazí

výsledok:

`"the grammar is unambiguous!"`, ak nenájde žiadne horizontálne a verikálne nejednoznačnosti

`"the grammar is ambiguous!"`, ak zisí vertikálne alebo horizontálne nejednozačnosti a potvrdí nejednoznačnosť parserom

`"the grammar might be ambiguous, but I'm not sure..."`, ak zisí iba potenciálne vertikálne alebo horizontálne nejednozačnosti.

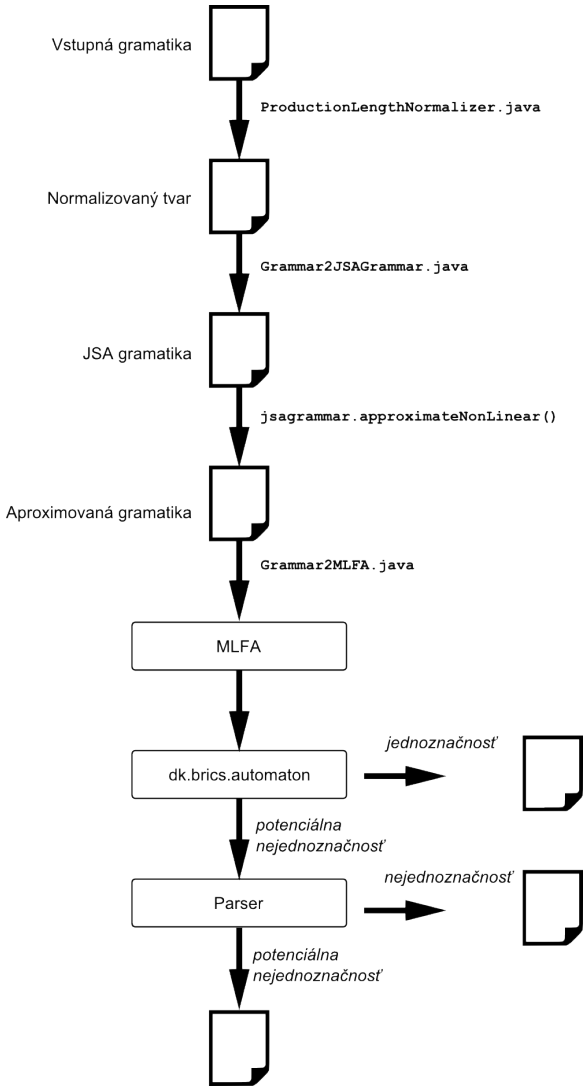



Obr. 4.1: Grafické rozhranie frameworku ACLA

Základom tohto nástroja je Mohri-Nederhofova transformácia, pred ktorej použitím využíva ešte aporoximačné stratégie ako $MayMust, FistLast, EmptyString$ spomínané v 2.2.4. My sa budeme zaoberať hlavne Mohri-Nederhofovou transformáciou a ako interne ACLA pracuje s touto aproximačnou stratégiou, lebo je všeobecnejšia oproti ostatným a samozrejme lepšia v tom zmysle, že je to užšia aproximácia.

#### 4.1.1.1 Popis detekcie nejednoznačnosti

V tejto časti si povieme, ako nástroj ACLA pracuje so vstupnou gramatikou a akým spôsobom detekuje nejednoznačnosť s využitím MN-transformácie. Táto aproximačná stratégia je implementovaná v triede `RegularApproximation` nástroja ACLA. Popis ako táto metóda funguje môžeme vidieť na obrázku 4.2.




Obr. 4.2: ACLA a RegularApproximation.java

### Normalizovaný tvar

V prvom kroku ACLA framework vytvorí zo vstupnej gramatiky normalizovaný tvar. Normalizácia spočíva v tom, že každé pravidlo s pravou stranou dĺžky väčšej ako *2 symboly* (dĺžka bude zodpovedať počtu symbolov na pravej strane) rozdelíme na pravidlá s dĺžkou *2 symboly* a to pridaním nových neterminálov. Postup si ukážeme na príklade.

**Príklad 4.1.1.** (Normalizovaný tvar)
Nech máme pravidlo $A \rightarrow$ a b $C$ d.
Pri použití normalizácie dostaneme pravidlá $A \rightarrow$ a $B'$, $B' \rightarrow$ b $C'$, $C' \rightarrow C$ d, pričom

novými pridanými neterminálmi sú $B'$ a $C'$.

Na základe tohto príkladu by mal byť jasný postup normalizácie gramatiky.

**JSA-gramatika**

Normalizovaný tvar gramatiky potrebuje nástroj ACLA preto, aby mohol prekonvertovať túto gramatiku do formátu gramatiky, ktorý je využívaný v balíku `dk.brics.string`. Tetno formát gramatiky budeme nazývať *jsa-gramatika* a je to vlastne obyčajné premenovanie neterminálov, kde každý neterminál dostane názov v tvare `x`[`číslo`]. Taktiež si pamätá, ktorým pôvodným neterminálom prislúchajú aké nové neterminály. Čísla neterminálov idú po poradí v akom sa postupne premenúvávajú. V tejto konverzii taktiež každému terminálu bude zodpovedať samostatný neterminál v rovnakom tvare (`x`[`číslo`]). Táto konverzia je potrebná preto, lebo v ďalších krokoch využíva balík `dk.brics.string`, ktorý vyžaduje takýto formát gramatiky. Na konverziu používa triedu `Grammar2JSAGrammar` z balíka `dk.brics.grammar`.

**Príklad 4.1.2.** (Normalizovaný tvar → JSA-gramatika)
Normalizovaná gramatika:
$A \to A\ A'$, $A \to$ c, $A' \to$ a b.
JSA-gramatika:
$X_0 \to X_0\ X_1$, $X_0 \to X_4$, $X_1 \to X_2\ X_3$, $X_2 \to$ a, $X_3 \to$ b, $X_4 \to$ c.

**MN-transformovaná gramatika**

Ďalším krokom, ktorý nasleduje pri detekcii, je vytvorenie samotnej aproximovanej gramatiky pomocou Mohri-Nederhofovej transformácie. Pri transformácii využíva triedu `StronglyConnectedComponents`, pomocou ktorej určí množiny vzájomne rekurzívnych neterminálov. Následne metódou `approximateNonLinear(hotspots)` aplikovanou na jsa-gramatike získame Mohri-Nederhofovu aproximáciu gramatiky ako je popísaná v časti 2.2.4.4.

**MLFA a dk.brics.automaton**

Po vytvorení aproximovanej gramatiky nástroj kontroluje prekryvy a prieniky jed-

notlivých pravidiel. Na to využíva balík `dk.brics.automaton`. Pred použitím tohto balíka, ale vytvrorí MLFA (Multi-Level Finite Automaton), čo je nový formalizmus bližšie popísaný v [CMS03], ktorý pomáha namapovať neterminály na stavy automatu a pre jednotlivé neterminály vytvoriť akoby podautomat (podautomat v zmysle časť automatu prislúchajúceho celej gramatike), a ktorého spoluatorom je aj tvorca framewroku ACLA. Takže v tomto kroku sa kontrolujú prekryvy a prieniky pričom automat zistí či sa vyskytujú potenciálne nejednozančnosti.

**Parser**

ACLA môže na konci zistiť, že gramatika je jednoznačná a detekcia je ukončená. No, keď má ACLA vygenerované potenciálne nejednoznačnosti, tak dodatočne ešte Earleyho parserom skontroluje, či potenciálne nejednoznačné slovo sa naozaj dá odvodiť z daného pravidla v pôvodnej gramatike. Ak sa dá, gramatika je nejednoznačná. Ak sa to nedá, nejednoznačnosť zostáva potenciálnou nejednoznačnosťou a framework nevie rozhodnúť, o nejednoznačnosti, respektíve jednoznačnosti gramatiky. Táto implementáci testuje len jedno slovo z prieniku, preto potenciálna nejednoznačnosť ostáva potenciálnou.

### 4.1.2 Náš prístup

Pri našom prístupe využijeme aproximačnú stratégiu rozbitia cyklov 2.2.4.5, ktoré implmentujeme do framworku ACLA s využitím jeho vylepšenia pomocou parsera. Túto stratégiu implmentujeme do triedy s názvom
`RegularApproximationCycleBreaking.java`.
Teraz si ukážeme ako sme pri implementácii postupovali, čo je znázornené aj na obrázku 4.3.

Pri našej implementácii rozbitia cyklov budeme na začiatku postupovať podobne ako implementácia v `RegularApproximation.java`. Vstupnú gramatiku prevedieme do normalizovaného tvaru, z ktorého vytvoríme jsa-gramatiku tak ako sme to popísali v 4.1.1.1. Avšak tuto sa náš prístup začína líšiť od pôvodného riešenia.

**MN-transformovaná gramatika**

V tomto momente z jsa-gramatiky vytvoríme MN-transformáciou aproximovanú gramatiku, pričom však pôvodnú jsa-gramatiku nemôžeme meniť, lebo ju budeme potrebovať. Takže budeme mať gramatiky dve: jsa-gramatiku a aproximáciu jsa-gramatiky gramatiky.

**Premenovanie gramatiky**

Tieto dve gramatiky budeme chcieť neskôr spojiť do jednej, no nemôžeme tak spraviť hneď. Názvy neterminálov v jsa-gramatike a aproximovanej jsa-gramatike by kolidovali. Preto neterminály v aproximovanej jsa-gramatike premenuje. Spravíme to takým spôsobom, že `[číslo]` pri neterminále zmeníme. Na to nám bude slúžiť metóda `shiftKeyValuesForNonterminals()`, ktorá tieto hodnoty zmení.
Avšak tieto hodnoty nemôžeme meniť hociako, lebo chceme zachovať konzistentnosť gramatiky, ktorá vznikne neskôr spojením jsa-gramatiky a premenovanej aproximovanej jsa-gramatiky. Všetky hodnoty čísel pri neterminálach teda zvýšime o také číslo, ktoré je rovné počtu neterminálov v jsa-gramatike (najvyššiemu použitému číslu pri neterminálach v jsa-gramatike zvýšeného o hodnotu 1).

**Spojená gramatika**

Teraz takto premenovanú aproximovanú jsa-gramatiku môžeme spojiť s pôvodnou jsa-gramatikou a vytvoriť novú, s ktorou budeme pracovať. Na spojenie nám slúži trieda `JSAGrammarConnector`, ktorá nám tieto dve gramatiky spojí do jednej. Keďže pri všetkých týchto konverziách gramatík meníme pomenovania neterminálov, je potrebné si pamätať, ktoré neterminály sa zmenili a aj ich novú hodnotu. To si priebežne ukladnáme do štruktúry `HashMap`, aby sme vedeli aké je spätné mapovanie neterminálov pri detekovaní nejednoznačností.

**Rozbitie cyklov**

Ďalšou fázou je rozbitie cyklov ako sme si ho popísali v 2.2.4.5. Na to, aby sme cykly našli sme využili metódu `getComponents()` jsa-gramtiky, ktorá nám vráti silne súvislé komponenty grafu a ich "zafarbenie" ako sme ho spomínali v 2.2.4.5. Táto metóda používa na zistenie cyklov a ich zafarbenia tiredu `StronglyConnectedComponens`, ktorá hľadá komponenty pomocou Tarjanovho algoritmu.

Ak zistíme, že cyklus nie je monochromatický, zrušíme prvú nájdenú hranu v tomto cykle a presmerujeme ju na príslušný neterminál z aproximovanej jsa-gramatiky (v tomto momente sú gramatiky už spojené, takže stačí iba zmeniť neterminál v pravidle, ktoré zodpovedalo danej hrane v grafe). Nájdenie príslušného neterminálu, na ktorý chceme presmerovať hranu je jednoduché, keďže sme neterminály premenuvávali iba zvýšením čísla o určitú hodnotu, ktorú vieme zistiť. Ďalej zistíme, či neexistujú ďalšie nemonochromatické cykly.
Takýmto spôsobom pokračujeme, až kým nebudeme mať všetky cykly monochromatické.
Tento proces s určitosťou zastaví, keďže rušíme nemonochromatické cykly jsa-gramatike (stávajú sa monochromatickými cyklami) a prepájame ich s neterminálmi v silne regulárnej gramatike (premenovaná jsa-gramatika), ktorá má určite monochromatické cykly podľa tvrdenia 2.2.11.
Takto získame gramatiku, v ktorej všetky cykly sú monochromatické a budeme ju nazývať *monochromatická*.

**Detekcia nejednoznačnosti**

Teraz pomocou tejto monochromatickej gramatiky, ktorá je vlastne novou aproximáciou pôvodnej vstupnej gramatiky, môžeme hľadať prekryvy a prieniky, pričom opäť využijeme MLFA a `dk.brics.atomaton` ako v pôvodnom riešení. Ak zistíme že nejednoznačnosti neexistujú, tak môžeme skončiť a prehlásiť, že gramatika je jednoznačná. No ak zistíme potenciálne nejednoznačnosti, tak ich opäť podrobíme testom pomocou parsera, ktorý môže detekovať stopercentné nejednoznačnosti.

## 4.2 Porovnanie výsledkov

V tejto časti si povieme aká je miera úspešnosti nášho riešenia a zároveň prezentujeme aj výsledky, ktoré sme dosiahli pri testovaní a porovnávaním našej implementácie oproti pôvodnej.

### 4.2.1 Miera úspešnosti

Na to, aby sme vedeli povedať, do akej miery je naše riešenie úspešné musíme si povedať, čo je vlastne meradlom úspešnosti riešenia.
Keďže pôvodné riešenie vie s určitosťou prehlásiť, či daná gramatika je jednoznačná, ak nezistí žiadne nejednoznačnosti, a taktiež vie s určitosťou prehlásiť, že gramatika je nejednoznačná, ak zistí stopercentnú nejednoznačnosť, zostáva nám jediný prípad, a to, ak pôvodná metóda nevie zistiť informáciu o jednoznačnosti a nejedoznačnosti gramatiky. Čiže tie prípady, kedy pôvodné riešenie dokáže zistiť iba potenciálne nejednoznačnosti, sú pre nás smerodajné.
Jednoducho potrebujeme znížiť počet potenciálnych nejednoznačností pri detekcii a zachovať výsledky pri jednoznačnnej gramatike a istotne nejednoznačnej.

Zníženie počtu potenciálnych nejednoznačností môžu nastať pri dvoch prípadoch:

- *potenciálna nejednoznačnosť sa pri novej metóde zmení na stopercentnú nejednoznačnosť* - pri daných pravidlách sa vyskytne slovo, ktoré parser vie rozpoznať
- *potenciálna nejednoznačnosť sa pri novej metóde zmení na jednoznačnosť* - vďaka užšej aproximácii pri daných pravidlách nevznikol prekryv alebo prienik

### 4.2.2 Výsledky

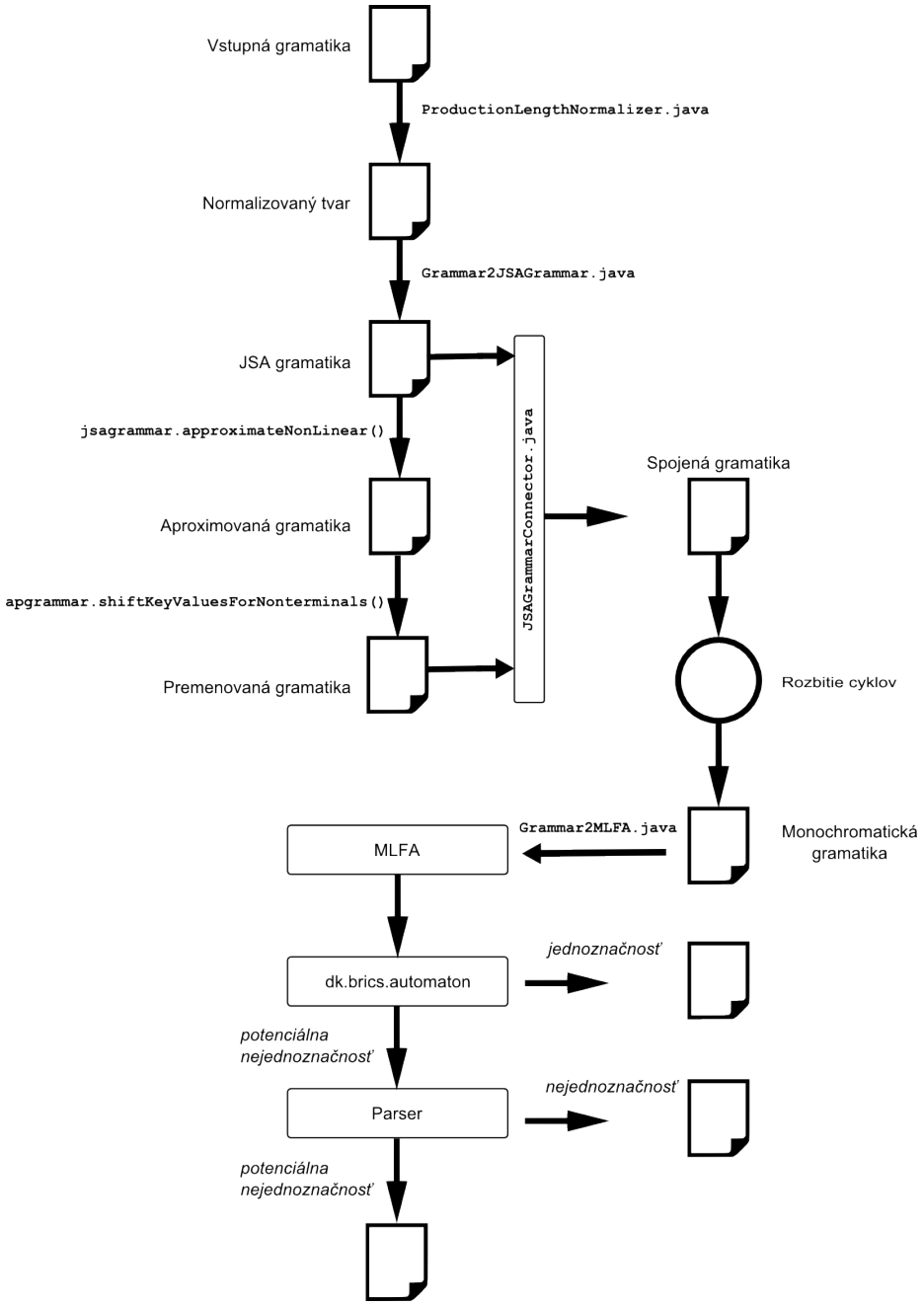
V tejto časti prezentujeme výsledky našej implementácie a porovnáme s pôvodnou implmentáciou. Našu aj pôvodnú implmentáciu sme testovali na testovacej množine gramatík, ktorú sme si stiahli na [acl]. Sú to gramatiky od Schmitza, bioinformatické od Giegericha a ďalšie príklady gramatík.

Výsledky sme zhrnuli do tabuliek 4.4, 4.5, 4.6, v ktorých hrubším písmom sú vyznačené zlepšenia oproti pôvodnej implementácii. Uvádzame vysvetlenie označenia použitého v týchto tabuľkách:

H - horizontálna nejednoznačnosť
V - vertikálna nejednoznačnosť
PH - potenciálne horizontálna nejednoznačnosť
PV - potenciálne vertikálna nejednoznačnosť
unambiguous - jednoznačnosť

### 4.2.3 Zhodnotenie

Na testovaných garmatikách sme si potrdili, že aproximácia pomocou rozbitia cyklov je lepšia než samotná Mohri-Nederhofova transformácia.
V testovanej množine gramatík sme dosiahli 5 zlepšení pri detekcii, takže naša implemetnácia bola v porovnaní s pôvodnou úspešnejšia.




Obr. 4.3: ACLA a RegularApproximationCycleBreaking.java

| Gramatika | `RegularApproximation` | `RACycleBreaking` |
|---|---|---|
| palindromes.cfg | unambigious | unambigious |
| reverse.cfg | unambigious | unambigious |
| exp.cfg | unambigious | unambigious |
| java-exp.cfg | 22PV, 26PH | 22PV, 26PH |
| odd-even.cfg | unambigious | unambigious |
| h-amb.cfg | 1H | 1H |
| v-amb.cfg | 1V | 1V |

Obr. 4.4: Malé testované gramatiky

| Gramatika | `RegularApproximation` | `RACycleBreaking` |
|---|---|---|
| R.cfg | unambiguous | unambiguous |
| g1.cfg | 5V, 1H | 5V, 1H |
| g1.cfg | 6V, 1H | 6V, 1H |
| g3.cfg | 2PV, 1PH | 2PV, 1PH |
| g4.cfg | 1PV, 2PH | 1PV, 2PH |
| g5.cfg | 2PH | 2PH |
| g6.cfg | 2PV, 2PH | 2PV, 2PH |
| g7.cfg | 4PV, 2PH | 4PV, 2PH |
| g8.cfg | 2PV, 4PH | 2PV, 4PH |
| basepairs.cfg | unambiguous | unambiguous |
| voss-light.cfg | 2PV, 2PH | **unambiguous** |
| voss.cfg | 39PV, 27PH | **38PV, 25PH** |

Obr. 4.5: Gramatiky z bioinformatiky (zozbierané Robertom Giegerichom):

| **Gramatika** | `RegularApproximation` | `RACycleBreaking` |
|---|---|---|
| s1.cfg | 1V, 2H | 1V, 2H |
| s2.cfg | unambiguous | unambiguous |
| s3.cfg | unambiguous | unambiguous |
| s4.cfg | 1V | 1V |
| s5.cfg | 1PV | 1PV |
| s6.cfg | unambiguous | unambiguous |
| s7.cfg | 1PH | 1PH |
| 076.cfg | 2PH | **1H, 1PH** |
| 011.cfg | 1H | 1H |
| 124.cfg | unambiguous | unambiguous |
| 170.cfg | 1V, 2H | 1V, 2H |
| 027.cfg | unambiguous | unambiguous |
| 081.cfg | unambiguous | unambiguous |
| 041.cfg | unambiguous | unambiguous |
| 047.cfg | 1V, 1PV, 6PH | **2V, 5H, 1PH** |
| 092.cfg | unambiguous | unambiguous |
| 114.cfg | unambiguous | unambiguous |
| 028.cfg | 2H, 3PV, 8PH | 2H, 3PV, 8PH |
| 036.cfg | 1V, 8PV, 18PH | 1V, 8PV, 18PH |
| 042.cfg | unambiguous | unambiguous |
| 014.cfg | 2V, 1PV, 5PH | **2V, 1H, 1PV, 4PH** |
| 030.cfg | 3PV, 7PH | 3PV, 7PH |
| 215.cfg | unambiguous | unambiguous |
| java_arrays.cfg | unambiguous | unambiguous |
| java_casts.cfg | unambiguous | unambiguous |
| java_modifiers.cfg | unambiguous | unambiguous |
| java_names.cfg | unambiguous | unambiguous |

Obr. 4.6: Ďalšie gramatiky (zozbierané Sylvainom Schmitzom):

# Záver

V tejto práci sme sa zaoberali detekciou nejednoznačnosti bezkontextových gramatík a popisom metód a nástrojov vhodných na jej detekciu, pričom sme sa sústredili na aproximačnú metódu ACLA.

Ďalším cieľom tejto práce bolo implementovanie lepšej aproximačnej stratégie než Mohri-Nederhofovej transformácie do framewroku ACLA. Lepšou aproximáciou je aproximácia pomocou rozbitia cyklov, ktorú sme popísali v časti 2.2.4.5. Podarilo sa nám ju úspešne implementovať a tak skĺbiť novú aproximáciu s frameworkom ACLA. Následne sme našu implementáciu porovnali s pôvodným riešením, ktoré využívalo Mohri-Nederhofovu transformáciu. Testovaním sme zistili, že naša implementácia aproximácie pomocou rozbitia cyklov je naozaj lepšia ako pôvodná implementácia, ktorá využíva MN transformáciu, čo sme ukázali v záverečnej kapitole 4.

Pri spracovávaní tejto diplomovej práce sme narazili na ďalšie možné vylepšenia, ktoré by sme chceli v budúcej práci zrealizovať.
Jedným z možných zlepšení by mohlo byť zlepšenie v oblasti rozbíjania cyklov. Cykly by sme mohli rozbíjať tak, aby sme týchto rozbití spravili čo najmenej. Ak spravíme, čo najemenej rozbití, tým je aproximácia viac podobná pôvednej gramatike.

Ďalšou zaujímavou oblasťou by bolo skombinovanie dolnej a hornej aproximácie gramatiky a pomocou takejto kombinácie následné detekovanie nejednoznačnosti s využitím horizontálnej a vertikálnej nejednoznačnosti.
Zaujímavým vylepšením by mohlo byť pri frameworku ACLA, parametrizovanie množstva slov z prienikov a prekryvov parsovaných parserom (v momentálnej implementácii sa testuje iba jedno slovo z prieniku).
Všetky tieto zlepšenia ale potrebujú dôkladnejšie preskúmanie a testovanie.

# Zoznam literatúry


[acl] dk.brics.grammar [online][cit. 25.4.2014]. `http://www.brics.dk/grammar/`.

[AGV02] Marcella Anselmo, Dora Giammarresi, and Stefano Varricchio. Non–self–embedding grammars as representation for regular languages. In *ICIAA Conference Proceedings*, 2002.

[AHL08] Roland Axelsson, Keijo Heljanko, and Martin Lange. Analyzing context-free grammars using an incremental sat solver. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 410–422, Berlin, Heidelberg, 2008. Springer-Verlag.

[amb] Amber, an ambiguity checker for context-free grammars [online][cit. 25.4.2014]. `http://accent.compilertools.net/Amber.html`.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[aut] dk.brics.automaton - finite-state automata and regular expressions for java [online][cit. 25.4.2014]. `http://www.brics.dk/automaton/`.

[Bas10] H. J. S. Basten. Tracking down the origins of ambiguity in context-free grammars. In *Proceedings of the 7th International Colloquium Conference on Theoretical Aspects of Computing*, ICTAC'10, pages 76–90, Berlin, Heidelberg, 2010. Springer-Verlag.

[Bas11] Bas Basten. *Ambiguity Detection for Programming Language Grammars*. These, Universiteit van Amsterdam, December 2011.

[BGM07] Claus Brabrand, Robert Giegerich, and Anders Møller. Analyzing ambiguity of context-free grammars. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, CIAA'07, pages 214–225, Berlin, Heidelberg, 2007. Springer-Verlag.

[BV12] Hendrikus J. S. Basten and Jurgen J. Vinju. Parse forest diagnostics with dr. ambiguity. In *Proceedings of the 4th International Conference on Software Language Engineering*, SLE'11, pages 283–302, Berlin, Heidelberg, 2012. Springer-Verlag.

[Can62] David G. Cantor. On the ambiguity problem of backus systems. *J. ACM*, 9(4):477–479, October 1962.

[cfga] Cfganalyzer [online][cit. 25.4.2014]. `http://www2.tcs.ifi.lmu.de/~mlange/cfganalyzer/`.

[CFGb] Context free grammar tool [online][cit. 25.4.2014]. `http://smlweb.cpsc.ucalgary.ca/start.html`.

[Cho59a] Noam Chomsky. A note on phrase structure grammars. *Information and Control*, 2(4):393–395, 1959.

[Cho59b] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

[CMS03] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In *Proceedings of the 10th International Conference on Static Analysis*, SAS'03, pages 1–18, Berlin, Heidelberg, 2003. Springer-Verlag.

[CS] Noam Chomsky and Marcel Paul Schützenberger. In P. Braffort and D. Hirshberg, editors, *Computer Programming and Formal Systems*, Studies in Logic, pages 118–161. North-Holland Publishing, Amsterdam.

[CU95] Bruce S.N. Cheung and R.C. Uzgalis. Ambiguity in context-free grammars. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, SAC '95, pages 272–276, New York, NY, USA, 1995. ACM.

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, February 1970.

[Ege09] Ömer Egecioglu. Strongly regular grammars and regular approximation of context-free languages. In *Developments in Language Theory*, pages 207–220, 2009.

[Flo62] Robert W. Floyd. On ambiguity in phrase structure languages. *Commun. ACM*, 5(10):526–, October 1962.

[GH67] Seymour Ginsburg and Michael A. Harrison. Bracketed context-free languages. *J. Comput. Syst. Sci.*, 1(1):1–23, April 1967.

[Gor63] Saul Gorn. Detection of generative ambiguities in context-free mechanical languages. *J. ACM*, 10(2):196–208, April 1963.

[gra] Grammophone [online][cit. 25.4.2014]. `http://mdaines.github.io/grammophone/`.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Jam05] S. Jampana. Exploring the problem of ambiguity in context-free grammars. Master's thesis, Oklahoma State University, Oklahoma, July 2005.

[jsa] Java string analyzer [online][cit. 25.4.2014]. `http://www.brics.dk/JSA/`.

[Knu65] Donald E. Knuth. On the translation of languages from left to rigth. *Information and Control*, 8(6):607–639, 1965.

[Kru08] Michael Kruse. Ambiguity detection for context-free grammars in eli. Bachelor's thesis, Universität Paderborn, Die Universität der Informationgesellschaft, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, 7th May 2008.

[MjN00] Mehryar Mohri and Mark jan Nederhof. Regular approximation of context-free grammars through transformation, 2000.

[Ned00] Mark-Jan Nederhof. Practical experiments with regular approximation of context-free languages. *Comput. Linguist.*, 26(1):17–44, March 2000.

[Poo] Bjorn Poonen. Undecidable problems: A sampler. Department of Mathematics, Massachusetts Institute of Technology, Cambridge, USA.

[RF13] Branislav Rovan and Michal Forišek. *Formálne jazyky a automaty (skriptá, verzia zo dňa 30.decembra 2013)*. December 2013.

[Sch01] F. W. Schröer. Amber, an ambiguity checker for context-free grammars. technical report, compilertools.net, 2001. http://accent.compilertools.net/Amber.html, 2001.

[Sch07a] Sylvain Schmitz. *Approximating Context-Free Grammars for Parsing and Verification*. Thèse de doctorat, Laboratoire I3S, Université de Nice-Sophia Antipolis, France, September 2007.

[Sch07b] Sylvain Schmitz. Conservative ambiguity detection in context-free grammars. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, pages 692–703, Berlin, Heidelberg, 2007. Springer-Verlag.

[Ull67] Joseph S. Ullian. Partial algorithm problems for context free languages. In *Information and Control 11*, pages 80–101, 1967.

# Prílohy

## Príloha A - Obsah elektronického média

- balík dk.brics.grammar doplnený o našu implementáciu **dipl_grammar.zip**
- balík dk.brics.string doplnený o našu implementáciu **dipl_string.zip**
- elektronická verzia diplomovejej práce **RuhalovskyJan.pdf**