VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MODIFIKACE HLUBOKÝCH ZÁSOBNÍKOVÝCH AUTOMATŮ

MODIFICATIONS OF DEEP PUSHDOWN AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MAREK PUTALA**
AUTHOR

VEDOUCÍ PRÁCE **ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

164999

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Putala Marek, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Matematické metody
Název:	**Modifikace hlubokých zásobníkových automatů**
Kategorie:	Teoretická informatika
Akademický rok:	2024/25

Zadání:

1. Seznamte se s hlubokými zásobníkovými automaty (Meduna, 2006) a stavovými gramatikami (Kasai, 1970) a nastudujte známé vlastnosti a otevřené problémy.
2. Po konzultaci s vedoucím navrhněte modifikace hlubokých zásobníkových automatů a studujte jejich vlastnosti jako vyjadřovací sílu, uzávěrové vlastnosti a rozhodnutelnost.
3. Pokuste se některé netriviální vlastnosti dokázat formálně, nebo je ověřte experimentálně. Pro experimentální ověření je nutné dané modely implementovat a navrhnout, provést a vyhodnotit experimenty.
4. Práci zhodnoťte a uveďte návrhy na budoucí práci.

Literatura:

- MEDUNA Alexander. Deep Pushdown Automata. *Acta Informatica*, roč. 2006, č. 98, s. 114-124. ISSN 0001-5903.
- MEDUNA Alexander. Finitely Expandable Deep PDAs. *Automata, Formal Languages and Algebraic Systems*. Hong Kong: Hong Kong University of Science and Technology, 2010, s. 113-123. ISBN 981-4317-60-8.
- PUTALA Marek. Částečně paralelní hluboké zásobníkové převodníky a jejich aplikace. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a část bodu 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Křivka Zbyněk, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	11.10.2024

Abstrakt

Cílem této práce bylo seznámit se s hlubokými zásobníkovými automaty a na základě získaných poznatků navrhnout, formálně definovat a implementovat modifikovaný automat, který nevyužívá nevstupní symboly, uvnitř seřazené posloupnosti n zásobníků. Tento redukovaný automat byl analyzován z hlediska rozpoznávací síly a bylo dokázáno, že třída jazyků přijímaných tímto automatem, leží mezi regulárními a stavovými gramatikami, přičemž byly identifikovány charakteristické prvky jazyků, na kterých model ztrácí svou sílu. Dále byl navržen a vyvinut nástroj využívající sady algoritmů definovaných touto práci, pro transformace mezi zmíněnými modely, s cílem funkčního testování formálních výsledků této práce na základě sady testovacích vstupů.

Abstract

The goal of this thesis was to become familiar with deep stack automata and, based on the knowledge gained, to design, formally define and implement a modified automaton that does not use non-input symbols inside an ordered sequence of n stacks. This reduced automaton was analyzed in terms of recognition power and it was proven that the class of languages accepted by this automaton lies between regular and state grammars, while the characteristic elements of languages on which the model loses its power were identified. Furthermore, a tool using the set of algorithms defined in this work was designed and developed for transformations between the mentioned models, with the aim of functionally testing the formal results of this work based on a set of test inputs.

Klíčová slova

Formální jazyky, gramatiky, automaty, Chomského klasifikace, simulace, hluboké zásobníkové automaty, stavové gramatiky, čisté seřazené n-zásobníkové automaty

Keywords

Formal languages, grammars, automata, Chomsky classification, simulation, deep pushdown automata, state grammars, ordered pure deep n-pushdown automata

Citace

PUTALA, Marek. *Modifikace hlubokých zásobníkových automatů*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce ing. Zbyněk Křivka, Ph.D.

Modifikace hlubokých zásobníkových automatů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana ing. Zbyňka Křivky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Marek Putala
19. května 2025

Poděkování

Rád bych poděkoval panu ing. Zbyňku Křivkovi, Ph.D, mému vedoucímu diplomové práce, za odborné vedení, ochotu, rychlou zpětnou vazbu, cenné rady, užitečné připomínky a inspirativní diskuse při zpracování této práce. Vřelé díky patří také mé rodině, přítelkyni a přátelům za podporu a motivaci v průběhu celého studia.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Již od počátků vývoje počítačů se zde objevovala otázka, jak efektivně předat těmto strojům své myšlenky a nápady způsobem, kterému by rozuměly a na základě kterého by mohly následně samostatně vykonávat zadané úlohy. Bylo tedy nutné navrhnout efektivní způsob komunikace, který by byl dostatečně informačně bohatý bez nadbytečných prvků, ale zároveň srozumitelný jak pro počítače, tak pro lidi, kteří s nimi pracují.

Z tohoto důvodu vznikly programovací jazyky, které slouží lidem pracujícím s počítačem, jako nástroj pro přetváření jejich nápadů do sad algoritmů, jež mohou být počítačem přijaty, pochopeny a provedeny.

Aby takový jazyk mohl vzniknout, je potřebné jej formálně definovat prostřednictvím sady pravidel. Nejobecnějším způsobem je tato pravidla ručně sestavit jakožto konečnou množinu specializovaných řetězců. Tento přístup je však neefektivní a hodí se zpravidla pro menší jazyky, protože s rostoucí složitostí daného jazyka se tento způsob stává nepraktickým, až nepoužitelným.

Z tohoto důvodu byly vyvinuty modely, které na základě své vnitřní struktury dokážou přijímat a generovat jazyky, jež mohou být svou velikostí nekonečně velké. Tyto modely jsou označovány jako gramatiky a automaty a dělí se podle své rozpoznávací, či generativní síly do vrstev hierarchie, známé jako Chomského hierarchie. V závislosti na složitosti řešené úlohy se využívají různé typy těchto modelů, tak aby bylo výsledné řešení nejefektivnější.

Tato práce se zaměřuje na tyto modely, konkrétně na ty, které jazyky přijímají. Jejím hlavním cílem je zavedení a návrh nového alternativního modelu automatu, vycházejícího z již existujícího zavedeného, jenž svou rozpoznávací silou spadá mezi silnější modely. Výzkum tohoto modelu zahrnuje nejen zavedení jeho formální definice, ale také analýzu způsobu, jakým řetězce přijímá, zhodnocení jeho rozpoznávací síly a zkoumání jeho podobnosti vůči jiným modelům. Získané poznatky jsou následně otestovány pomocí nástroje, který byl rovněž navržen a implementován v rámci této práce.

V druhé kapitole 2 jsou představeny teoretické základy v oblasti formálních jazyků. Začátek kapitoly vymezuje, co formální jazyky představují, jak se definují a jak jsou podle složitosti rozdělovány do hierarchických tříd. Následovat bude přehled modelů automatů a gramatik, které na základě své vnitřní struktury dokážou tyto jazyky přijímat a generovat.

Ve třetí kapitole 3 se navazuje na koncepty automatů a gramatik představené v předchozí kapitole a podrobněji se zde představují složitější a silnější varianty těchto modelů. Nadnesené jsou zde vzájemné vztahy mezi těmito modely a jak jejich specifické vlastnosti umožňují rozšiřovat třídu jazyků, které dokážou generovat nebo přijímat.

Ve čtvrté kapitole 4, která tvoří hlavní a zároveň výstupní část této práce, je zaveden nový alternativní model automatu, jenž je oproti modelům popsaným v předchozí kapitole

redukován, což mění třídu jazyků, které přijímá. Kapitola obsahuje formální definici tohoto automatu, konfiguraci, typy přechodů a požadavky, které musí jazyk splnit, aby byl tímto automatem přijat. Následně jsou předvedeny algoritmy pro simulaci tohoto modelu pomocí již existujícího modelu a naopak. Závěrečná část kapitoly se zabývá přijímací silou tohoto redukovaného modelu a jeho zařazením do hierarchie formálních modelů.

V páté kapitole 5 se přechází k praktické části práce, kde je nejprve popsán návrh a implementace konzolové aplikace, která umožňuje experimentovat s transformacemi mezi modely nově navrženým redukovaným modelem této práce a již existujícím modelem podle postupů z kapitoly 4. Dále je detailně popsána struktura tohoto nástroje, způsoby jeho spuštění a také představení testovacích vstupů. Závěrem kapitoly jsou demonstrovány provedené experimenty s následným vyhodnocením získaných výsledků.

Kapitola 2

Základy teorie

Pro plné pochopení náplně této práce a jejích výstupů je nejprve nutné zaměřit se na základy teorie formálních jazyků a automatů. Porozumění těmto základům je klíčové zejména pro nováčky v tomto oboru, protože tvoří nezbytný základ pro pochopení dalších kapitol. Následující část práce se proto bude věnovat vysvětlení základních pojmů, jako jsou abecedy, slova, formální jazyky, gramatiky, automaty a Chomského hierarchie.

2.1 Abecedy, slova a jazyky

Tato sekce pojednává o konceptech formálních jazyků a operací nad nimi, což tvoří základ pro pochopení problematiky této práce. Poklady pro tuto podkapitolu byly čerpány z prací [12, str.5-7] a [16, str.8-12]. Pro podrobnější studium této látky se také doporučuje nastudovat [3, str.3-20], kde kromě detailnějšího vysvětlení a ukázkových příkladů jsou k nalezení také demonstrace formou kódu.

Definice 2.1.1 *Abeceda Σ je libovolná neprázdná množina elementů, které jsou nazývány jakožto symboly abecedy.*

Příklad 2.1.1 *Abeceda symbolů $A-\check{Z}, a-\check{z}$, je využívána pro tvoření slov v českém jazyce.*

Abecedy lze dále libovolně sdružovat, rozdělovat nebo odčítat. Platí-li $\Sigma_1 \subseteq \Sigma_2$ tzv., Σ_1 je podmnožinou Σ_2, pak se hovoří o Σ_1 jako o podabecedě abecedy Σ_2.

Definice 2.1.2 *Libovolným konečným zřetězením symbolů abecedy Σ, vzniká řetězec (také slovo či věta) nad abecedou Σ. Jedná-li se o prázdný řetězec, tzv. neobsahuje žádný symbol, je nazýván jako prázdné slovo a označován symbolem ε.*

Příklad 2.1.2 *Je-li $\Sigma = \{0, 1\}$ abeceda, pak ε, 0, 1, 01, 11, 010, jsou některé řetězce nad abecedou Σ.*

Množina všech možných kombinací nad abecedou Σ, včetně prázdného slova ε, je označována jako Σ^*. Množina všech možných kombinací nad abecedou Σ, bez prázdného slova ε, je označována jako $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Definice 2.1.3 *Je-li Σ abeceda a Σ^* množina všech možných kombinací nad abecedou Σ, pak každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad abecedou Σ.*

Příklad 2.1.3 *Nechť Σ je abeceda obsahující symboly {A-Ž, a-ž}. Poté slova {ano, ne, asi} tvoří jazyk L nad abecedou Σ, protože platí, že $L \subseteq \Sigma^*$.*

Pro další definici je zavedena operace $|x|$, kde $x \in \Sigma^*$. Tato operace vrací délku slova x, přičemž platí $|x| \geq 0$. Operace $|x|$ lze rozšířit i na celé jazyky. Nechť L je jazyk nad abecedou Σ, tj. $L \subseteq \Sigma^*$. Potom operace $|L|$ vrací počet řetězců v L (včetně případného neprázdného řetězce ε), přičemž platí $|L| \geq 0$.

Definice 2.1.4 *Nechť L je jazyk nad abecedou Σ. Jestliže jazyk L obsahuje konečný počet řetězců, tj. $|L| = n$ a $n \geq 0$, tak poté se jedná o jazyk konečný. V opačném případě se jedná o jazyk nekonečný.*

Příklad 2.1.4 *Nechť $\Sigma = \{n\}$ je abeceda. Poté jazyk $L_* = \Sigma^*$ je nekonečný, protože obsahuje všechny libovolně dlouhé posloupnosti symbolu n, včetně prázdného slova ε. Naopak, jazyk $L_2 \subseteq \Sigma^*$, který obsahuje slova x s maximální délkou 2, tj. $|x| \leq 2$, je konečný, protože obsahuje pouze slova $\{\varepsilon, n, nn\}$.*

Definice 2.1.5 *Nechť $L_1, L_2 \subseteq \Sigma^*$ jsou dva jazyky, pro které jsou základní operace definovány následovně:*

- ***Sjednocení ($\cup$):*** *Jazyk $L_1 \cup L_2$ obsahuje všechny řetězce, které patří alespoň do jednoho z jazyků L_1 nebo L_2, tj.*

$$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ nebo } x \in L_2\}.$$

- ***Průnik ($\cap$):*** *Jazyk $L_1 \cap L_2$ obsahuje všechny řetězce, které patří současně do jazyků L_1 i L_2, tj.*

$$L_1 \cap L_2 = \{x \mid x \in L_1 \text{ a } x \in L_2\}.$$

- ***Rozdíl ($\setminus$):*** *Jazyk $L_1 \setminus L_2$ obsahuje všechny řetězce, které patří do L_1, ale nepatří do L_2, tj.*

$$L_1 \setminus L_2 = \{x \mid x \in L_1 \text{ a } x \notin L_2\}.$$

- ***Konkatenace ($\cdot$):*** *Jazyk $L_1 \cdot L_2$ obsahuje všechny řetězce, které lze vytvořit konkatenací řetězce $x \in L_1$ a řetězce $y \in L_2$, tj.*

$$L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}.$$

- ***Doplněk jazyka ($\overline{L}$):*** *Doplněk jazyka L obsahuje všechny řetězce nad abecedou Σ, které nepatří do L, tj.*

$$\overline{L} = \Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}.$$

- ***Kleeneho uzávěr (L^*):*** *Kleeneho uzávěr (rovněž iterace) jazyka L obsahuje všechny řetězce, které lze vytvořit z nulového nebo více opakování řetězců z L, tj.*

$$L^* = \bigcup_{i=0}^{\infty} L^i,$$

kde $L^0 = \{\varepsilon\}$ a $L^i = L \cdot L^{i-1}$ pro $i \geq 1$.

- ***Pozitivní iterace (L^+):*** *Pozitivní iterace jazyka L obsahuje všechny řetězce, které lze vytvořit z jednoho nebo více opakování řetězců z L, tj.*

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L \cdot L^*.$$

2.2 Gramatiky

Jelikož jednoduchý výpis všech možných řetězců je často nedostačující, zejména pro složitější jazyky, či dokonce nepoužitelný při práci s nekonečnými jazyky, je proto nutné zavést formální systém, který umožní na základě své interní definice popis celé množiny řetězců daného jazyka. Tato podkapitola se těmito modely zabývá a podklady k jejímu vypracování byly čerpány z [11, str.20-30] a [8, str.34-38].

Gramatiky proto slouží jako základní nástroje pro formální popis jazyků a umožňují definovat strukturu jazyka pomocí pravidel, která určují, jak se jednotlivé symboly skládají do řetězců, čímž umožňují popsat i nekonečné množiny řetězců. Tyto systémy využívají v rámci pravidel gramatiky dva typy symbolů: terminální a neterminální.

- **Terminály** (T) označují konkrétní lexémy, které se objevují v jazyce. Jsou to základní stavební bloky jazyka, které se dále nepřepisují.
- **Neterminály** (N) slouží k reprezentaci syntaktických konstrukcí, jako jsou například výrazy. Jsou definovány pomocí pravidel gramatiky a mohou být dále rozkládány na posloupnosti terminálů a neterminálů.

Gramatiky lze rozdělit podle omezení jejich struktury a generativní síly do několika typů. Nejprve je představen obecnější typ, frázová gramatika, která svou generativní silou stojí na vrcholu hierarchie. Tento způsob klasifikace gramatik je v dalším textu představen a následně se práce zaměřuje na generativně slabší typy, které jsou klíčové pro účely této práce.

Definice 2.2.1 *Frázová gramatika je přepisovací systém, který je definován jakožto čtveřice*

$$G = (N, T, P, S),$$

kde

N *je konečná množina neterminálních symbolů*
T *je konečná množina terminálních symbolů, přičemž* $N \cap T = \emptyset$
P *je konečná relace z* $(N \cup T)^* N (N \cup T)^*$ *do* $(N \cup T)^*$
$S \in N$ *je počáteční symbol (neterminál)*

Pravidla $(x, y) \in P$ jsou uspořádané dvojice, která budou pro lepší čitelnost zapisována ve tvaru $x \to y$. Dále je možné je označovat identifikátorem $r : x \to y$ a namísto tvaru $x \to y \in P$ zapisovat $r \in P$. Dvojice $x \to y \in P$ jsou dále označována jako pravidla přepisovací a mazací. Jestliže platí $y = \varepsilon$, tak se jedná o pravidlo mazací. V opačném případě se jedná o pravidlo přepisovací. Levá strana pravidla x obsahuje alespoň jeden neterminál, tj. $(N \cup T)^* N (N \cup T)^*$. Pravá strana se skládá z libovolné posloupnosti terminálů a neterminálů, tj. $(N \cup T)^*$.

Gramatiky při generování slov využívají proces zvaný derivace. Jedná se o postup, při kterém se syntaktické konstrukce jazyka postupně vytvářejí pomocí pravidel gramatiky. Derivace začíná od speciálního neterminálního symbolu S, označovaného jako počáteční symbol, a probíhá v několika postupných krocích. Každý krok derivace je symbolicky označován symbolem $\Rightarrow$ a spočívá v nahrazení neterminálu řetězcem terminálů a neterminálů, podle definovaných pravidel gramatiky. Proces derivace končí, jakmile derivované slovo obsahuje pouze terminální symboly.

2.3 Chomského klasifikace gramatik

Nyní je vhodné se zaměřit na klasifikaci gramatik, známou také jako Chomského hierarchie, která zkoumá, jaké druhy pravidelností mohou vznikat z gramatik omezených různými podmínkami na jejich strukturu. Tato hierarchie rozlišuje čtyři typy gramatik, které jsou seřazeny podle síly přepisovacích pravidel: typ 0, typ 1, typ 2 a typ 3, od nejsilnějších po nejslabší. Vzhledem k tomu, že se jedná o hierarchii, platí, že gramatiky nebo jazyky ve vyšších úrovních mohou být zároveň klasifikovány v nižších úrovních. Například gramatika typu 2 je zároveň gramatikou typu 3. Tato podkapitola vychází z textů [5] a [16, str.16-20].

- **Typ 0 (neomezené gramatiky)**: Nejobecnější typ gramatik, které mohou generovat libovolné formální jazyky. Tyto gramatiky nejsou omezeny a mohou používat jakékoliv přepisovací pravidlo. Proto se nazývají neomezené.

  $$\alpha \to \beta, \quad \alpha \in (N \cup T)^* N (N \cup T)^*, \quad \beta \in (N \cup T)^*$$

  Jazyky generované těmito gramatikami jsou akceptovatelné Turingovými stroji

- **Typ 1 (kontextové gramatiky)**: Tyto gramatiky jsou oproti typu 0 omezeny tím, že v rámci pravidla musí být pravá strana alespoň stejně dlouhá jako levá strana. To znamená, že gramatika bude splňovat následující omezení,

  $$\text{Omezení 1:} \quad \alpha A \beta \to \alpha \gamma \beta \quad \text{kde } \gamma \text{ je neprázdný řetězec}$$

  To znamená že při generování nemůže dojít ke zkrácení generovaného řetězce. Jedinou výjimkou je pravidlo pro generování prázdného řetězce,

  $$S \to \varepsilon$$

  Gramatiky typu 1 se nazývají kontextové, protože jejich samotná pravidla provádějí přepis neterminálu na základě kontextu. To znamená že neterminál A může být nahrazen řetězcem γ pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α. Pravidla jsou tedy ve tvaru,

  $$\alpha A \beta \to \alpha \gamma \beta, \quad A \in N, \quad \alpha, \beta \in (N \cup T)^*, \quad \gamma \in (N \cup T)^+$$

  nebo

  $$S \to \varepsilon$$

- **Typ 2 (bezkontextové jazyky)**: Tento typ gramatik je oproti typu 1 omezen tím, že v pravidlech již není kontrolován kontext, ve kterém se přepisovaný neterminál nachází. Gramatika pracuje s konečným počtem neterminálů a pravidel, což znamená, že každý neterminál může být rozepsán podle jednoho z konečně mnoha definovaných pravidel, které tato gramatika obsahuje. Zajímavé je, že na rozdíl od kontextových gramatik, bezkontextové gramatiky smí obsahovat ε-pravidla, což kromě využití ε-pravidla přepisující počáteční symbol jako u kontextových gramatik dává možnost definování ε-pravidel i pro jiné neterminály. Tato gramatika tedy splňuje následující omezení,

  $$\text{Omezení 2:} \quad A \to \gamma \quad \text{kde } \gamma \text{ je neprázdný řetězec}$$

  V tomto případě je opět jedinou výjimkou pravidlo pro generování počátečního neterminálu na prázdný řetězec,

  $$S \to \varepsilon$$

Tato možnost se obecně doporučuje, protože kromě lepší čitelnosti také zajišťuje konzistentní generování řetězců, což je výhodné při praktickém využití.

Zahrnutím druhé možnosti (ε-pravidla i pro jiné neterminály) získáme pravidla, která budou formálně mít tvar,

$$A \to \gamma \quad \text{kde } A \in N,\ \gamma \in (N \cup T)^*$$

Každou bezkontextovou gramatiku s více než jedním ε-pravidlem lze transformovat tak, že bude obsahovat nejvýše jedno ε-pravidlo, aniž by došlo ke změně jazyka, který je touto gramatikou generován. Z tohoto důvodu je v takové gramatice povoleno mít více ε-pravidel, protože tato transformace nezmění generované řetězce.

- **Typ 3 (regulární gramatiky)**: Nejslabší typ gramatik, které generují jednoduché jazyky. Tyto gramatiky jsou oproti typu 2 omezeny tím, že již nepomohou v rámci pravidla rozepisovat neterminál na libovolnou posloupnost terminálů a neterminálů. Jsou tedy pravidla, kde neterminál A může být nahrazen řetězcem γ pouze tehdy, platí-li $\gamma \in x$ nebo $\gamma \in xB$, kde $x \in T$, $B \in N$. Tímto vzniká další omezení kterého tato gramatika nabývá,

$$\text{Omezení 3:} \quad A \to x \quad \text{nebo} \quad A \to xB \quad \text{kde } x \in T,\ B \in N$$

Lze si všimnout, že gramatiky, které splňují Omezení 2, zároveň splňují i Omezení 1, a gramatiky splňující Omezení 3 splňují obě předchozí omezení. Tímto vzniká hierarchie gramatik se čtyřmi vrstvami.

- Typ 0, který je neomezený a tedy nesplňuje žádná omezení
- Typ 1, který je kontextový a splňuje pouze první omezení
- Typ 2, který je bezkontextový a splňuje první a druhé omezení
- Typ 3, který je nejslabší a splňuje všechny tři omezení

2.4 Automaty

Předcházející sekce se zabývala klasifikací modelů gramatik, které na základě interně definovaných pravidel slouží pro formální popis a generování formálních jazyků, které mohou být nekonečné. Představeno bylo, jak postupné redukování omezení těchto modelů navyšuje sílu a komplexitu jazyků, které popisují. Tato sekce se zaměřuje na jiný druh modelu, který jazyky naopak přijímá, známý pod pojmem automaty, které slouží k rozpoznávání řetězců patřících do daného jazyka.

Automaty, se podobně jako gramatiky dělí podle Chomského hierarchie do skupin. Toto dělení odráží, jak složité formální jazyky dokážou automaty na základě své vnitřní struktury přijmout a díky tomu rozlišit, které vstupní řetězce do daného jazyka patří a které nikoli.

Nyní bude následovat postupné představení těchto modelů od jednodušších k složitějším, přičemž jejich přijímací síla bude s každým dalším modelem narůstat. Podklady pro tuto sekci vycházely z [10, str.113] a [7, kap. 2].

Definice 2.4.1 *Konečný automat (KA) je přepisovací systém, který je definován jakožto pětice*

$$M_{KA} = (Q, \Sigma, R, s, F),$$

kde

- Q *je konečná množina stavů;*
- Σ *je vstupní abeceda (konečná množina symbolů);*
- $R \subseteq (Q, \Sigma \cup \{\varepsilon\}, Q)$ *je konečná množina pravidel;*
- $s \in Q$ *je počáteční stav;*
- $F \subseteq Q$ *je množina akceptujících (koncových) stavů.*

Pravidla jsou trojice $(q, x, p) \in R$, které budou v rámci této práce zapisovány ve tvaru $qx \to p \in R$. Konfigurace KA je dvojice $\chi = (q, \omega)$, kde $q \in Q$ je aktuální stav a $\omega \in \Sigma^*$ je dosud nepřečtený řetězec na vstupní pásce. Nechť X_{KA} představuje množinu všech konfigurací KA. Nechť $(q, x\omega), (p, \omega) \in X_{KA}$ je konfigurace a symbol $\vdash$ označující přechod. Potom přechod mezi konfiguracemi je ve tvaru $(q, x\omega) \vdash (p, \omega)$ za použití pravidla $qx \to p \in R$. Pro zachování přehlednosti je vhodné přechod doplnit o použité pravidlo,

$$(q, x\omega) \vdash (p, \omega) \quad [qx \to p]$$

Opakovaným prováděním přechodů přechází automat mezi stavy a čte řetězec ze vstupní pásky zleva doprava tím, že posouvá čtecí hlavu po jednotlivých znacích, dokud nepřečte celý řetězec a dokud se nedostane do některého z koncových stavů. V tomto případě platí, že automat vstupní řetězec přijímá. Množinu všech řetězců, které automat přijímá, je označována jako jazyk $L(M_{KA})$ přijímaný automatem M_{KA}. Tento jazyk je definován jako,

$$L(M_{KA}) = \{w \mid w \in \Sigma^*, (s, w) \vdash^* (f, \varepsilon), f \in F\}$$

Přijímací síla konečných automatů odpovídá generativní síle regulárních gramatik, spadající do třídy T3 v Chomského hierarchii. To znamená, že pro libovolnou regulární gramatiku existuje konečný automat a naopak. Platí tedy,

$$L(M_{KA}) = L(G_{RG})$$

Definice 2.4.2 *Zásobníkový automat (ZA) je přepisovací systém, který je definován jakožto sedmice*

$$M_{ZA} = (Q, \Sigma, \Gamma, R, s, S, F),$$

kde

- Q *je konečná množina stavů;*
- $\Sigma \subset \Gamma$ *je vstupní abeceda (konečná množina symbolů);*
- Γ *je zásobníková abeceda (konečná množina symbolů);*

- $R \subseteq (\Gamma^*, Q, \Sigma \cup \{\varepsilon\}, \Gamma^*, Q)$ *je konečná množina pravidel;*
- $s \in Q$ *je počáteční stav;*
- $S \in \Gamma \setminus \Sigma$ *je počáteční neterminál;*
- $F \subseteq Q$ *je množina akceptujících (koncových) stavů.*

Pravidla jsou pětice $(N, q, x, v, p) \in R$, které budou v rámci této práce zapisována ve tvaru $Nqx \to vp \in R$. Konfigurace ZA je trojice $\chi = (q, \omega, \beta)$, kde $q \in Q$ je aktuální stav, $\omega \in \Sigma^*$ je dosud nepřečtený řetězec na vstupní pásce a $\beta \in \Gamma^*$ je aktuální stav zásobníku. Nechť X_{ZA} představuje množinu všech konfigurací ZA. Platí, že $(q, x\omega, N\alpha), (p, \omega, v\alpha) \in X_{ZA}$ je konfigurace a $Nqx \to vp$ je pravidlo. Potom přechod mezi konfiguracemi bude ve tvaru,

$$(q, x\omega, N\alpha) \vdash (p, \omega, v\alpha) \quad [Nqx \to vp]$$

Zásobníkové automaty ZA se dělí na tři podtypy podle podmínek pro přijímání vstupních řetězců. Zásobníkový automat M_{ZA} přijímá vstupní řetězec, pokud byl během zpracování načten vstupní řetězec, zásobník byl vyprázdněn a automat přešel do jednoho z množiny koncových stavů. Jazyk přijímaný tímto automatem definujeme jako,

$$L(M_{ZA}) = \{w \mid w \in \Sigma^*, (s, w, S) \vdash^* (f, \varepsilon, \varepsilon), f \in F\}$$

Zásobníkový automat ${}_fM_{ZA}$ přijímá vstupní řetězec, pokud byl během zpracování načten vstupní řetězec a automat přešel do jednoho z množiny koncových stavů. Jazyk přijímaný tímto automatem je definován jako,

$$L({}_fM_{ZA}) = \{w \mid w \in \Sigma^*, (s, w, S) \vdash^* (f, \varepsilon, v), f \in F, v \in \Gamma^*\}$$

Zásobníkový automat ${}_pM_{ZA}$ přijímá vstupní řetězec, pokud byl během zpracování načten vstupní řetězec a zásobník byl vyprázdněn. Jazyk přijímaný tímto automatem je definován jako,

$$L({}_pM_{ZA}) = \{w \mid w \in \Sigma^*, (s, w, S) \vdash^* (p, \varepsilon, \varepsilon), p \in Q\}$$

Přijímací síla zásobníkových automatů odpovídá generativní síle bezkontextových gramatik, spadajících do třídy T2 v Chomského hierarchii. To znamená, že pro libovolnou bezkontextovou gramatiku existuje zásobníkový automat a naopak. Platí tedy,

$$L(M_{ZA}) = L({}_fM_{ZA}) = L({}_pM_{ZA}) = L(G_{BKG})$$

Zároveň platí, že libovolný jazyk přijatý konečným automatem je přijatý také zásobníkovým automatem, ale ne každý jazyk přijatý zásobníkovým automatem je přijatý také konečným automatem. To znamená, že třída jazyků přijímaná konečnými automaty je vlastní podmnožinou třídy jazyků přijímaných zásobníkovými automaty. Formálně je tento vztah zapsán jako

$$L(M_{KA}) = L(G_{RG}) \subset L(M_{ZA}) = L({}_fM_{ZA}) = L({}_pM_{ZA}) = L(G_{BKG})$$

Kapitola 3

Modifikované typy automatů a gramatik

V minulé kapitole byl představen koncept Chomského hierarchie, v níž bylo ukázáno, že jazykově generující a přijímací modely, známé jako gramatiky a automaty, jsou podle své síly rozděleny do čtyř tříd od nejslabších T3 až po nejsilnější T0.

Tato kapitola pojednává o nových modelech, které jsou se svou přijímací, nebo generativní silou řazeny mezi třídy T2 a T1 podle Chomského hierarchie a jejichž pochopení je nezbytné pro pochopení výstupu této práce.

3.1 Stavové gramatiky

Tato sekce pro niž byly podklady čerpány z [6, 4], představuje alternativní typ jazykově generujícího modelu, stavové gramatiky, které rozšiřují bezkontextové gramatiky (třída T2) s ε-pravidly, o stavový mechanismus. Tento mechanismus dále reguluje derivační proces tím, že při aplikaci pravidel bere v potaz aktuální vnitřní stav gramatiky. Jazyky generované stavovými gramatikami se nazývají stavové jazyky, přičemž je dokázáno, že třída stavových gramatik je identická s třídou kontextových gramatik.

Zavedením dodatečných omezení na stavové gramatiky lze definovat nekonečnou hierarchii podtříd $\mathcal{L}_\infty, \mathcal{L}_\in, \ldots, \mathcal{L}_\infty, \mathcal{L}_\omega$ stavových jazyků.

Definice 3.1.1 *Stavová gramatika (SG) je přepisovací systém, který je definován jakožto šestice*

$$G_{SG} = (Q, N, T, P, s, S),$$

kde

- Q *je konečná množina stavů;*
- N *je konečná množina neterminálů;*
- T *je konečná množina terminálů;*
- $P \subseteq (Q \times N \times Q \times (N \times T)^*$ *je konečná množina pravidel;*
- $s \in Q$ *je počáteční stav;*

- $S \in N$ *je počáteční neterminál.*

Pravidla jsou čtveřice $(p, \xi, q, v) \in P$, které se využívají při derivacích a jsou zapisována ve tvaru $(p, \xi) \rightarrow (q, v)$. Neterminál $\xi \in N$ je přepisovatelný na posloupnost symbolů $v \in (N \times T)^*$ ve stavu $p \in Q$, pokud:

1. Existuje pravidlo $(p, \xi, q, v) \in P$.
2. Neterminál ξ je nejlevější přepisovatelný neterminál při aktuálním vnitřním stavu a aplikovatelných pravidlech.

Po vykonání pravidla, rovněž přejde gramatika do stavu $q \in Q$. Použitím pravidla provede gramatika derivační krok, který je značen symbolem $\Rightarrow$. Dále $\Rightarrow^*$ označuje posloupnost derivačních kroků o délce větší než 0 a > 0 a $\Rightarrow^+$ označuje posloupnost jednoho nebo více derivačních kroků. Jazyk generovaný stavovou gramatikou G_{SG} je definován jako

$$L(G_{SG}) = \{w \in T^* | (s, S) \Rightarrow^* (q, w), \text{ kde } q \in Q\}$$

a je nazýván pod pojmem stavový jazyk $L(G_{SG})$ generovaný stavovou gramatikou G_{SG}.

Stavová gramatika G_{SG} je stupně $n \geq 1$ pouze tehdy, když platí $L(G_{SG}; n) = L(G)$. Stavový jazyk L je stupně n, pokud existuje stavová gramatika G_{SG} stupně n generující tento jazyk $L = L(G_{SG})$. Pokud $L \neq L(G_{SG})$ pro libovolný stupeň $n \geq 1$, tak potom je gramatika G_{SG} a jazyk L stupně ∞.

Vzniká hierarchie stavových jazyků, pro které platí $\forall n \geq 1 : \mathcal{L}_n \subseteq \mathcal{L}_{n+1}$. Celá třída stavových jazyků je označována jako $\mathcal{L}_\omega$, pro kterou platí $\cup_{n=1}^{\infty} \mathcal{L}_n \subseteq \mathcal{L}_\omega$. Celá třída jazyků $\mathcal{L}_{BG}$ generovaná bezkontextovými gramatikami je identická třídě jazyků $\mathcal{L}_1$ generované stavovými gramatikami stupně 1.

Nyní následuje příklad stavové gramatiky, která generuje jazyk $L(G_{SG}) = \{a^i b^j c^k | i = j + k, j \leq k, i \geq 1\}$

Příklad 3.1.1 *Nechť* $G_{SG} = (Q, \{A, C\}, \{a, b, c\}, P, s, S)$ *jde stavová gramatika, kde* $Q = \{s, p, q, f, g, h, k, j, v, n\}$ *a množina pravidel* P *nabývá,*

$$\begin{array}{lllll}
1 & : & s(S) & \rightarrow & q(AC), \\
2 & : & q(A) & \rightarrow & p(aA), \\
3 & : & p(C) & \rightarrow & q(Cc), \\
4 & : & q(A) & \rightarrow & g(a), \\
5 & : & g(C) & \rightarrow & f(c), \\
6 & : & s(S) & \rightarrow & h(AC), \\
7 & : & h(A) & \rightarrow & k(aaA), \\
8 & : & k(A) & \rightarrow & j(Ab), \\
9 & : & j(C) & \rightarrow & h(Cc), \\
10 & : & k(A) & \rightarrow & v(Ab), \\
11 & : & v(C) & \rightarrow & q(Cc), \\
12 & : & h(A) & \rightarrow & n(aab), \\
13 & : & n(C) & \rightarrow & f(c)
\end{array}$$

Pro řetězec aaaaabbccc generovaný touto gramatikou by sekvence derivací vypadala následovně,

$$
\begin{array}{lcll}
(s,S) & \Rightarrow & (h,AC) & [s(S) \to q(AC)] \\
 & \Rightarrow & (k,aaAC) & [h(A) \to k(aaA)] \\
 & \Rightarrow & (j,aaAbC) & [k(A) \to j(Ab)] \\
 & \Rightarrow & (h,aaAbCc) & [j(C) \to h(Cc)] \\
 & \Rightarrow & (k,aaaaAbCc) & [h(A) \to k(aaA)] \\
 & \Rightarrow & (v,aaaaAbbCc) & [k(A) \to v(Ab)] \\
 & \Rightarrow & (q,aaaaAbbCcc) & [v(C) \to q(Cc)] \\
 & \Rightarrow & (g,aaaaabbCcc) & [q(A) \to g(a)] \\
 & \Rightarrow & (f,aaaaabbccc) & [g(C) \to f(c)]
\end{array}
$$

3.2 Hluboký zásobníkový automat

Tato sekce se zaměřuje na definici a přijímací sílu hlubokých zásobníkových automatů (DPDAs). Tyto automaty, na rozdíl od klasických zásobníkových automatů, dokážou operovat hlouběji uvnitř zásobníku způsobem, že odkazují na indexy nevstupních symbolů hlouběji ponořených v zásobníku. To zvyšuje jejich přijímací sílu oproti modelu ZA, který má přístup pouze k symbolům na vrcholu zásobníku.

Toto rozšíření modelu dává vzniknout novému typu automatu, jehož přijímací síla je podobná generativní síle regulovaných bezkontextových gramatik bez ε-pravidel. To znamená, že hluboké zásobníkové automaty jsou silnější než běžné zásobníkové automaty, ale slabší než kontextové gramatiky. Tím vzniká nekonečná hierarchie tříd jazyků, generovaných n-limitovanými stavovými gramatikami, které tento automat dokáže přijmout.

Podklady pro tuto podkapitolu byly čerpány z [3, str.545–562] a [9], kde se pro detailnější studium nacházejí také důkazy o ekvivalenci se stavovými gramatikami.

Definice 3.2.1 *Hluboký zásobníkový automat stupně n je přepisovací systém, který je definován jakožto osmice*

$$
{}_nM = (Q, \Sigma, \Gamma, R, s, S, F),
$$

kde

- Q *je konečná množina vnitřních stavů;*
- Σ *je vstupní abeceda a* $\Sigma \subseteq \Gamma$*;*
- Γ *je zásobníková abeceda a* $\Gamma \cup \{\#\}$ *je dno;*
- $R \subseteq (\mathbb{N} \times Q \times (\Gamma \setminus (\Sigma \cup \{\#\})) \times Q \times (\Gamma \setminus \{\#\})^+) \cup (\mathbb{N} \times Q \times \{\#\} \times Q \times (\Gamma \setminus \{\#\})^*\{\#\})$ *je konečná množina pravidel;*
- $s \in Q$ *je počáteční stav;*
- $S \in \Gamma$ *je počáteční symbol na zásobníku;*
- $F \subseteq Q$ *je konečná množina koncových stavů.*

Pro lepší čitelnost budou pravidla v rámci této práce zapisována ve tvaru $q(nA) \to p(v) \in R$ namísto $(n, q, A, p, v) \in R$, kde q je stav, ve kterém se má automat aktuálně nacházet, p je stav, do kterého má automat následně přejít, n je index přepisovaného symbolu, A přepisovaný symbol, a v posloupnost symbolů, na které bude původní symbol přepsán.

Konfigurace hlubokých zásobníkových automatů je zapisována jako trojice $\chi = (Q \times \Sigma^* \times (\Gamma \setminus \{\#\})^*\{\#\})$ kde Q představuje vnitřní stav automatu, Σ^* reprezentuje posloupnost znaků vstupního řetězce, které ještě nebyly přijaty. Dále výraz $(\Gamma \setminus \{\#\})^*\{\#\}$ vyjadřuje aktuální stav zásobníku, kde # je zásobníkové dno.

Nechť X_M označuje množinu všech možných konfigurací, kterých může dosáhnout hluboký zásobníkový automat. Existují dvě konfigurace $x, y \in X_M$. Dále existují dva typy přechodů, které může automat vykonávat `pops` a *expands*. Pokud automat provede porovnání vstupního symbolu na vrcholu zásobníku s aktuálně čteným symbolem na vstupní pásce a dojde ke shodě, symbol je vyjmut ze zásobníku a čtecí hlava na vstupní pásce se posune na následující symbol. Tento krok je označován jako přechod *pops* a v rámci této práce bude symbolicky zapisován jako,

$$x \vdash_p y$$

přičemž platí, $(q, au, az) \vdash_p (q, u, z)$, kde $q \in Q$, $a \in \Sigma$, $u \in \Sigma^*$ a $z \in (\Gamma \setminus \{\#\})^*\{\#\}$. Dalším typ přechodu *expands*, vykonává přechod z konfigurace x do y, během kterého dochází k přepisu nevstupního symbolu uvnitř zásobníku na libovolnou posloupnost symbolů totální abecedy. Operace je symbolicky zapisována jako,

$$x \vdash_e y$$

přičemž platí, $(q, w, uAz) \vdash_e (p, w, uvz)$, kde $q, p \in Q$, $w \in \Sigma^*$, $A \in N$, $u \in \Sigma^*(\Sigma^* N \Sigma^*)^m$, $v \in \Gamma^*$, $z \in (\Gamma \setminus \{\#\})^*\{\#\}$, $N \in \Gamma \setminus \Sigma$, $m \geq 0$. Pro tento přechod musí být použito pravidlo $m(qA) \to p(v)$. Samotný přechod *expands* je proto vhodné doplnit o informaci o použitém pravidle,

$$(q, w, uAz) \vdash_e (p, w, uvz)[q(mA) \to p(v)]$$

Nechť M je hluboký zásobníkový automat hloubky $n \in \mathbb{N}$, kde $n \geq 1$. Jazyk L je přijímán automatem ${}_nM$ pouze tehdy, pokud ${}_nM$ přijme všechny řetězce, které do něj patří. Tento model přijímá vstupní řetězec za předpokladu, že byl načten celý vstupní řetězec, aktuální vnitřní stav patří do množiny koncových stavů a zásobník automatu je vyprázdněn, tj. obsahuje pouze jeden speciální symbol #, označující dno zásobníku. Přijímaný jazyk je formálně definován jakožto,

$$L({}_nM) = \{w \in \Sigma^* | (s, w, S\#) \vdash^* (f, \varepsilon, \#), \text{ platí-li } f \in F\}$$

Jazyk L může být také přijímán prázdným zásobníkem, přičemž tento jazyk je formálně označován jakožto,

$$E({}_nM) = \{w \in \Sigma^* | (s, w, S\#) \vdash^* (f, \varepsilon, \#), \text{ platí-li } f \in F\}$$

Pro libovolné $n \geq 1$, hluboký zásobníkový automat ${}_nM$ definuje třídy jazyků přijímané tímto automatem v hloubce i, kde $1 \geq i \geq n$. Podobně může automat přijímat třídy jazyků přijímané automatem ${}_nM$ s prázdným zásobníkem v hloubce i, kde $1 \geq i \geq n$.

Nyní následuje příklad fungování tohoto modelu a jeho postupu při zpracování vstupního řetězce v jazyce, který byl představen u stavových gramatik. Automat je navržen tak, aby rozpoznával řetězce patřící do jazyka $L = \{a^i b^j c^k | i = j + k, j \leq k, i \geq 1\}$.

Příklad 3.2.1 *Nechť* $M_{nDPDA} = (Q, \{a,b,c\}, \{a,b,c,A,C,\#\}, R, s, S, \{f\})$ *je hluboký zásobníkový automat, kde* $Q = \{s,p,q,f,g,h,k,j,v,n\}$ *a konečné množině pravidel* R *náleží,*

$$
\begin{array}{llcl}
1 & : & s(1S) & \rightarrow & q(AC) \\
2 & : & q(1A) & \rightarrow & p(aA) \\
3 & : & p(2C) & \rightarrow & q(Cc) \\
4 & : & q(1A) & \rightarrow & g(a) \\
5 & : & g(1C) & \rightarrow & f(c) \\
6 & : & s(1S) & \rightarrow & h(AC) \\
7 & : & h(1A) & \rightarrow & k(aaA) \\
8 & : & k(1A) & \rightarrow & j(Ab) \\
9 & : & j(2C) & \rightarrow & h(Cc) \\
10 & : & k(1A) & \rightarrow & v(Ab) \\
11 & : & v(2C) & \rightarrow & q(Cc) \\
12 & : & h(1A) & \rightarrow & n(aab) \\
13 & : & n(1C) & \rightarrow & f(c)
\end{array}
$$

Pro řetězec $aaaaaabbcccc$ *přijímaný automatem* M_{nDPDA} *by sekvence přechodů vypadala následovně,*

$$
\begin{array}{lcll}
(s, aaaaaabbcccc, S\#,) & \vdash_e & (h, aaaaaabbcccc, AC\#) & [6] \\
 & \vdash_e & (k, aaaaaabbcccc, aaAC\#) & [7] \\
 & \vdash_p & (k, aaaabbcccc, AC\#) & \\
 & \vdash_e & (j, aaaabbcccc, AbC\#) & [8] \\
 & \vdash_e & (h, aaaabbcccc, AbCc\#) & [9] \\
 & \vdash_e & (k, aaaabbcccc, aaAbCc\#) & [7] \\
 & \vdash_p & (k, aabbcccc, AbCc\#) & \\
 & \vdash_e & (v, aabbcccc, AbbCc\#) & [10] \\
 & \vdash_e & (q, aabbcccc, AbbCcc\#) & [11] \\
 & \vdash_e & (p, aabbcccc, aAbbCcc\#) & [2] \\
 & \vdash_p & (p, abbcccc, AbbCcc\#) & \\
 & \vdash_e & (q, abbcccc, AbbCccc\#) & [3] \\
 & \vdash_e & (g, abbcccc, abbCccc\#) & [4] \\
 & \vdash_p & (g, cccc, Cccc\#) & \\
 & \vdash_e & (f, cccc, cccc\#) & [5] \\
 & \vdash_p & (f, , \#) &
\end{array}
$$

3.3 Ekvivalence stavových gramatik a hlubokých zásobníkových automatů

V této sekci je předvedeno, že třída jazyků generovaných stavovými gramatikami (SG) je ekvivalentní třídě jazyků přijímaných hlubokými zásobníkovými automaty (DPDA) a zároveň je zde nastíněn způsob dokázání této ekvivalence. Z této rovnosti mezi modely vyplývá následující tvrzení pro $\forall n \geq 1$:

$$\mathcal{L}(G_{SG}; n) = \mathcal{L}({}_nM_{DPDA}) \implies$$

$$\mathbf{DEPDA}_n = \mathbf{DPDA}_n \subset \mathbf{DEPDA}_{n+1} = \mathbf{DPDA}_{n+1} \subset \mathbf{KG}$$

Toto tvrzení lze dokázat tím, že bude existovat způsob, jak libovolnou stavovou gramatiku G_{SG} stupně $n \geq 1$ převést na hluboký zásobníkový automat ${}_nM_{DPDA}$, přičemž platí

$L(G_{SG}; n) = L({}_nM_{DPDA})$. Zároveň je třeba ukázat, že existuje i opačná transformace z ${}_nM_{DPDA}$ na G_{SG}.

V rámci této podkapitoly bude předvedena jedna z těchto transformací, konkrétně transformace ${}_n$DPDAs podle zadaných stavových gramatik SG_n stupně $n \geq 1$. Tento text vychází z práce [3, str.545–562], přičemž se pro podrobnější studium doporučuje přečíst tuto práci.

Definice 3.3.1 *Nechť $G_{SG} = (Q, N, T, P, s, S)$ je stavová gramatika stupně $n \geq 1$. Mějme homomorfismus f nad $(N \cup T \cup \{\#\})^*$ ve tvaru $f(A) = A$, kde $A \in \{\#\} \cup N$, $f(a) = \varepsilon$ a $a \in V$ jsou terminály. Dále máme DPDA hloubky n ve tvaru*

$$ {}_nM = (Q, \Sigma, \Gamma \cup \{\#\}, R, s, S, \{\$\}),$$

kde $Q = \{S, \$\} \cup \{\langle p, u\rangle | p \in Q, u \in prefix(N^*\{\#\}^n, n), |u| \leq n\}$, a v množině R jsou pravidla, která se konstruují následovně:

1. $\forall((p, S) \to (q, x)) \in P \Rightarrow (s(1S) \to \langle p, S\rangle(S)) \in R$, *kde* $p, q \in Q$, $x \in (N \cup T)^+$
2. $(((p, A) \to (q, x)) \in P) \wedge (\langle p, uAv\rangle \in Q) \Rightarrow \langle p, uAv\rangle(|uA|A) \to \langle q, prefix(uf(x)v, n)\rangle(x)$ $\in R$, *kde* $p, q \in Q, A \in N, x \in (N \cup T)^+, u \in N^*, v \in N^*\{\#\}^*, |uAv| = n, p \notin G_{states}(u)$
3. $R = R \cup \{|uA|\langle p, uv\rangle A \to \langle p, uAv\rangle, |uA|\langle p, uv\rangle\# \to \langle p, uv\#\rangle\#\}$, *pokud* $A \in N$, $p \in Q$, $u \in N^*$, $v \in \{\#\}^*$, $|uv| \leq n - 1$, $p \notin G_{states}(u)$
4. $\forall q \in Q : R = R \cup \{1\langle q, \#^n\rangle\# \to \$\#\}$

S použitím této konstrukce lze simulovat libovolnou stavovou gramatiku stupně $n \geq 1$ do podoby hlubokého zásobníkového automatu se zanořením n stavů, přičemž definovaný jazyk mezi těmito modely bude ekvivalentní.

Lemma 3.3.1 *Pro každou stavovou gramatiku G_n, kde $n \geq 1$, existuje hluboký zásobníkový automat ${}_nM$ s hloubkou zanoření n, přičemž platí $L(G; n) = L({}_nM)$*

Dále bylo dokázáno, že existuje sada algoritmů, pomocí kterých lze simulovat libovolný model DPDA stupně $n \geq 1$ do podoby simulující stavové gramatiky G_n. Platí tedy

Lemma 3.3.2 *Pro každý hluboký zásobníkový automat ${}_nM$ s hloubkou zanoření $n \geq 1$ existuje stavová gramatika G_n, přičemž platí $L(G; n) = L({}_nM)$*

Je tedy dáno, že s možností simulace mezi libovolnými modely G_n a ${}_nM$, kde $n \geq 1$, přičemž platí $L(G; n) = L({}_nM)$, se síla těchto modelů rovná a tedy přijímají stejnou třídu jazyků. Platí tedy věta,

Věta 3.3.1 $\boldsymbol{DPDA_n = SG_n}$, *pro* $n \geq 1$

Kapitola 4

Čistý hluboký seřazený n-zásobníkový automat

Tato kapitola, která tvoří hlavní a zároveň výstupní část této práce, představuje zavedení nového alternativního typu hlubokého zásobníkového automatu. Tento model, na rozdíl od původního automatu s jedním zásobníkem, pracuje s n zásobníky s pevným pořadím a oproti původnímu modelu pracuje pouze se vstupními symboly uvnitř zásobníků, bez možnosti využití nevstupních symbolů. Každý zásobník obsahuje speciální oddělovač, který označuje jeho dno. Definice symbolu pro dno zásobníku je integrální součástí specifikace samotného automatu.

Cílem této kapitoly je výzkum, zda a případně jak se změní přijímací síla tohoto modelu a jakou třídu jazyků dokáže přijímat v rámci Chomského hierarchie. Nejprve následuje definice nového modelu.

4.1 Definice

Definice 4.1.1 *Čistý hluboký seřazený n-zásobníkový automat (${}_nOPDPDA$) je alternativní verze hlubokého zásobníkového automatu (DPDA) a definujeme jej jakožto sedmici,*

$$M_{{}_nOPDPDA} = (Q, \Sigma, R, s, \tau, \xi, F),$$

kde

- Q *je konečná množina vnitřních stavů;*
- Σ *je totální abeceda a* $\xi \notin \Sigma$*;*
- $R \subseteq (Q \times (\mathbb{N} \times \Sigma) \times \mathbb{N} \times Q \times \Sigma^*) \cup (Q \times (\{\xi\} \times \mathbb{N}) \times Q \times (\Sigma \setminus \{\xi\})^*\{\xi\}) \cup (Q \times Q)$ *je konečná množina pravidel;*
- $s \in Q$ *je počáteční stav;*
- ξ *je speciální znak, označující dna zásobníků (a* $\{\xi\}$ *je tedy jednoprvková množina);*
- τ *je počáteční symbol v zásobnících a* $\tau \in \Sigma$*;*
- $F \subseteq Q$ *je konečná množina koncových stavů.*

Pro zjednodušení a lepší čitelnost budou pravidla v rámci této práce zapisována, ve tvaru $q(ka)j \to p(v)$ namísto $(q, (k, a), j, p, v)$, kde q je stav, ve kterém se automat aktuálně nachází, p je cílový stav, k je index přepisovaného symbolu, a přepisovaný symbol, j index zásobníku, ve kterém má být provedena změna a v posloupnost symbolů, na které bude původní symbol přepsán. Modely ${}_n$OPDPDAs obsahují dva typy přechodových pravidel: *pops* a *expands*.

Pravidla typu *expands* je nutné explicitně definovat v množině pravidel R. Na počátku běhu automatu obsahují zásobníky speciální symbol ξ označující dno a také počáteční vstupní symbol $\tau \in \Sigma$, z kterého se bude dále odvíjet generování posloupností v zásobníku. Symboly uvnitř zásobníku je možné mazat pomocí ε-pravidel.

Pravidla typu *pops* jsou přirozenou součástí modelů ${}_n$OPDPDAs a není nutné je explicitně definovat v rámci množiny R. Pro každý vstupní symbol $u \in \Sigma$ a koncový stav $f \in F$ je definováno pravidlo typu *pops*. V případě dosažení koncového stavu se symbol ze vstupní pásky porovnává se symbolem prvního neprázdného zásobníku a v případě shody jsou přijaty. Celý proces se opakuje, dokud není načten řetězec ze vstupní pásky. Zatímco u modelů DPDAs bylo možné pravidla aplikovat kdykoli, pokud se na vrcholu zásobníku nacházel vstupní symbol, v tomto mechanismu je jejich vykonávání omezeno aktuálním vnitřním stavem.

4.2 Konfigurace a typy přechodů

Konfigurace čistých hlubokých seřazených n-zásobníkových automatů je zapisována podobně jako u hlubokých zásobníkových automatů ve formě $\chi = (Q \times \Sigma^* \times (\Sigma^*\{\xi\})^n)$, kde Q představuje vnitřní stav automatu, Σ^* reprezentuje posloupnost znaků vstupního řetězce, které ještě nebyly přijaty. Dále výraz $(\Sigma^*\xi)^n$ vyjadřuje aktuální stavy seřazených n zásobníků, kde ξ označuje dno zásobníku a $\Sigma^*\xi$ představuje posloupnost vstupních symbolů zakončenou dnem, některého z řady n zásobníků. V případě, že mluvíme o prázdném zásobníku, rozumíme tím zásobník obsahující pouze speciální symbol ξ označující jeho dno.

Nechť X_M označuje množinu všech možných konfigurací, kterých může dosáhnout automat M typu ${}_n$OPDPDA. Mějme dvě konfigurace $x, y \in X_M$. Pokud automat provede krok, při kterém porovná vstupní symbol na vrcholu prvního neprázdného zásobníku s aktuálně čteným symbolem na vstupní pásce a tyto symboly se shodují, dojde k vyjmutí těchto symbolů a čtecí hlava na vstupní pásce se posune na následující symbol. Tento krok, spočívající v porovnání a vyjmutí symbolů, je označován jako *pops* a je symbolicky zapisován jako,

$$x \vdash_p y$$

a příkladem takové operace může být $(q, au, saz) \vdash_p (q, u, sz)$, kde $q \in Q$, $a \in \Sigma$, $u \in \Sigma^*$, $s \in \xi^j$, $z \in (\Sigma^*\xi)^k$, přičemž platí $n = j + k$.

Dalším typem kroku, který automat může provést, jsou přechody typu *expands*. Vykoná-li automat přechod z konfigurace x do y, během kterého dochází k přepisu vstupních symbolů uvnitř některého z dostupných zásobníků způsobem, kdy je daný symbol přepsán na jeden či více znaků, nebo naopak odstraněn, pak je tato operace symbolicky zapisována jako,

$$x \vdash_e y$$

Příkladem takové operace může být $(q, u, saz) \vdash_e (p, u, svz)[q(ka)j \to p(v)]$, kde $q, p \in Q$, $u \in \Sigma^*$, $s \in (\Sigma^*\xi)^{j-1}\Sigma^{k-1}$, $k, j > 0$, $a \in \Sigma$, $v \in \Sigma^*$, $z \in (\Sigma^*\xi)^{n-j}$. Při použití pravidla je

vhodné tento zápis doplnit zvoleným pravidlem definujícím daný přechod mezi konfiguracemi. Dále je vhodné podotknout, že index označující přepisovaný symbol v posloupnosti symbolů by neměl převyšovat délku celé posloupnosti. To rovněž platí pro index označující číslo zásobníku v pořadí zásobníků.

Příklad 4.2.1 *Mějme jazyk* $L = \{a^j b^j c^j d^j | j \geq 0\}$ *pro který definujeme čistý hluboký seřazený n-zásobníkový automat, který jej bude přijímat.*
Nechť $M_{4OPDPDA} = (\{s, q_*, p_*, p, g_*, g, v_s, v_*, u_*, u, f_*, f\}, \{a, b, c, d\}, R, s, \#, a, \{f\})$ *je čistý hluboký seřazený 4-zásobníkový automat, kde konečné množině* R *náleží pravidla,*

1	:	$s(1\tau)1$	$\rightarrow$	$q_*(\tau a)$
2	:	$q_*(1\tau)2$	$\rightarrow$	$p_*(\tau b)$
3	:	$p_*(1\tau)3$	$\rightarrow$	$p(\tau c)$
4	:	$p(1\tau)4$	$\rightarrow$	$s(\tau d)$
5	:	$s(1\tau)1$	$\rightarrow$	$g_*(\varepsilon)$
6	:	$g_*(1a)1$	$\rightarrow$	$g(\epsilon)$
7	:	$g(1\tau)2$	$\rightarrow$	$g_*(\tau a)$
8	:	$g_*(1\#)1$	$\rightarrow$	$v_s(\#)$
9	:	$v_s(1\tau)2$	$\rightarrow$	$v_*(\varepsilon)$
10	:	$v_*(1\tau)3$	$\rightarrow$	$u_*(\varepsilon)$
11	:	$u_*(1c)3$	$\rightarrow$	$u(\varepsilon)$
12	:	$u(1\tau)4$	$\rightarrow$	$u_*(\tau c)$
13	:	$u_*(1\#)3$	$\rightarrow$	$f_*(\#)$
14	:	$f_*(1\tau)4$	$\rightarrow$	$f(\varepsilon)$

Pro řetězec aabbccdd přijímaný automatem $M_{4OPDPDA}$ *by sekvence přechodů vypadala následovně,*

$(s, aabbccdd, a\#a\#a\#a\#,)$	$\vdash_e$	$(q_*, aabbccdd, aa\#a\#a\#a\#)$	[1]
	$\vdash_e$	$(p_*, aabbccdd, aa\#ab\#a\#a\#)$	[2]
	$\vdash_e$	$(p, aabbccdd, aa\#ab\#ac\#a\#)$	[3]
	$\vdash_e$	$(s, aabbccdd, aa\#ab\#ac\#ad\#)$	[4]
	$\vdash_e$	$(q_*, aabbccdd, aaa\#ab\#ac\#ad\#)$	[1]
	$\vdash_e$	$(p_*, aabbccdd, aaa\#abb\#ac\#ad\#)$	[2]
	$\vdash_e$	$(p, aabbccdd, aaa\#abb\#acc\#ad\#)$	[3]
	$\vdash_e$	$(s, aabbccdd, aaa\#abb\#acc\#add\#)$	[4]
	$\vdash_e$	$(g_*, aabbccdd, aa\#abb\#acc\#add\#)$	[5]
	$\vdash_e$	$(g, aabbccdd, a\#abb\#acc\#add\#)$	[6]
	$\vdash_e$	$(g_*, aabbccdd, a\#aabb\#acc\#add\#)$	[7]
	$\vdash_e$	$(g, aabbccdd, \#aabb\#acc\#add\#)$	[6]
	$\vdash_e$	$(g_*, aabbccdd, \#aaabb\#acc\#add\#)$	[7]
	$\vdash_e$	$(v_s, aabbccdd, \#aaabb\#acc\#add\#)$	[8]
	$\vdash_e$	$(v_*, aabbccdd, \#aabb\#acc\#add\#)$	[9]
	$\vdash_e$	$(u_*, aabbccdd, \#aabb\#cc\#add\#)$	[10]
	$\vdash_e$	$(u, aabbccdd, \#aabb\#c\#add\#)$	[11]
	$\vdash_e$	$(u_*, aabbccdd, \#aabb\#c\#acdd\#)$	[12]
	$\vdash_e$	$(u, aabbccdd, \#aabb\#\#acdd\#)$	[11]
	$\vdash_e$	$(u_*, aabbccdd, \#aabb\#\#accdd\#)$	[12]
	$\vdash_e$	$(f_*, aabbccdd, \#aabb\#\#accdd\#)$	[13]
	$\vdash_e$	$(f, aabbccdd, \#aabb\#\#ccdd\#)$	[14]
	$\vdash_p^*$	$(f, , \#\#\#\#)$	

4.3 Přijímané jazyky

Mějme čistý hluboký seřazený n-zásobníkový automat M. Jazyk L je přijímán automatem M pouze tehdy, pokud M přijme všechny řetězce, které do něj patří. Model ${}_n$OPDPDA přijímá vstupní řetězec za předpokladu, že byl načten celý vstupní řetězec, aktuální vnitřní stav patří do množiny koncových stavů a všechny zásobníky automatu byly vyprázdněny, tj. v každém zásobníku zůstává pouze jeden speciální symbol ξ, označující dno zásobníku. Přijímaný jazyk je formálně definován jakožto,

$$L(M) = \{w \in \Sigma^* | (s, w, (\tau\xi)^n) \vdash_e^* (f, w, (v\xi)^n) \vdash_p^* (f, \varepsilon, \xi^n), kde\ f \in F, v \in \Sigma^*\}$$

V definici přijímaného jazyka automat až do dosažení vnitřního stavu $f \in F$ opakovaně aplikuje pravidla typu *expands* a až následně potom zahájí fázi porovnávání vstupního řetězce s obsahem zásobníků:

- načte další symbol ze vstupní pásky,
- porovná jej se symbolem na vrcholu prvního neprázdného zásobníku,
- pokud se symboly shodují, odstraní symbol ze zásobníku a posune čtecí hlavu o krok vpřed.

4.4 Simulace DPDA pomocí OPDPDA

Nyní následuje představení simulace modelů DPDAs pomocí modelů OPDPDAs. Předpokládá se, že existuje simulovaný automat DPDA stupně $k \geq 1$, který bude označován jako ${}_kM$. Nechť

$${}_kM = (Q, \Sigma, \Gamma, R, s, S, F),$$

přičemž se bez ztráty obecnosti předpokládá, že množina pravidel R obsahuje pravidla ve tvaru

$$q(mA) \to p(X),$$

kde

- $q, p \in Q$;
- $m \in \mathbb{N}$;
- $A \in \Gamma \setminus (\Sigma \cup \{\#\})$;
- $X \in ((\Gamma \setminus (\Sigma \cup \{\#\}))^* \cup \Sigma^* \cup \left(\Sigma^*(\Gamma \setminus (\Sigma \cup \{\#\}))\Sigma^*\right))$.

Dále se bez ztráty obecnosti předpokládá, že automat neobsahuje pravidla, u nichž by byla volba dalšího kroku určována pouze na základě nevstupních symbolů. To znamená, že je preferován model automatu, u kterého při přechodech mezi konfiguracemi hraje roli stav z množiny Q společně s aktuálním stavem zásobníku. Například namísto množiny pravidel

$$R = \{q(1S) \to q(A),\ q(1S) \to q(B)\},$$

je vhodné použít upravenou verzi automatu s explicitně uvedenými stavy

$$R = \{q(1S) \to q_A(A),\ q(1S) \to q_B(B)\}.$$

Pro simulaci DPDA pomocí simulujícího OPDPDA, je nutné si nejprve uvědomit, jak OPDPDAs fungují. Tyto modely během zpracování vstupního řetězce pracují s předem stanoveným maximálním počtem n zásobníků, které nahrazují nevstupní symboly u simulovaných DPDAs. Jelikož původní model přepisuje tyto nevstupní symboly pomocí indexování do hloubky, je třeba simulující model navrhnout tak, aby zachovával informaci o tom, v jakém zásobníku a na které pozici v něm bude proveden přepis podle použitého pravidla. Tento úkol může být náročnější, než se na první pohled zdá, neboť s narůstající komplexitou simulovaného modelu, tedy s rostoucím počtem nevstupních symbolů v zásobníku během zpracování řetězce, se zvyšuje i počet potřebných zásobníků simulujícího OPDPDA a možností, jak s nimi pracovat.

Tato práce zavádí jeden ze způsobů, jak vytvořit simulující model OPDPDA, a to nalezením pouze takových posloupností expanzivních pravidel, jejichž aplikací přejde simulovaný model DPDA z počátečního stavu s do koncového $f \in F$, přičemž na zásobníku zůstane vygenerovaný vstupní řetězec ω zakončený symbolem # představující dno zásobníku. Podmínkou simulovaného DPDA je, že nesmí obsahovat pravidla $r \in R$, která mohou neomezeně v cyklech navyšovat či měnit zastoupení nevstupních symbolů v zásobníku. Takové modely OPDPDA neumí simulovat a v následujícím textu je vysvětlen důvod tohoto omezení.

Algorithm 1 Simulace $DPDA_n$ pomocí ${}_mOPDPDA_n$

Input: $M_{DPDA_n} = (Q, \Sigma, \Gamma, R, s, S, F)$
Output: $M_{OPDPDA_n} = (Q', \Sigma, R', s, \tau, \xi, F')$
foreach *path* $\in$ *paths* **do**
  (*pd_state*, *pd_init*) $\leftarrow (S, \xi)$ **foreach** *edge* $q(jC) \to p(\nu Y \mu)$ *in path* **do**
    foreach *cycle_path* $\in$ *cycle_paths* **do**
      foreach *cycle_edge* $v(lA) \to u(\nu B \mu)$ *in cycle_path where* $\nu, \mu \in \Sigma^*$, $B \in \Delta$ **do**
        pd_state $\leftarrow$ create_transition(*pd_state*, *cycle_edge*)
      end
    end
    (*pd_state*, *pd_init*) $\leftarrow$ create_transition(*pd_state*, *edge*, *pd_init*)
  end
  $R' \leftarrow R' \cup \{\, s(\texttt{SET}, pd\_init) \to s^S_{<\texttt{path}>}\}$
end

Algoritmus výše představuje postup konstrukce simulovaného DPDA pomocí OPDPDA. Procházena je zde množina posloupností pravidel (cest)

$$paths = \big\{(r_0, \ldots, r_k) \mid r_i \in R,\ (s, w, S\#) \vdash_e^{r_0} \cdots \vdash_e^{r_k} (f, w, w\#),\\ f \in F,\ r_j \neq r_l\ \forall\, 0 \leq j < l \leq k\big\}$$

přičemž každá taková posloupnost reprezentuje ve výsledném simulujícím modelu OPDPDA jeden strom začínající v počátečním stavu s a celá množina *paths* tak tvoří les.

Algoritmus následně prochází pravidla aktuálně zpracovávané cesty $path \in paths$ modelu DPDA. Při přejití do následujícího stavu $p \in Q$ dle aktuálního pravidla v sekvenci pravidel *path* provádí průchod všemi dostupnými cyklickými posloupnostmi pravidel *cycle_paths*, dostupných ze stavu p simulovaného modelu, pro něž platí, že automat po

jejich vykonání vždy skončí v původním stavu p a zároveň se nezmění počet ani pořadí nevstupních symbolů v zásobníku.

$$\begin{aligned} \mathit{cycle\_paths} = \{(r_0, r_1, \ldots, r_{k-1}, r_k) \mid r_i \in R,\ r_i = v(lA) \to u(X),\ l \in \mathbb{N},\ v, u \in Q, \\ A \in \Delta,\ X \in \Sigma^* \Delta \Sigma^*,\ i \leq k,\ (q, w, \theta\#) \vdash_e^{r_0} \cdots \vdash_e^{r_k} (q, w, \Theta\#), \\ q \in Q,\ \#_\Delta(\theta) = \#_\Delta(\Theta),\ \Delta = \Gamma - \Sigma - \{\#\}\} \end{aligned}$$

Jinými slovy, jedná se o takové cyklické posloupnosti pravidel, které pouze cyklicky navyšují počet vstupních symbolů v zásobníku, bez modifikování zastoupení nevstupních symbolů, například $q(2A) \to q(aAb)$, nikoliv $q(2A) \to q(AA)$. Během tohoto procesu jsou generována pravidla pro množinu R' a zároveň je aktualizován stav simulovaných zásobníků v rámci nově vytvářeného OPDPDA.

Během zpracování $path \in paths$ si konstruktor nad danou cestou udržuje informaci o aktuálních stavech zásobníků. Tyto informace jsou udržovány v proměnných `pd_state` a `pd_init`. Zatímco `pd_state` slouží pro udržení a aktualizování informací o stavech a počtu zásobníků OPDPDA, tak `pd_init` slouží pro udržení informací pro následné vytvoření pravidla `SET`, na konci zpracování stromu. Tyto informace v obou proměnných se v průběhu zpracování dané cesty mění a jejich existence je nezbytná pro dosažení korektního indexování symbolů a zásobníků v rámci pravidel simulujícího OPDPDA.

Algorithm 2 Funkce create_transition

Function `Create_transition`($pd\_state, q(jA) \to p(X), pd\_init \leftarrow False$)**:**

$\gamma = (\Sigma \cup \{\xi\})^*((\Sigma \cup \{\xi\})^*\Delta(\Sigma \cup \{\xi\})^*)^j$, $\eta = (\Sigma \cup \Delta \cup \{\xi\})^*$, pd_state $= \gamma A \eta$,
$\varrho = (\Sigma \cup \{\xi\})^*((\Sigma \cup \{\xi\})^*\Delta(\Sigma \cup \{\xi\})^*)^j$, $\vartheta = (\Sigma \cup \Delta \cup \{\xi\})^*$, pd_init $= \varrho A \vartheta$
if $X \in \Sigma^*\Delta\Sigma^*$ **then**
  $X = \nu B \mu, B \in \Delta, \nu, \mu \in \Gamma$
  $X \leftarrow \{Y \in \Delta^x | x = \#_\Delta(X), l = \#_\Gamma(X), X = X_1X_2 \dots X_l, Y_k = X_i, \#_\Delta(X_1 \dots X_i) = k\}$
end
if $X \in \Sigma^*, \Delta^*$ **then**
  $\nu, \mu \in \varepsilon$
end
if $\#(\nu) > 0, \gamma.last\_two() \neq \xi^2, \gamma.last\_two() \neq \xi$ **then**
  R'.increment_pd($\#_\xi(\gamma) + 1$*, path),*
  $\gamma \leftarrow \gamma\xi, \varrho \leftarrow \varrho\xi$
  if $\gamma.last() \neq \varepsilon, \gamma.last() \neq \xi$ **then**
    R'.increment_pd($\#_\xi(\gamma) + 1$*, path),*
    $\gamma \leftarrow \gamma\xi, \varrho \leftarrow \varrho\xi$
  end
end
if $\#(\mu) > 0, \eta.first() \neq \xi$ **then**
  $\eta \leftarrow \xi\eta, \vartheta \leftarrow \xi\vartheta$
  R'.increment_pd($\#_\xi(\gamma) + 2$*, path, True)*
end
pd_state $\leftarrow \gamma X \eta$
if $\#_\Sigma(X) = \#(X)$ **then**
  $X \leftarrow \tau$
end
if $X \in \Delta^*$ & $\#(X) > 0$ **then**
  R'.append($q() \to p()$*)*
end
else if $\#(\nu) > 0$ **then**
  R'.append($q(COPY)\#_\Delta(\gamma) \to p_{uid}(\nu)$*),*
  R'.append($p_{uid}(1\tau)\#_\Delta(\gamma) + 1 \to p(\tau\mu)$*),* $Q'.append(p_{uid})$
end
else if $\#(\nu) = 0$ **then**
  $\theta \leftarrow \#(\gamma) - \max\{\, i \mid \gamma_i = \xi\} + 1$,
  change $\leftarrow$ *if B then* $\tau\mu$ *else* $X, R'.append(q(\theta\tau)\#_\Delta(\gamma) + 1 \to p(change))$
end
if *pd_init* **then**
  pd_init $\leftarrow \varrho X \vartheta$, **return***pd_state, pd_init*
end
return *pd_state*

Algoritmus uvedený výše popisuje postup, jak za běhu programu vytvářet nová pravidla $r' \in R'$ pro simulující model OPDPDA. Na začátku funkce se nejprve analyzuje samotný typ pravidla, dle jeho pravé strany

$$X \in (\Gamma \setminus (\Sigma \cup \{\#\}))^+ \cup \Sigma^* \cup (\Sigma^*(\Gamma \setminus (\Sigma \cup \{\#\}))\Sigma^*).$$

V případě, že se jedná o pravidlo $q(nA) \to p(X)$ s pravou stranou,

$$X \in (\Gamma \setminus (\Sigma \cup \{\#\}))^+$$

bude pro OPDPDA konstruováno pravidlo $R' = R' \cup \{q_{path}() \to p_{path}()\}$, které bez jakékoli změny nad zásobníky provede přechod mezi stavy. Dále v rámci konstruktoru bude upraven aktuální stav simulovaných zásobníků OPDPDA v proměnných `pd_state` a `pd_init` a to přepsáním n-tých nevstupních symbolů A na cílovou posloupnost X.

Jestliže se jedná o pravidlo $q(nA) \to p(X)$ s pravou stranou

$$X \in \Sigma^*$$

bude pro OPDPDA konstruováno pravidlo $R' = R' \cup \{q_{path}(j\tau)m \to p_{path}(X)\}$, které kromě přejití do následujícího stavu přepíše daný symbol τ na posloupnost X. Index j odpovídá počtu symbolů + 1, mezi n-tým nevstupním symbolem A v `pd_state` a m odpovídá počtu symbolů ξ + 1 před daným znakem A. Tato změna bude rovněž vepsána do aktuálního simulovaného stavu zásobníků OPDPDA způsobem, kdy v `pd_state` bude přepisovaný n-tý nevstupní symbol rozgenerován na posloupnost X a v `pd_init` bude přepsán na symbol τ.

V případě, že právě zpracovávané pravidlo $q(nA) \to p(X)$ přepisuje nevstupní A symbol na posloupnost

$$X \ \to \ \nu\,\Delta\,\mu, \qquad \Delta \in \Gamma \setminus (\Sigma \cup \{\#\}),\ \nu, \mu \in \Sigma^*,$$

provádí se následující úpravy zásobníků:

- *Prefixová podmínka:* Pokud $\#(\nu) > 0$, pak se v `pd_state` před přepisovaným n-tým nevstupním symbolem musí nacházet dva symboly ξ (tj. alokuje se nový zásobník vyhrazený pro operaci `COPY`).
- *Postfixová podmínka:* Pokud $\#(\mu) > 0$, pak se za přepisovaným n-tým nevstupním symbolem v `pd_state` musí nacházet jeden symbol dna ξ (tj. alokuje se další zásobník, do kterého se budou zapisovat zbylé symboly).

V případě změny pořadí a počtu zásobníků je nutné aktualizovat již existující množinu pravidel R' tak, že se upraví indexy pozic přepisovaných symbolů a odpovídajících zásobníků. Tato úprava se týká zejména pravidel, která mají indexy pozice přepisovaných symbolů a zásobníků větší nebo rovné indexům nově vložených pravidel.

Příklad 4.4.1 *Nechť* `pd_state` $\leftarrow ABb\xi Cc\xi$ *představuje aktuální simulovaný stav zásobníků OPDPDA, kde* $A, B, C \in \Delta$, $a, b, c \in \Sigma$ *a množině pravidel* R' *náleží pravidla:*

1. $q(2\tau)1 \to q(\tau b)$
2. $q(1\tau)2 \to q(\tau c)$

Potom s příchozím pravidlem $q(1A) \to q(Aa)$ *bude v* `pd_state` *vytvořen nový zásobník způsobem* `pd_state` $\leftarrow Aa\xi Bb\xi Cc\xi$. *Dále se upraví množina pravidel do podoby:*

1. $q(1\tau)2 \to q(\tau b)$
2. $q(1\tau)3 \to q(\tau c)$
3. $q(1\tau)1 \to q(\tau a)$

Po úpravě množiny pravidel R' je ještě potřebné zkonstruovat nová pravidla, reprezentující zpracovávané pravidlo $q(nA) \to p(X)$. V rámci modelů OPDPDAs může nastat situace, kdy tento typ pravidla modelu DPDA nepůjde realizovat jediným přechodem v novém simulujícím modelu. Tento model totiž funguje tak, že

- pro vygenerování prefixu ν použije pravidlo typu `COPY` na zásobník s indexem m-1, je-li $\#(\nu) > 0$,
- a zbývající část řetězce, včetně symbolu τ, vloží na aktuální zásobník s indexem m.

Z toho důvodu se každé původní přepisové pravidlo modelu DPDA převede na dva na sebe navazující samostatné přechody, platí-li $\#(\nu) > 0$. Nová pravidla jsou tedy zkonstruována následovně:

- Platí-li $\#(\nu) = 0$, potom $R' \leftarrow R' \cup \{q_{path}(j\tau)m \to p_{path}(\tau\mu)\}$, kde j odpovídá počtu symbolů + 1, mezi n-tým nevstupním symbolem A v `pd_state` a m odpovídá počtu symbolů $\xi + 1$ před daným znakem A.
- Platí-li $\#(\nu) > 0$, potom $R' \leftarrow R' \cup \{q_{path}(COPY)m\text{-}1 \to p^{uid}_{path}(\nu), p^{uid}_{path}(j\tau)m \to p_{path}(\tau\mu)\}$, kde j odpovídá počtu symbolů + 1, mezi n-tým nevstupním symbolem A v `pd_state`, m odpovídá počtu symbolů $\xi + 1$ před daným znakem A a uid je unikátní id pro danou dvojici pravidel.

Definice 4.4.1 *Pravidlo `COPY` abstraktně reprezentuje přerušovanou posloupnost přechodů, při níž dochází k vložení invertovaného řetězce na zásobník vyhrazený pro pravidla typu `COPY`. Během chodu automatu může být pravidlo `COPY` využito libovolně krát. Ve chvíli, kdy automat dosáhne stavu $f \in F'$, dojde, ještě před přechody pops, k přesunu symbolů ze zásobníků `COPY` do zásobníků s o jeden vyšším indexem. Pravidlo `COPY` definujeme jakožto,*

$$r_{copy} = (Q \times \Sigma^* \times \mathbb{N} \times Q),$$

Pro každé pravidlo $q(\texttt{COPY})m \to p(X) \in R'$ platí:

- $q(1\tau)m \to p(\tau X^R)$*, kde X^R je obrácený řetězec X*
- *Jakmile automat dosáhne koncového stavu $f \in F'$, je započata fáze kopírování:*
  - $f^n_{COPY}(2v)m \to f^n_{COPY_v}(\varepsilon)$*, kde $v \in \Sigma$*
  - $f^n_{COPY_v}(1\tau)m{+}1 \to f^n_{COPY}(\tau v)$*, kde τ je dočasný pomocý symbol*

První řádek říká, že každé pravidlo typu $q(\texttt{COPY})m \to p(X) \in R'$ vkládá na zásobník m za symbol τ na indexu 1, řetězec X v obráceném pořadí.

Po dosažení stavu $f \in F'$ automatem, se spustí proces kopírování, kdy,

$$f^n_{\mathrm{COPY}}(2v)m \to f^n_{\mathrm{COPY}_v}(\varepsilon),$$

odstraní druhý symbol ze zásobníku a přejde do mezistavu, pomocí kterého je uchována informace o odstraněném symbolu. Následně pomocí pravidla

$$f^n_{\mathrm{COPY}_v}(1\tau)m{+}1 \to f^n_{\mathrm{COPY}}(\tau v)$$

je tento symbol uložen do zásobníku od index výše, za pomocný dočasný symbol τ. Model se následně vrátí do původního stavu a pokračuje v kopírování, dokud nenarazí na dno zásobníku.

Před zahájením kopírování zásobníku m se na začátek zásobníku $m+1$ vloží pomocný dočasný symbol τ. Po dokončení celého kopírování je tento symbol τ odstraněn, a zároveň se odstraní také inicializační symbol τ ze zásobníku m. Celý tento proces se lineárně provede nad všemi zásobníky, dedikovanými pro pravidla typu `COPY`.

Je vhodné podotknout, že symboly τ, sloužící ke vkládání obrácených řetězců X^R na zásobníky vyhrazené pro pravidla typu `COPY`, zůstávají v těchto zásobnících po celou dobu generování řetězců a zpracování vstupního řetězce. Teprve po dosažení koncového stavu, kdy se spustí proces kopírování nad daným zásobníkem, je symbol τ ze zásobníku odstraněn po dokončení procesu kopírování nad daným zásobníkem.

Při tvorbě stromů simulujícího modelu OPDPDA se vždy na konci vytváří s pomocí proměnné `pd_init` pravidlo `SET`, které abstraktně reprezentuje lineární posloupnost přechodů, které buďto navyšují počet symbolů τ v zásobnících, nebo naopak tyto symboly odstraňují, jedná-li se o nadbytečné zásobníky pro daný strom.[1]

Definice 4.4.2 *Pravidlo `SET` abstraktně reprezentuje lineární posloupnost přechodů, ve kterých dochází k navýšení počtu τ symbolů v zásobnících či k odstranění symbolů v nadbytečných zásobnících. Pravidlo `SET` definujeme jakožto,*

$$r_{set} = (Q \times (x\xi)^k \times Q),$$

kde

- $k \leq m$, *přičemž* m *reprezentuje množství zásobníků modelu* ${}_mOPDPDA$;
- k *představuje počet zásobníků, použitých v daném stromě;*
- $x \in \{\tau\}^*$ *je posloupnost počátečních vstupních symbolů;*

Každý podřetězec $(x\xi)^j$, kde $1 \leq j \leq k$ představuje nově inicializovaný stav j-tého zásobníku, přičemž x označuje posloupnost τ symbolů, které se v daném zásobníku nacházejí po aplikaci pravidla r_{set}. Platí-li $\#(x) = 0$ v $(x\xi)^j$, považujeme daný j-tý zásobník určený pro operace `COPY` a zůstane v něm obsažen jeden symbol τ. Situace, ve které by byly vedle sebe dva a více takových zásobníků, není povolena. Zásobníky s indexy $[k+1 \ldots m]$, které v rámci operace `SET` nejsou specifikovány, označujeme jako nadbytečné a počáteční symboly τ jsou z nich odstraněny[2].

V této práci budeme pro lepší čitelnost pravidla typu `SET` zapisovat ve tvaru

$$q(\texttt{SET})m \;\rightarrow\; p(X),$$

kde

- $q, p \in Q'$;
- $m \in \mathbb{N}$ označuje index zásobníku;
- $X \in (x\xi)^k$;

[1] Poznámka: Na začátku chodu automatu začíná každý zásobník s hodnotou τ následovanou symbolem ξ

[2] Poznámka: Zásobníky budou vyprázdněny a budou obsahovat pouze speciální symbol dna ξ.

- `SET` vyjadřuje typ pravidla;

Nyní následuje příklad, ve kterém lze poukázat, jak by vypadaly dva ekvivalentní modely OPDPDA definující stejný jazyk, přičemž první z nich využívá abstraktní pravidla `COPY` a `SET`, zatímco druhý používá pouze běžná atomická pravidla ve tvaru $(Q \times (\mathbb{N} \times \Sigma) \times \mathbb{N} \times Q \times \Sigma^*)$.

Příklad 4.4.2 *Mějme model* ${}_3M_{OPDPDA_1} = (Q, \Sigma, R, s, a, \#, f)$*, který má dle zadání 3 zásobníky* [3] *s následně definovanými parametry,*

$$
\begin{array}{lcll}
1 & : & s(SET) & \rightarrow & q(\#aa\#) \\
2 & : & q(COPY)1 & \rightarrow & q_0(ab) \\
3 & : & q_0(2a)2 & \rightarrow & q(ac) \\
4 & : & q(1a)2 & \rightarrow & p(cc) \\
5 & : & p(3a)2 & \rightarrow & f(aa)
\end{array}
$$

Potom by ekvivalentní automat, s počátečním stavem s_{set2}, v němž jsou abstraktní pravidla `COPY` a `SET` rozložena na posloupnosti základních přechodů (tj. atomických pravidel), vypadal následovně,

$$
\begin{array}{lcll}
1 & : & s_{set2}(1a)2 & \rightarrow & s_{set3}(\#aa\#) \\
2 & : & s_{set3}(1a)3 & \rightarrow & q_{copy1}(\varepsilon) \\
3 & : & q_{copy1}(1a)1 & \rightarrow & q^0_{copy1}(aba) \\
4 & : & q^0_{copy1}(2a)2 & \rightarrow & q_{copy1}(ac) \\
5 & : & q_{copy1}(1a)2 & \rightarrow & p(cc) \\
6 & : & p(3a)2 & \rightarrow & f_{Init1}(aa) \\
7 & : & f_{Init1}(1c)2 & \rightarrow & f_{mc1}(ac) \\
8 & : & f_{mc1}(2a)1 & \rightarrow & f_{mc1a}(\varepsilon) \\
9 & : & f_{mc1a}(1a)2 & \rightarrow & f_{mc1}(aa) \\
10 & : & f_{mc1}(2b)1 & \rightarrow & f_{mc1b}(\varepsilon) \\
11 & : & f_{mc1b}(1a)2 & \rightarrow & f_{mc1}(ab) \\
12 & : & f_{mc1}(2c)1 & \rightarrow & f_{mc1c}(\varepsilon) \\
13 & : & f_{mc1c}(1a)2 & \rightarrow & f_{mc1}(ac) \\
14 & : & f_{mc1}(2\#)1 & \rightarrow & f_{mc1e1}(\#) \\
15 & : & f_{mc1e1}(1a)1 & \rightarrow & f_{mc1e2}(\varepsilon) \\
16 & : & f_{MC1e2}(1a)2 & \rightarrow & f(\varepsilon)
\end{array}
$$

Ve chvíli, kdy automat přejde do stavu f, zahájí se operace `pops` a následné porovnávání symbolů v zásobnících se symboly na vstupní pásce. Pokud přitom dojde k načtení celého vstupního řetězce a současnému vyprázdnění všech zásobníků, automat úspěšně skončí a řetězec je přijat.

Příklad 4.4.3 *Nechť $M_{DPDA} = (Q, \{a, b, c\}, \{a, b, c, A, C, \#\}, R, s, S, \{f\})$ je hluboký zásobníkový automat, kde $Q = \{s, p, q, f, g, h, k, j, v, n\}$ a konečné množině pravidel R náleží,*

[3] Třetí zásobník je zde cíleně navíc, a bude sloužit pro lepší představení rozepsaného pravidla `COPY`

$$
\begin{array}{llcl}
1 & : & s(1S) & \rightarrow & q(AC) \\
2 & : & q(1A) & \rightarrow & p(aA) \\
3 & : & p(2C) & \rightarrow & q(Cc) \\
4 & : & q(1A) & \rightarrow & g(a) \\
5 & : & g(1C) & \rightarrow & f(c) \\
6 & : & s(1S) & \rightarrow & h(AC) \\
7 & : & h(1A) & \rightarrow & k(aaA) \\
8 & : & k(1A) & \rightarrow & j(Ab) \\
9 & : & j(2C) & \rightarrow & h(Cc) \\
10 & : & k(1A) & \rightarrow & v(Ab) \\
11 & : & v(2C) & \rightarrow & q(Cc) \\
12 & : & h(1A) & \rightarrow & n(aab) \\
13 & : & n(1C) & \rightarrow & f(c)
\end{array}
$$

Konstrukce OPDPDA:

$$paths = \{(1,4,5),(6,7,10,11,4,5),(6,12,13)\}$$

1. Path ← (1,4,5)

(a) $s(1S) \rightarrow q(AC), pd\_state = S\#, pd\_init = S\#,\ cycle\_paths = \emptyset$

$$R'_{ac} = \Big\{ \qquad\qquad s_{ac}() \rightarrow q_{ac}() \ \Big\}.$$

(b) $q(1A) \rightarrow g(a), pd\_state = AC\#, pd\_init = AC\#,\ cycle\_paths = \{(2,3)\}$

$$R'_{ac} = \Big\{ \begin{array}{l} s()_{ac} \rightarrow q_{ac}(), \quad q_{ac}(\mathrm{COPY})1 \rightarrow p^0_{ac}(a), \\ p^0_{ac}(1b)2 \rightarrow p_{ac}(b), \quad p_{ac}(2b)2 \rightarrow q_{ac}(bc), \\ q_{ac}(1b)2 \rightarrow g_{ac}(a) \end{array} \Big\}.$$

(c) $g(1C) \rightarrow f(c), pd\_state = \#aC\#, pd\_init = \#bC\#,\ cycle\_paths = \emptyset$

$$R'_{ac} = \Big\{ \begin{array}{l} s()_{ac} \rightarrow q_{ac}(), \quad q_{ac}(\mathrm{COPY})1 \rightarrow p^0_{ac}(a), \\ p^0_{ac}(1b)2 \rightarrow p_{ac}(b), \quad p_{ac}(2b)2 \rightarrow q_{ac}(bc), \\ q_{ac}(1b)2 \rightarrow g_{ac}(a), \quad g_{ac}(2b)2 \rightarrow f_{ac}(c) \end{array} \Big\}.$$

(d) $SET, pd\_state = \#ac\#, pd\_init = \#bb\#$

$$R'_{ac} + \{s(SET) \rightarrow s_{ac}(\#bb\#)\}$$

2. Path ← (6,7,10,11,4,5)

(a) $s(1S) \rightarrow h(AC), pd\_state = S\#, pd\_init = S\#,\ cycle\_paths = \emptyset$

$$R'_{aaabcc} = \Big\{ \qquad\qquad s_{aaabcc}() \rightarrow h_{aaabcc}() \ \Big\}.$$

(b) $h(1A) \rightarrow k(aaA), pd\_state = AC\#, pd\_init = AC\#,\ cycle\_paths = \{7,8,9\}$

$$R'_{aaabcc} = \Big\{ \begin{array}{l} s_{aaabcc}() \rightarrow h_{aaabcc}(), \quad h_{aaabcc}(\mathrm{COPY})1 \rightarrow k^0_{aaabcc}(aa), \\ k^0_{aaabcc}(1b)2 \rightarrow k_{aaabcc}(b), \quad k_{aaabcc}(1b)2 \rightarrow j_{aaabcc}(bb), \\ j_{aaabcc}(1b)3 \rightarrow h_{aaabcc}(bc) \end{array} \Big\}.$$

(c) $k(1A) \rightarrow v(Ab), pd\_state = \#A\#C\#, pd\_init = \#A\#C\#, cycle\_paths = \{7, 8, 9\}$

$$R'_{aaabcc} = \Big\{ \begin{array}{ll} s_{aaabcc}() \rightarrow h_{aaabcc}(), & h_{aaabcc}(\text{COPY})1 \rightarrow k^0_{aaabcc}(aa), \\ k^0_{aaabcc}(1b)2 \rightarrow k_{aaabcc}(b), & k_{aaabcc}(1b)2 \rightarrow j_{aaabcc}(bb), \\ j_{aaabcc}(1b)3 \rightarrow h_{aaabcc}(bc), & k_{aaabcc}(1b)2 \rightarrow v_{aaabcc}(bb) \end{array} \Big\}.$$

(d) $v(2C) \rightarrow q(Cc), pd\_state = \#A\#C\#, pd\_init = \#A\#C\#, cycle\_paths = \emptyset$

$$R'_{aaabcc} = \Big\{ \begin{array}{ll} s_{aaabcc}() \rightarrow h_{aaabcc}(), & h_{aaabcc}(\text{COPY})1 \rightarrow k^0_{aaabcc}(aa), \\ k^0_{aaabcc}(1b)2 \rightarrow k_{aaabcc}(b), & k_{aaabcc}(1b)2 \rightarrow j_{aaabcc}(bb), \\ j_{aaabcc}(1b)3 \rightarrow h_{aaabcc}(bc), & k_{aaabcc}(1b)2 \rightarrow v_{aaabcc}(bb), \\ v_{aaabcc}(1b)3 \rightarrow q_{aaabcc}(bc) & \end{array} \Big\}.$$

(e) $q(1A) \rightarrow g(a), pd\_state = \#A\#C\#, pd\_init = \#A\#C\#, cycle\_paths = \{2, 3\}$

$$R'_{aaabcc} = \Big\{ \begin{array}{ll} s_{aaabcc}() \rightarrow h_{aaabcc}(), & h_{aaabcc}(\text{COPY})1 \rightarrow k^0_{aaabcc}(aa), \\ k^0_{aaabcc}(1b)2 \rightarrow k_{aaabcc}(b), & k_{aaabcc}(1b)2 \rightarrow j_{aaabcc}(bb), \\ j_{aaabcc}(1b)3 \rightarrow h_{aaabcc}(bc), & k_{aaabcc}(1b)2 \rightarrow v_{aaabcc}(bb), \\ v_{aaabcc}(1b)3 \rightarrow q_{aaabcc}(bc), & q_{aaabcc}(\text{COPY}) \rightarrow p^0_{aaabcc}(a), \\ p^0_{aaabcc}(1b)2 \rightarrow p_{aaabcc}(b), & p_{aaabcc}(1b)3 \rightarrow q_{aaabcc}(bc), \\ q_{aaabcc}(1b)2 \rightarrow g_{aaabcc}(b) & \end{array} \Big\}.$$

(f) $g(1C) \rightarrow f(c), pd\_state = \#a\#C\#, pd\_init = \#b\#C\#, cycle\_paths = \emptyset$

$$R'_{aaabcc} = \Big\{ \begin{array}{ll} s_{aaabcc}() \rightarrow h_{aaabcc}(), & h_{aaabcc}(\text{COPY})1 \rightarrow k^0_{aaabcc}(aa), \\ k^0_{aaabcc}(1b)2 \rightarrow k_{aaabcc}(b), & k_{aaabcc}(1b)2 \rightarrow j_{aaabcc}(bb), \\ j_{aaabcc}(1b)3 \rightarrow h_{aaabcc}(bc), & k_{aaabcc}(1b)2 \rightarrow v_{aaabcc}(bb), \\ v_{aaabcc}(1b)3 \rightarrow q_{aaabcc}(bc), & q_{aaabcc}(\text{COPY}) \rightarrow p^0_{aaabcc}(a), \\ p^0_{aaabcc}(1b)2 \rightarrow p_{aaabcc}(b), & p_{aaabcc}(1b)3 \rightarrow q_{aaabcc}(bc), \\ q_{aaabcc}(1b)2 \rightarrow g_{aaabcc}(b), & g_{aaabcc}(1b)3 \rightarrow f_{aaabcc}(c) \end{array} \Big\}.$$

(g) $SET, pd\_state = \#a\#c\#, pd\_init = \#b\#b\#, cycle\_paths = \emptyset$

$$R'_{aaabcc} + \{s(SET) \rightarrow s_{aaabcc}(\#b\#b\#)\}$$

3. *Path ← (6,12,13)*

(a) $s(1S) \rightarrow h(AC), pd\_state = S\#, pd\_init = S\#, cycle\_paths = \emptyset$

$$R'_{aabc} = \Big\{ s_{aabc}() \rightarrow h_{aabc}() \Big\}.$$

(b) $h(1A) \rightarrow n(aab), pd\_state = AC\#, pd\_init = AC\#, cycle\_paths = \{7, 8, 9\}$

$$R'_{aabc} = \Big\{ \begin{array}{ll} s_{aabc}() \rightarrow h_{aabc}(), & h_{aabc}(\text{COPY})1 \rightarrow k^0_{aabc}(aa), \\ k^0_{aabc}(1b)2 \rightarrow k_{aabc}(b), & k_{aabc}(1b)2 \rightarrow j_{aabc}(bb), \\ j_{aabc}(1b)3 \rightarrow h_{aabc}(bc), & h_{aabc}(1b)2 \rightarrow n_{aabc}(aab) \end{array} \Big\}.$$

(c) $n(1C) \to f(c), pd\_state = \#aab\#C\#, pd\_init = \#b\#C\#$, $cycle\_paths = \emptyset$

$$R'_{aabc} = \Big\{ \begin{array}{l} s_{aabc}() \to h_{aabc}(), \quad h_{aabc}(\text{COPY})1 \to k^0_{aabc}(aa), \\ k^0_{aabc}(1b)2 \to k_{aabc}(b), \quad k_{aabc}(1b)2 \to j_{aabc}(bb), \\ j_{aabc}(1b)3 \to h_{aabc}(bc), \quad h_{aabc}(1b)2 \to n_{aabc}(aab), \\ n_{aabc}(1c)3 \to f_{aabc}(c) \end{array} \Big\}.$$

(d) $SET, pd\_state = \#aab\#c\#, pd\_init = \#b\#b\#$, $cycle\_paths = \emptyset$

$$R'_{aabc} + \{s(SET) \to s_{aabc}(\#b\#b\#)\}$$

Jakmile je dokončena konstrukce všech pravidel, obdržíme finální instanci modelu OPDPDA,

$$_3M_{OPDPDA_2} = (Q', \Sigma, R', s, b, \#, F')$$

, kde

- $Q' = \{q|\exists\alpha,\beta,p,n,m : q(n\alpha)m \to p(\beta) \in R'\} \cup \{q|\exists\alpha,\beta,p,n,m : p(n\alpha)m \to q(\beta) \in R'\} \cup \{q|\exists,\beta,p,n,m : q(COPY)m \to p(\beta) \in R'\} \cup \{q|\exists,\beta,p,n,m : p(COPY)m \to q(\beta) \in R'\} \cup \{q|\exists,\beta,p,n,m : q(SET)m \to p(\beta) \in R'\} \cup \{q|\exists,\beta,p,n,m : p(SET)m \to q(\beta) \in R'\}$
- $R' = R'_{ac} \cup R'_{aabc} \cup R'_{aaabcc}$
- $F' = \{f_{ac}, f_{aabc}, f_{aaabcc}\}$

Se zvyšující se složitostí simulovaného DPDA roste i počet potřebných zásobníků. Tato podkapitola předvedla, jak původní model analyzovat a přesně převést do nového alternativního modelu OPDPDA bez ztráty či modifikace charakteristiky přijímaného jazyka.

Modely OPDPDAs však nedokážou simulovat DPDAs, které obsahují cyklická pravidla rozšiřující množství nevstupních symbolů v zásobnících. Takové DPDAs mohou díky těmto pravidlům generovat libovolně dlouhé, potenciálně nekonečné, posloupnosti nevstupních symbolů a tyto symboly dále individuálně rozepisovat na další potenciálně nekonečné posloupnosti vstupních symbolů. Modely OPDPDAs by k tomu musely disponovat nekonečným množstvím zásobníků, neboť každý vygenerovaný nevstupní symbol simulovaného modelu by vyžadoval v OPDPDA v horším případě dva zásobníky. Jelikož by těchto symbolů mohlo být nekonečně mnoho, tak i s tím by bylo zapotřebí nekonečně mnoho zásobníků a pravidel nad nimi operující v simulujícím OPDPDA.

Aby byl model OPDPDA schopen přijímat jazyky definované těmito DPDA automaty, musel by disponovat mechanismem, který by mu umožňoval průběžně během běhu automatu uvolňovat symboly z některých či všech jeho zásobníků s následnou re-inicializací. Toto by bylo prováděno vždy na konci dogenerování podřetězců, které jsou na sebe nějak spjaté, dle pravidel samotného jazyka. Bez této schopnosti nemůže automat představený v této práci simulovat všechny typy DPDAs a množina jazyků, které takový automat přijímá, je tedy podmnožinou množiny jazyků přijímané standardními modely DPDAs. V následující části je představen způsob simulace mezi modely v opačném směru.

4.5 Simulace OPDPDA pomocí DPDA

Tato kapitola představuje, postup konstrukce standardního modelu DPDA simulujícího libovolný simulovaný model OPDPDA $M_{nOPDPDA}$. Nechť

$$M_{nOPDPDA} = (Q, \Sigma, R, s, \tau, \xi, F)$$

kde se bez ztráty obecnosti předpokládá, že simulovaný model OPDPDA je definován stejným způsobem, jak byl popsán v předchozí kapitole 4.4.

Konstruktor na začátku zpracování každého stromu $path \in paths$[4] inicializuje proměnnou `pd_state` $= S\#$ simulující aktuální stav zásobníku DPDA a nastaví zásobníkovou abecedu $\Gamma = \Sigma \cup \{\xi, S\}$, kde S je zároveň počáteční nevstupní symbol simulujícího DPDA.

Konstruktor poté pokračuje daným stromem tak, že nejprve zpracuje pravidlo `SET` které se nachází na začátku každého stromu modelu OPDPDA. Pravidlo $r : s(SET) \rightarrow p_{path}(v)$ je do simulujícího modelu transformováno jako $R' = R' \cup \{s(1S) \rightarrow p_{path}(S)\}$, přičemž $\Gamma = \Gamma \cup \{w\}$, kde w je nevstupní symbol reprezentující řetězec v. Dále je aktualizován `pd_state` přepsáním prvního nevstupního symbolu S na nově vytvořený symbol w.

Z tohoto nevstupního symbolu se během konstrukce simulujícího automatu odvozují další interní symboly, které vznikají rozdělením konfigurace zásobníků simulovaného OPDPDA, kterou tento symbol představuje. Získané znaky pak konstruktor na základě pravidel dále rozděluje a modifikuje tím simulovaný stav zásobníku `pd_state`.

Příklad 4.5.1 *Nechť OPDPDA je simulovaný model, kterému z počátečního stavu vychází libovolný počet stromů. Uvažujme jeden takový strom, začínající přechodem reprezentující použití pravidla*

$$s(SET) \rightarrow p(\#a\#a\#),$$

kde s je počáteční stav, $p \in Q$, $a \in \tau$, $\# \in \xi$.

Při konstrukci simulujícího automatu DPDA vznikne nevstupní symbol

$$\langle \#a\#a\# \rangle \in \Gamma,$$

který reprezentuje konfiguraci zásobníků modelu OPDPDA pro daný strom.

Z tohoto symbolu mohou dále vznikat nové symboly dělením samotné konfigurace zásobníků:

$$\langle \#a \rangle, \quad \langle \#a\# \rangle \in \Gamma.$$

Nyní následuje předvedení algoritmu, který z existujícího simulovaného modelu OPDPDA sestaví ekvivalentní DPDA. Stejně jako v předchozím algoritmu jsou zde procházeny lineárně všechny cesty vedoucí z počátečního stavu do koncových stavů napříč jednotlivými stromy. Během průchodu je postupně upravována množina nevstupních symbolů automatu i jejich aktuální zastoupení v simulovaném zásobníku proměnné `pd_state`. Konstruktor si takto při průchodu zachovává informaci o stavu zásobníku, pro korektní generování nových pravidel do množiny R'.

[4] Poznámka: Paths reprezentují sekvence pravidel množiny R, reprezentující cesty z počátečního stavu do koncových.

Algorithm 3 Simulace OPDPDA_n pomocí DPDA_n

Input: $M_{\text{OPDPDA}_n} = (Q, \Sigma, R, s, \tau, \xi, F)$
Output: $M_{\text{DPDA}_n} = (Q', \Sigma, \Gamma, R', s, S, F')$
for *path* $\in$ *paths* **do**
  pd_state $\leftarrow S\xi$ **for** *edge* $\in$ *path* **do**
    for *cycle_path* $\in$ *cycle_paths* **do**
      for *cycle_edge* $\in$ *cycle_path* **do**
        pd_state, *expands* $\leftarrow$ `analyze_transition`(*pd_state*, `cycle_edge`, *expands*)
      end
    end
    pd_state, *expands* $\leftarrow$ `analyze_transition`(*pd_state*, `edge`, *expands*)
    expands $\leftarrow$ `insert_expands`(`q` in `edge` $= q(\cdots \rightarrow \ldots$, *expands*) **for** *cycle_path* $\in$ *cycle_paths* **do**
      for *cycle_edge* $\in$ *cycle_path* **do**
        `insert_transition`(pd_state, `cycle_edge`, True)
      end
    end
    `insert_transition`(pd_state, `edge`)
    if *edge* $= \cdots \rightarrow q(\omega)$ ***where*** $q \in F$ **then**
      for *tuple* $(q, p, count, symbol, value) \in$ *final_changes* **do**
        forall $(q(count,\ symbol) \rightarrow p(\omega')) \in R'$ **do**
          $\omega' \leftarrow value \cdot \omega'_{2\ldots|\omega'|}$
        end
      end
    end
  end
end

Nechť existuje jedna konkrétní cesta z počátečního ke koncovému stavu. Algoritmus postupným procházením pravidel dané cesty navštěvuje jednotlivé stavy a na každém kroku nejprve analyzuje:

1. všechny cykly vycházející z daného stavu,
2. přechod, který vede do následujícího stavu v rámci *path*.

Cílem těchto analýz je analyzovat tato proveditelná pravidla a podle toho případně rozšířit zastoupení nevstupních symbolů v množině Γ', jak bylo předvedeno v příkladu 4.5.4, a tyto změny následně promítnout do simulovaného zásobníku `pd_state`, rozepsáním cílených nevstupních symbolů na nově vzniklé (generované) symboly. Proces těchto analýz je dále popsán v následujícím textu.

V rámci funkce `insert_expands` se do množiny R' simulujícího modelu DPDA konstruují nová pravidla, která mezi předchozím a aktuálním stavem na zpracovávané cestě lineárně rozšiřují zastoupení nevstupních symbolů v zásobníku, podle výsledků předchozí analýzy.

Konkrétně, pokud analýza odhalí, že některý nevstupní symbol, reprezentující část konfigurace zásobníku OPDPDA, je potřebné rozdělit na více symbolů 4.5.4, pak je aktualizována proměnná `pd_state`, přepsáním původního nevstupního symbolu na nově vytvořené.

Zároveň je zachována informace o indexu n tohoto přepisovaného symbolu, načež je vytvořeno nové pravidlo pro množinu R', které mezi předchozím a aktuálním stavem přepisuje symbol S na indexu n na posloupnost SS. Podle výsledků analýz může těchto lineárně za sebou aplikovatelných pravidel vzniknout více, přičemž také dochází k rozšíření stavového prostoru automatu, reprezentovaného množinou Q'.

Poté následuje přetvořit samotná analyzovaná pravidla, která reprezentují přechody v původním simulovaném modelu, tak, že.

1. všechna cyklická pravidla vycházející z aktuálního stavu,

2. pravidlo odpovídající přechodu do následujícího stavu na zvolené cestě.

Pokud by konstruktor nad každou cestou vykonával tuto sérii kroků až do dosažení koncového stavu a v tento moment by svůj průběh ukončil, výsledný automat by zůstal neúplný, protože v případě jeho spuštění a dosažení koncového stavu by na jeho zásobníku zůstalo m nevstupních symbolů, které by nebyly odstraněny [5], kde $m = \#_\tau(w)$ a w je pravá strana pravidla `SET` nad daným stromem. Nelze totiž dopředu určit, kdy se nevstupní symbol přepíše ve vstupní. Proto je nutné průběžně zaznamenávat, do jakých symbolů byl každý vstupní symbol v simulovaném modelu naposledy rozepsán, zejména jeho nejlevější část, která se dále považuje za symbol pro budoucí přepisy, který je v simulujícím modelu nahrazen nevstupním symbolem.

Na základě této informace je potřebné upravit při dosažení koncového stavu pravidla, která operovala s nevstupními symboly [6] tak, že se upraví pravá strana přepsáním prvního nevstupního symbolu na prvním indexu na původní nejlevější vstupní symbol pravidla simulovaného modelu.

Příklad 4.5.2 *Nechť M_{opdpda} je simulovaný model OPDPDA,*

$$M_{opdpda} = \{\{q, f\}, \{a, b, c\}, \{s(1a)1 \to f(abc)\}, s, a, \#, \{f\}\}$$

potom v momentě, kdy konstruktor dosáhne stavu $f \in F$, simulující model DPDA nabývá stavu množiny pravidel $R' = \{s(1S) \to f(Sbc)\}$.

Následně dojde k modifikaci pravidel R', což pro pravidlo

$$s(1S) \to f(Sbc)$$

znamená modifikaci do stavu

$$s(1S) \to f(abc)$$

dle nejlevější strany pravidla simulovaného modelu

$$s(1a)1 \to f(abc)$$

.

Další části této podkapitoly se detailněji zaměří na analýzu pravidel simulovaného modelu a následnou konstrukci do podoby simulujícího modelu DPDA.

[5] Poznámka: Takové které nebyly přepsány na posloupnost $v \in \Sigma^*$

[6] Poznámka: Pro každý nevstupní symbol existuje jedno takové pravidlo

Algorithm 4 Funkce `analyze_transition`

Function `analyze_transition`(pd_state, cycle_edge, $expands$)**:**

```
if cycle_edge = q(SET)m → p(ω) then
    pd_state ← ⟨ω⟩ ξ  Γ ← Γ ∪ {⟨ω⟩}
end
else if cycle_edge = q(na)m → p(ω) then
    count ← 0  find_pd ← m  find_ix ← n  for i ← 0 to |pd_state| do
        c ← pd_state_i  if c ∈ Δ then
            count ← count + 1  for ix ← 0 to |c| do
                s ← c_ix  if s = ξ then
                    find_pd ← find_pd − 1
                end
                else if s ∈ Σ ∧ find_pd = 1 ∧ find_ix = 1 then
                    Γ ← Γ ∪ {⟨c_0...ix⟩}  seq ← ⟨c_0...ix⟩ if |c_ix+1...end| > 1 then
                        Γ ← Γ ∪ {⟨c_ix+1...end⟩}  seq ← seq‖⟨c_ix+1...end⟩  expands ←
                        expands‖count
                    end
                    splitted ← true  break
                end
                else if s ∈ Σ ∧ find_pd = 1 then
                    find_ix ← find_ix − 1
                end
            end
            if splitted then
                pd_state ← (pd_state_1...i−1) ‖ seq ‖ (pd_state_i+1...end)  break
            end
        end
    end
end
else if cycle_edge = q(COPY)m → p(ω) then
    count ← 0  find_pd ← m  for i ← 1 to |pd_state| do
        c ← pd_state_i  if c ∈ Δ then
            count ← count + 1  for ix ← 1 to |c| do
                symbol ← c_ix  if symbol = ξ ∧ find_pd = 1 then
                    Γ ← Γ ∪ {⟨c_ix...end⟩}  seq ← ⟨c_ix...end⟩  if ix > 1 then
                        Γ ← Γ ∪ {⟨c_1...ix−1⟩}  seq ← ⟨c_1...ix−1⟩ ‖ seq  expands ←
                        expands‖count
                    end
                    splitted ← true  break
                end
                else if symbol = ξ then
                    find_pd ← find_pd − 1
                end
            end
            if splitted then
                pd_state ← pd_state_1...i−1 ‖ seq ‖ pd_state_i+1...end  break
            end
        end
    end
end
return pd_state, expands
```

Algoritmus výše popisuje funkci `analyze_transition`, která analyzuje jedno vstupní pravidlo *edge* a upravuje aktuální simulovaný stav zásobníků `pd_state` a konstruuje nové nevstupní symboly, které přidá do zásobníkové abecedy Γ'. Proces probíhá následovně. Nejprve funkce určí typ pravidla r:

- *Pravidlo SET:* Pokud $r : q(SET) \to p(\omega)$, pak celou konfiguraci `pd_state` nahradí jediným nevstupním symbolem $\langle\omega\rangle$, následovaným symbolem dna ξ. Tento nový zakódovaný symbol se rovněž přidá do zásobníkové abecedy $\Gamma' = \Gamma' \cup \{\langle\omega\rangle\}$.
- *Běžné pravidlo:* Pokud $r : q(na)m \to p(\beta)$) prochází se lineárně každý symbol aktuální konfigurace zásobníku `pd_state` simulujícího modelu DPDA:
  1. Pokud je symbol vstupní ($\in \Sigma$), ignoruje se.
  2. Pokud je symbol nevstupní ($\in \Delta$), dekóduje se pro získání konfigurace ω a v ní se hledá vstupní symbol $a \in \Sigma$ na pořadovém indexu n v zásobníku m.
     (a) Pokud ve zkoumané části symbol s požadovanými indexy neexistuje (např. hledáme index 3, ale nevstupní symbol modeloval pouze zásobníky s indexy 1–2), přejde se k dalšímu symbolu a proces pokračuje.
     (b) Jakmile je hledaný symbol nalezen, rozdělí se část ω reprezentovaná nevstupním symbolem na prefix a zbytek. Pokud je zbytek delší než 1 (neobsahuje pouze znak dna), vzniknou dva nové nevstupní symboly, v opačném případě se nic neděje. V prvním případě se oba podřetězce zakódují jako nové nevstupní symboly $\langle\omega_{1\ldots i}\rangle$ a $\langle\omega_{i+1\ldots|\omega|}\rangle$, uloží do Γ' a zaznamená se, který symbol byl rozšířen na více symbolů pro pozdější tvorbu pravidel lineárně generující další nevstupní symboly v simulujícím DPDA pomocí funkce `insert_expands`.
- *Pravidlo COPY:* Pokud platí $r : q(COPY)m \to p(\omega)$, postup je podobný běžnému pravidlu, ale místo vyhledávání vstupního symbolu se hledá symbol dna ξ s indexem m. Jakmile se dosáhne dna, příslušný nevstupní symbol $\langle u\,\xi_m\,v\rangle$, kde $u, v \in (\Gamma \cup \{\xi\})^*$ a ξ_m je m-tý znak v zásobníku, se rozdělí na dva nové symboly $\langle v\rangle$ a $\langle\xi_m u\rangle$, oba se přidají do Γ' a opět se zaznamená jaký symbol s jakým indexem je potřebné rozepsat. V případě že prefix $\#(v) = 0$, nic se neděje.

Po analýze všech hran vycházejících z daného stavu (tj. cyklických i těch, které vedou dále po aktuálně zpracovávané cestě) má konstrukční algoritmus k dispozici informace o aktuálním požadovaném stavu zásobníku modelu DPDA. To zahrnuje množství, pořadí a konkrétní typy nevstupních symbolů, které by se měly v zásobníku nacházet.

Na základě těchto informací je nyní nezbytné doplnit množinu pravidel R' o taková pravidla, která lineárně zajistí požadované navýšení nevstupních symbolů v zásobníku.

Algorithm 5 Funkce `insert_expands`

Function `Insert_expands`(*node, expands*)**:**

```
if |expands| > 0 then
    foreach r : ··· → p(...) ∈ R′ such that p = node do
        p ← p‖0
    end
    for ix, count ∈ enumerate(expands) do
        if ix + 1 = |expands| then
            R′ ← R′ ∪ {(node‖ix)(count S) → node(SS)}
        end
        else
            R′ ← R′ ∪ {(node‖ix)(count S) → (node‖ix + 1)(SS)}
        end
    end
end
return expands ← []
```

Algoritmus výše popisuje, jak konstruktor jednoduše a efektivně pro daný strom vytvoří nová pravidla, která rozšiřují působnost nevstupních symbolů v zásobníku. V případě, že je n počet těchto pravidel pro navýšení zastoupení nevstupních symbolů v zásobníku větší než nula, bude vytvořeno n nových mezistavů množiny Q', mezi kterými budou konstruována tato pravidla, která tyto expanze $\in \Delta$ symbolů provádějí. Přičemž poslední pravidlo v pořadí přechází do aktuálního stavu a pravidlo vycházející z předchozího stavu bude přecházet do prvního stavu v pořadí těchto nových mezistavů.

Příklad 4.5.3 *Nechť M_{dpda} je právě konstruovaný simulující model DPDA, s aktuální množinou pravidel $R' = \{s(1S) \to q(Sa)\}$, simulovaným stavem zásobníku* `pd_state` $= \langle\tau\#\tau\#\rangle a$.

Konstruktor právě přešel do stavu q a provedl analýzu všech cyklických pravidel a následujícího pravidla v rámci cesty. Bylo zjištěno, že nevstupní symbol $\langle\tau\#\tau\#\rangle$ na indexu 1 byl rozdělen na dva nové symboly $\langle\tau\rangle, \langle\#\tau\#\rangle \in \Gamma$ a `pd_state` $= \langle\tau\rangle\langle\#\tau\#\rangle a$.

Na základě analýzy bude konstruováno jedno pravidlo, které rozšiřuje zastoupení nevstupních symbolů v zásobníku.

Vznikne další vnitřní stav $Q' = Q' \cup \{q_0\}$ a pravidlo $s(1S) \to q(Sa)$ bude nyní odkazovat na tento nový stav. Následně zbývá přidat nové pravidlo

$$R' = R' \cup \{q_0(1S) \to q(SS)\}$$

Tímto vznikne nový stav množiny pravidel

$$R' = \{s(1S) \to q_0(Sa), q_0(1S) \to q(SS)\}$$

Algorithm 6 Funkce `insert_transition`

```
Function Insert_transition(edge, pd_state, final_changes, cycle ← false):
    if edge = q(COPY)m → p(ω) then
        count ← 0  pd_index ← m  for char ∈ pd_state do
            if char ∈ Δ then
                count ← count + 1  for symbol ∈ char do
                    if symbol = ξ ∧ pd_index = 1 then
                        R′ ← R′ ∪ { q( count$) → p( ω$)}
                    else if symbol = ξ then
                        pd_index ← pd_index − 1
                    end
                end
            end
        end
    else if edge = q(na)m → p(ω) then
        pd_index ← m  ch_index ← n  count ← 0  for (ix, char) ∈ enumerate(pd_state)
        do
            if char ∈ Δ then
                count ← count + 1  for symbol ∈ char do
                    if symbol = a ∧ ch_index = 1 ∧ pd_index = 1 then
                        R′ ← R′ ∪ { q( count$) → p( $ ω_{2...|ω|})}  ch_index ← ch_index − 1  if
                        cycle = false then
                            pd_state ← pd_state_{1...ix} ‖ ω_{2...|ω|} ‖ pd_state_{ix+2...end}
                            final_changes[count] ← (q, p, count, $, ω_1)
                        end
                    else if symbol = ξ then
                        pd_index ← pd_index − 1
                    end
                    else if symbol ∈ Σ ∧ pd_index = 1 then
                        ch_index ← ch_index − 1
                    end
                end
            end
        end
    end
    else
        R′ ← R′ ∪ { q() → p()}
    end
```

Algoritmus výše představuje finální funkci vytvářející transformovaná pravidla simulovaného modelu OPDPDA do podoby simulujícího modelu DPDA, splňující charakteristiky původního modelu. Funkce nejprve identifikuje typ vstupního pravidla a lokalizuje nevstupní symbol, jenž má být na základě analýzy původního pravidla rozepsán. Aktuální simulovaný stav zásobníku simulujícího DPDA je díky předchozí analýze již nastaven, a proto je pro konstrukci nového pravidla potřebné pouze zjistit index přepisovaného nevstupního symbolu v zásobníku, který cílený přepisovaný vstupní symbol původního vstupního pravidla obsahuje.

V případě vstupního pravidla $q(COPY)m \to p(\omega)$ typu `COPY` se po nalezení chtěného nevstupního symbolu A v `pd_state` a jeho indexu n, kde A je n-tý nevstupní symbol v `pd_state`, indexováno od 1, vytvoří nové pravidlo $R' = R' \cup \{q(nS) \to p(\omega S)\}$. Je vhodné si všimnout, že ve všech pravidlech nového modelu se pracuje vždy pouze s jedním nevstupním symbolem a to počátečním S.

Pokud se jedná o běžné pravidlo $q(j\tau)m \to p(\omega)$, pak se nové pravidlo konstruuje obdobně jako u pravidel typu `SET` tzn. nejprve se najde příslušný symbol A v `pd_state` s indexem n a následně se vytvoří pravidlo $R' = R' \cup \{q(nS) \to p(S\omega_{2\ldots|\omega|})\}$, ve kterém je na prvním místě nevstupní symbol S, následovaný přepisovacím řetězcem původního pravidla zkráceným o první znak.

Jestliže je toto běžné pravidlo tvořeno v rámci konkrétní cesty (není cyklické), dojde rovněž k uložení informace, že se jedná o poslední pravidlo, které přepisovalo nevstupní symbol s indexem n, a dále si algoritmus zapamatuje první znak ω_1 pravé strany původního pravidla. Tyto údaje poslouží později k úpravě konečných pravidel a zabránění vzniku nežádoucích zbytkových nevstupních symbolů popsaných u hlavního algoritmu.

V případě jiného typu pravidla se vytvoří ε-přechod, tj. přechod, který pouze přechází mezi stavy bez jakéhokoli zásahu do zásobníku.

Příklad 4.5.4 *Nechť* $M_{2OPDPDA} = (Q, \{a,b,c\}, R, s, b, \#, \{f_{ac}, f_{aabc}, f_{aaabcc}\})$ *je čistý hluboký seřazený 2-zásobníkový automat, kde konečné množině pravidel* R *náleží,*

Přechody pro R'_{ac}

1 :	$s_{ac}()$	$\rightarrow$	$q_{ac}()$
2 :	$q_{ac}(\mathrm{COPY})1$	$\rightarrow$	$p^0_{ac}(a)$
3 :	$p^0_{ac}(1b)2$	$\rightarrow$	$p_{ac}(b)$
4 :	$p_{ac}(2b)2$	$\rightarrow$	$q_{ac}(bc)$
5 :	$q_{ac}(1b)2$	$\rightarrow$	$g_{ac}(a)$
6 :	$g_{ac}(2b)2$	$\rightarrow$	$f_{ac}(c)$
7 :	$s(SET)$	$\rightarrow$	$s_{ac}(\#bb\#)$

Přechody pro R'_{aaabcc}

8 :	$s_{aaabcc}()$	$\rightarrow$	$h_{aaabcc}()$
9 :	$h_{aaabcc}(\mathrm{COPY})1$	$\rightarrow$	$k^0_{aaabcc}(aa)$
10 :	$k^0_{aaabcc}(1b)2$	$\rightarrow$	$k_{aaabcc}(b)$
11 :	$k_{aaabcc}(1b)2$	$\rightarrow$	$j_{aaabcc}(bb)$
12 :	$j_{aaabcc}(1b)3$	$\rightarrow$	$h_{aaabcc}(bc)$
13 :	$k_{aaabcc}(1b)2$	$\rightarrow$	$v_{aaabcc}(bb)$
14 :	$v_{aaabcc}(1b)3$	$\rightarrow$	$q_{aaabcc}(bc)$
15 :	$q_{aaabcc}(\mathrm{COPY})$	$\rightarrow$	$p^0_{aaabcc}(a)$
16 :	$p^0_{aaabcc}(1b)2$	$\rightarrow$	$p_{aaabcc}(b)$
17 :	$p_{aaabcc}(1b)3$	$\rightarrow$	$q_{aaabcc}(bc)$
18 :	$q_{aaabcc}(1b)2$	$\rightarrow$	$g_{aaabcc}(b)$
19 :	$g_{aaabcc}(1b)3$	$\rightarrow$	$f_{aaabcc}(c)$
20 :	$s(SET)$	$\rightarrow$	$s_{aaabcc}(\#b\#b\#)$

Přechody pro R'_{aabc}

21 :	$s_{aabc}()$	$\rightarrow$	$h_{aabc}()$
22 :	$h_{aabc}(\mathrm{COPY})1$	$\rightarrow$	$k^0_{aabc}(aa)$
23 :	$k^0_{aabc}(1b)2$	$\rightarrow$	$k_{aabc}(b)$
24 :	$k_{aabc}(1b)2$	$\rightarrow$	$j_{aabc}(bb)$
25 :	$j_{aabc}(1b)3$	$\rightarrow$	$h_{aabc}(bc)$
26 :	$h_{aabc}(1b)2$	$\rightarrow$	$n_{aabc}(aab)$
27 :	$n_{aabc}(1c)3$	$\rightarrow$	$f_{aabc}(c)$
28 :	$s(SET)$	$\rightarrow$	$s_{aabc}(\#b\#b\#)$

Konstrukce DPDA:

$$paths = \{(7,1,5,6),(20,8,9,10,13,14,18,19),(28,21,26,27)\}$$

1. *Path* $\leftarrow$ *(7,1,5,6)*

   (a) $s(SET) \rightarrow s_{ac}(\#bb\#), pd\_state = S\#, cycle\_edges = \emptyset$

      i. *analyze_transitions:*

      $$pd\_state = S\# \implies \langle \#bb\# \rangle \#.$$

      ii. *insert_expands:*

      $$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

      iii. *insert_transitions:*

      $$R' \leftarrow R' \cup \{\, s() \rightarrow s_{ac}() \,\}.$$

(b) $s_{ac}() \to q_{ac}(), pd\_state = \langle\#bb\#\rangle\#, cycle\_edges = \emptyset$

i. *analyze_transitions:*

$$pd\_state = \langle\#bb\#\rangle\# \quad (\text{žádná změna stavu zásobníku}).$$

ii. *insert_expands:*

$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. *insert_transitions:*

$$R' \leftarrow R' \cup \{\, s_{ac}() \to q_{ac}() \,\}.$$

(c) $q_{ac}(1b)2 \to g_{ac}(a), pd\_state = \langle\#bb\#\rangle\#, cycle\_edges = \{2, 3, 4\}$

i. *analyze_transitions:*

$$pd\_state = \langle\#bb\#\rangle\# \implies \langle\#b\rangle\langle b\#\rangle\#.$$

ii. *insert_expands:*

$$R' \leftarrow R' \setminus \{s_{ac}() \to q_{ac}()\} \cup \{s_{ac}() \to {}^0q_{ac}(), {}^0q_{ac}(1S) \to q_{ac}(SS)\}.$$

iii. *insert_transitions:*

$$R' \leftarrow R' \cup \{\, q_{ac}(1S) \to q^0_{ac}(aS), q^0_{ac}(S) \to p_{ac}(S), p_{ac}(2S) \to q_{ac}(Sc),$$
$$q_{ac}(1S) \to g_{ac}(S)\}.$$

(d) $g_{ac}(2b)2 \to f_{ac}(c), pd\_state = \langle\#b\rangle\langle b\#\rangle\#, cycle\_edges = \emptyset$

i. *analyze_transitions:*

$$pd\_state = \langle\#b\rangle\langle b\#\rangle\# \quad (\text{žádná změna stavu zásobníku}).$$

ii. *insert_expands:*

$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. *insert_transitions:*

$$R' \leftarrow R' \cup \{\, g_{ac}(2S) \to f_{ac}(S)\}.$$

(e) $Final\_changes = \{(q_{ac}, g_{ac}, 1, S, a), (g_{ac}, f_{ac}, 2, S, c)\}$

i.

$$R' \leftarrow R' \setminus \{q_{ac}(1S) \to g_{ac}(S), g_{ac}(2S) \to f_{ac}(S)\}.$$

ii.

$$R' \leftarrow R' \cup \{q_{ac}(1S) \to g_{ac}(a), g_{ac}(1S) \to f_{ac}(c)\}.$$

2. *Path ← (20, 8, 9, 10, 13, 14, 18, 19)*

(a) $s(SET) \to s_{aaabcc}(\#b\#b\#), pd\_state = S\#, cycle\_edges = \emptyset$

i. *analyze_transitions:*

$$pd\_state = S\# \implies \langle\#b\#b\#\rangle\#.$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\textit{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{\, s() \rightarrow s_{aaabcc}() \,\}.$$

(b) $s_{aaabcc}() \rightarrow h_{aaabcc}(), pd\_state = \langle \#b\#b\# \rangle \#, cycle\_edges = \emptyset$

i. analyze_transitions:

$$pd\_state = \langle \#b\#b\# \rangle \# \quad (\textit{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\textit{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{\, s_{aaabcc}() \rightarrow h_{aaabcc}() \,\}.$$

(c) $h_{aaabcc}(COPY)1 \rightarrow k^0_{aaabcc}(aa), pd\_state = \langle \#b\#b\# \rangle \#,$
$cycle\_edges = \{9, 10, 11, 12\}$

i. analyze_transitions:

$$pd\_state = \langle \#b\#b\# \rangle \# \implies \langle \#b \rangle \langle \#b\# \rangle \#.$$

ii. insert_expands:

$$R' \leftarrow R' \setminus \{s_{aaabcc}() \rightarrow h_{aaabcc}()\} \cup \{s_{aaabcc}() \rightarrow {}^0h_{aaabcc}(),$$
$${}^0h_{aaabcc}(1S) \rightarrow h_{aaabcc}(SS)\}.$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{\, h_{aaabcc}(1S) \rightarrow k^0_{aaabcc}(aaS), k^0_{aaabcc}(S) \rightarrow k_{aaabcc}(S),$$
$$k_{aaabcc}(1S) \rightarrow j_{aaabcc}(Sb), j_{aaabcc}(2S) \rightarrow h_{aaabcc}(Sc) \,\}.$$

(d) $k^0_{aaabcc}(1b)2 \rightarrow k_{aaabcc}(b), pd\_state = \langle \#b \rangle \langle \#b\# \rangle \#,$
$cycle\_edges = \{9, 10, 11, 12\}$

i. analyze_transitions:

$$pd\_state = \langle \#b \rangle \langle \#b\# \rangle \# \quad (\textit{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\textit{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \varnothing \quad (\textit{žádná změna množiny pravidel}).$$

(e) $k_{aaabcc}(1b)2 \to v_{aaabcc}(bb), pd\_state = \langle \#b \rangle \langle \#b\# \rangle \#,$
$cycle\_edges = \{9, 10, 11, 12\}$

i. analyze_transitions:

$$pd\_state = \langle \#b \rangle \langle \#b\# \rangle \# \quad (\text{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{k_{aaabcc}(1S) \to v_{aaabcc}(Sb)\}.$$

(f) $v_{aaabcc}(1b)3 \to q_{aaabcc}(bc), pd\_state = \langle \#b \rangle \langle \#b\# \rangle \#, cycle\_edges = \emptyset$

i. analyze_transitions:

$$pd\_state = \langle \#b \rangle \langle \#b\# \rangle \# \quad (\text{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{v_{aaabcc}(2S) \to q_{aaabcc}(Sc)\}.$$

(g) $q_{aaabcc}(1b)2 \to g_{aaabcc}(a), pd\_state = \langle \#b \rangle \langle \#b\# \rangle \#, cycle\_edges = \{15, 16, 17\}$

i. analyze_transitions:

$$pd\_state = \langle \#b \rangle \langle \#b\# \rangle \# \quad (\text{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{q_{aaabcc}(1S) \to p^0_{aaabcc}(aS), p^0_{aaabcc}(1S) \to p_{aaabcc}(S),$$
$$p_{aaabcc}(2S) \to q_{aaabcc}(Sc), q_{aaabcc}(1S) \to g_{aaabcc}(S)\}.$$

(h) $g_{aaabcc}(1b)3 \to f_{aaabcc}(c), pd\_state = \langle \#b \rangle \langle \#b\# \rangle \#, cycle\_edges = \emptyset$

i. analyze_transitions:

$$pd\_state = \langle \#b \rangle \langle \#b\# \rangle \# \quad (\text{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{g_{aaabcc}(2S) \to f_{aaabcc}(S)\}.$$

(i) $Final\_changes = \{(q_{aaabcc}, g_{aaabcc}, 1, S, a), (g_{aaabcc}, f_{aaabcc}, 2, S, c)\}$

i.
$$R' \leftarrow R' \setminus \{q_{aaabcc}(1S) \rightarrow g_{aaabcc}(S), g_{aaabcc}(2S) \rightarrow f_{aaabcc}(S)\}.$$

ii.
$$R' \leftarrow R' \cup \{q_{aaabcc}(1S) \rightarrow g_{aaabcc}(a), g_{aaabcc}(1S) \rightarrow f_{aaabcc}(c)\}.$$

3. *Path* ← *(28,21,26,27)*

(a) $s(SET) \rightarrow s_{aabc}(\#b\#b\#), pd\_state = S\#, cycle\_edges = \emptyset$

i. *analyze_transitions:*
$$pd\_state = S\# \implies \langle \#b\#b\# \rangle \#.$$

ii. *insert_expands:*
$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. *insert_transitions:*
$$R' \leftarrow R' \cup \{\, s() \rightarrow s_{aabc}() \,\}.$$

(b) $s_{aabc}() \rightarrow h_{aabc}(), pd\_state = \langle \#b\#b\# \rangle \#, cycle\_edges = \emptyset$

i. *analyze_transitions:*
$$pd\_state = \langle \#b\#b\# \rangle \# \quad (\text{žádná změna stavu zásobníku}).$$

ii. *insert_expands:*
$$R' \leftarrow R' \cup \varnothing \quad (\text{žádná změna množiny pravidel}).$$

iii. *insert_transitions:*
$$R' \leftarrow R' \cup \{\, s_{aabc}() \rightarrow h_{aabc}() \,\}.$$

(c) $h_{aabc}(1b)2 \rightarrow n_{aabc}(aab), pd\_state = \langle \#b\#b\# \rangle \#, cycle\_edges = \{9, 10, 11, 12\}$

i. *analyze_transitions:*
$$pd\_state = \langle \#b\#b\# \rangle \# \implies \langle \#b \rangle \langle \#b\# \rangle \#.$$

ii. *insert_expands:*
$$R' \leftarrow R' \setminus \{s_{aabc}() \rightarrow h_{aabc}()\} \cup \{s_{aabc}() \rightarrow {}^0h_{aabc}(), {}^0h_{aabc}(1S) \rightarrow h_{aabc}(SS)\}.$$

iii. *insert_transitions:*
$$R' \leftarrow R' \cup \{\, h_{aabc}(1S) \rightarrow k^0_{aabc}(aaS), k^0_{aabc}(S) \rightarrow k_{aabc}(S),$$
$$k_{aabc}(1S) \rightarrow j_{aabc}(Sb), j_{aabc}(2S) \rightarrow h_{aabc}(Sc),\ h_{aabc}(1S) \rightarrow n_{aabc}(Sab)\}.$$

(d) $n_{aabc}(1c)3 \rightarrow f_{aabc}(c), pd\_state = \langle \#b \rangle \langle \#b\# \rangle \#, cycle\_edges = \emptyset$

i. *analyze_transitions:*
$$pd\_state = \langle \#b \rangle \langle \#b\# \rangle \# \quad (\text{žádná změna stavu zásobníku}).$$

ii. insert_expands:

$$R' \leftarrow R' \cup \varnothing \quad (\textit{žádná změna množiny pravidel}).$$

iii. insert_transitions:

$$R' \leftarrow R' \cup \{\, n_{aabc}(2S) \rightarrow f_{aabc}(S)\}.$$

(e) $Final\_changes = \{(h_{aabc}, n_{aabc}, 1, S, a), (n_{aabc}, f_{aabc}, 2, S, c)\}$

i.

$$R' \leftarrow R' \setminus \{h_{aabc}(1S) \rightarrow n_{aabc}(Sab), n_{aabc}(2S) \rightarrow f_{aabc}(S)\}.$$

ii.

$$R' \leftarrow R' \cup \{h_{aabc}(1S) \rightarrow n_{aabc}(aab), n_{aabc}(1S) \rightarrow f_{aabc}(c)\}.$$

Jakmile je dokončena konstrukce všech pravidel, obdržíme finální instanci modelu DPDA, simulující model OPDPDA,

$$DPDA_2 = (Q', \Sigma, \Gamma, R, s, S, F)$$

, kde

- $Q' = \{q|\exists\alpha, \beta, p, n : q(n\alpha) \rightarrow p(\beta) \in R'\} \cup \{q|\exists\alpha, \beta, p, n : p(n\alpha)m \rightarrow q(\beta) \in R'\}$
- $\Gamma = \Sigma \cup \{S, \#\}$
- $F' = \{f_{ac}, f_{aabc}, f_{aaabcc}\}$

4.6 Přijímací síla

Minulé sekce 4.4 a 4.5 představily, jak je možné některé automaty DPDAs simulovat pomocí OPDPDAs a naopak, jak libovolné automaty OPDPDAs simulovat pomocí DPDAs. Tato práce představila, že existuje třída jazyků, které modely OPDPDAs nemohou přijmout, protože by k tomu potřebovaly nekonečné zastoupení zásobníků s nekonečným množstvím pravidel nad nimi generujícími.

Lemma 4.6.1 *${}_m$**OPDPDA**$_n \subseteq$ **DPDA**$_n$, pro n,m* ≥ 1

Důkaz. Mějme libovolný OPDPDA$_n$, u kterého lze bez ztráty obecnosti předpokládat, že splňuje vlastnosti korektně konstruovaného automatu popsané v předchozích sekcích. Tento automat lze simulovat pomocí algoritmu 4.5 zkonstruovaného DPDA$_n$, přičemž je zachován stupeň zanořování stavů v cyklech.

Nechť ${}_nM$, je ${}_m$OPDPDA$_n$, kde $n \geq 1$ a

$${}_nM = (Q, \Sigma, R, s, \tau, \xi, F),$$

Bez ztráty obecnosti se předpokládá, že ${}_nM$ obsahuje m zásobníků, které jsou seřazeny vzestupně, tj. $1 \preceq 2 \preceq \cdots \preceq m$. Nyní si představíme ${}_nM'$ reprezentující model DPDA$_n$. Nechť

$${}_nM' = (Q', \Sigma, \Gamma, R', s, S, F),$$

Pomocí konstruktoru dojde k analyzování vstupního automatu a generování hodnot, definujících nový simulující automat DPDA$_n$.

Konstruktor vždy skončí úspěšně, neboť jeho úkolem je pouze prohledávat stavový prostor, vytvářet nové nevstupní symboly podle aktuálního rozdělení počáteční konfigurace zásobníků a transformovat či vytvářet nová pravidla pro simulující DPDA$_n$.

V novém modelu je zachováno původní zanoření stavů ve všech cyklech stromů vycházejících z počátečního stavu. To znamená, že oba modely dosahují stejného stupně zanoření a přijímají tutéž třídu jazyků.

Jediná pravidla vytvořená zcela nově tj. nikoli pouhou transformací, jsou taková, která rozšiřují zastoupení nevstupních symbolů v zásobníku. Nacházejí se však výhradně na cestách stromů od počátečního stavu ke koncovým, nikoli v cyklech. Tyto pravidla jako jediná rozšiřují stavový prostor množiny Q' oproti Q.

S pomocí algoritmu 4.4 následuje konstrukce OPDPDA$_n$ simulující model DPDA$_n$ vytvořený v předchozím kroku. Konstruktor opět vždy skončí úspěchem, neboť v automatu ${}_nM'$ neexistují pravidla, která by cyklicky navyšovala zastoupení nevstupních symbolů v zásobníku.

Nechť ${}_nM''$, je OPDPDA$_n$, kde $n \geq 1$ a

$$ {}_nM'' = (Q'', \Sigma, R'', s, \tau, \xi, F), $$

Při porovnání modelů ${}_nM''$ a ${}_nM$ lze zjistit, že jediným rozdílem je rozšíření množiny přechodových pravidel o ε-pravidla na cestách stromů. Toto rozšíření pravidel však nemění jazyk, který automaty přijímají.

Cykly a jejich zanoření zůstávají identické, což implikuje zachování stupně mezi oběma modely i po sérii transformací mezi modely.

Lemma 4.6.2 $\boldsymbol{DPDA_n - {}_mOPDPDA_n \neq \emptyset}$, *pro* $n, m \geq 1$

Důkaz. Mějme automat DPDA$_n$, jehož množina pravidel R obsahuje přechody, jež cyklicky navyšují počet nevstupních symbolů v zásobníku.

Pokud nad takovým automatem dojde ke spuštění konstruktoru 4.4 pro vytvoření simulujícího modelu ${}_m$OPDPDA$_n$, konstruktor svůj proces generování nikdy nedokončí. Není totiž schopen zkonstruovat všechny cesty (stromy) vedoucí z počátečního stavu ke koncovým, protože by při každé nové cestě nekonečně prodlužoval délku cesty i množství potřebných zásobníků. Z toho plyne, že existují jazyky, které nelze přijmout modely OPDPDAs, ale naopak lze modely DPDAs, neboť není možné realizovat automat s nekonečným počtem pravidel, stavů a zásobníků.

Lemma 4.6.3 $\boldsymbol{RG \subseteq {}_mOPDPDA_n}$, *pro* $n,m \geq 1$

Důkaz. Mějme libovolný konečný automat M.

$$ M = (Q, \Sigma, R, s, F), $$

Pro konečné automaty platí **KA** = **RG**. Nechť ${}_nM'$, je ${}_m$OPDPDA$_n$, kde $n, m \geq 1$ a

$$ {}_nM' = (Q', \Sigma, R', s, \tau, \xi, F'), $$

Potom pro každé pravidlo $r : qx \to p \in R$, je zkonstruováno pravidlo

$$ r' : q(COPY)1 \to p(x) \in R' $$

Dále $F' = \{f_{uid}\}$, $\forall f \in F : f(1\tau)2 \to f_{uid}(\varepsilon)$, $Q' = Q \cup \{f_{uid}\}$, $\tau = \Sigma$, $\xi = \#$.

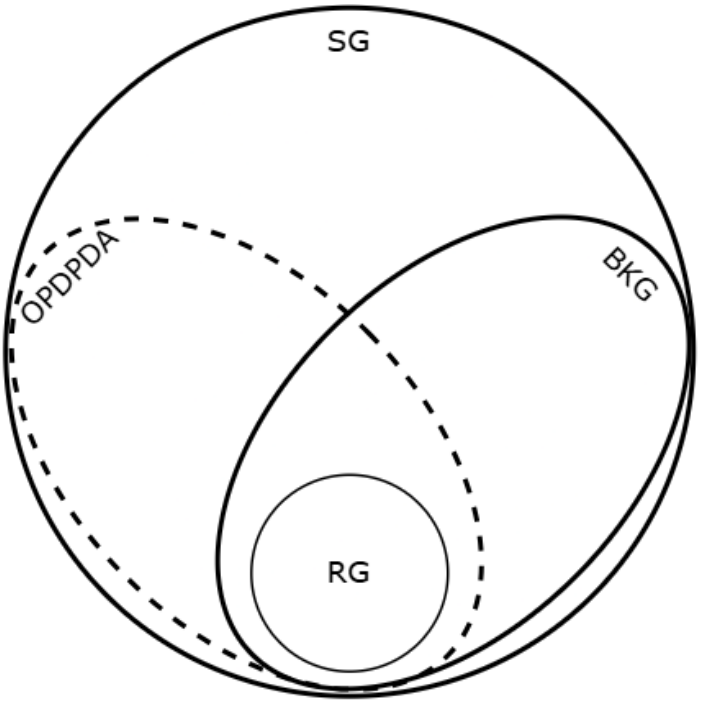
Lemma 4.6.4 $_m\boldsymbol{OPDPDA}_n - \boldsymbol{RG} \neq \emptyset,\ pro\ n,m \geq 1$

Důkaz. Jak již bylo dokázáno v lemmatu 4.6.1, modely OPDPDAs jsou schopny přijímat jazyky, které konečné automaty nemohou.

Existuje tedy třída jazyků přijímaná modely OPDPDAs avšak negenerovatelná regulárními gramatikami.

Věta 4.6.1 $\boldsymbol{RG} = \boldsymbol{KA} \subset {}_m\boldsymbol{OPDPDA}_n \subset \boldsymbol{DPDA}_n = \boldsymbol{SG}_n,\ pro\ n,m \geq 1$

Tímto bylo dokázáno, že jazykově-přijímací modely OPDPDAs, leží svou silou mezi regulárními a stavovými gramatikami, přičemž existuje třída jazyků generovaná bezkontextovými gramatikami a stavovými gramatikami menšího či stejného stupně, kterou tento model nepřijme.



Obrázek 4.1: Abstraktní porovnání síly OPDPDA s klasickými modely

Kapitola 5

Implementace

V této části, která představuje praktickou experimentální část práce, bude představeno samotné použití a implementace simulačních a transformačních algoritmů popsaných v předchozí kapitole 4. Tyto algoritmy budou integrovány do plně funkční a modifikovatelné aplikace, jejímž cílem je na základě vstupního automatu, předaného ve strukturované formě, provést oboustrannou simulaci, tedy konstrukci ekvivalentních automatů mezi modely DPDAs a OPDPDAs. Výstupem bude strukturovaný popis těchto automatů a jejich grafická reprezentace, která přispěje k lepší srozumitelnosti a pochopení výsledků.

5.1 Návrh aplikace

Jelikož je cílem aplikace samotná transformace mezi modely s následným výstupem ve formě zvoleného strukturovaného zápisu reprezentujícího simulující model a jeho grafickou reprezentaci pro lepší a přehlednější vizualizaci modelu, byla zvolena realizace ve formě jednoduché konzolové aplikace v jazyce `Python 3`[13]. Tato aplikace je implementována jako `Python` knihovna, jejímž úkolem je načíst strukturovaný zápis automatu ve formátu `.json`[1], interně jej převést do grafové reprezentace pomocí specializovaných modulů a následně provést sérii analytických a transformačních kroků vedoucích ke konstrukci ekvivalentních automatů mezi modely DPDAs a OPDPDAs.

Implementace umožňuje export výstupních modelů jak ve strukturovaném formátu `.json`, tak i ve formě grafické reprezentace ve formátu `.png`.

Každá transformace mezi modely pracuje ve stejném pořadí hlavních operací. Aplikace nejprve načte vstupní automat ve formátu `.json`, ten následně zparsuje do interní datové struktury v jazyce Python, z níž poté vytvoří grafovou strukturu prostřednictvím modulů `Graph` a `Pure_Graph`. Z těchto struktur je následně vygenerována `.png` reprezentace automatu odpovídající danému typu a chování modelu. Samotná grafová struktura poté prochází analýzou, při které se vyhledávají cesty od počátečního stavu ke koncovým stavům, přičemž se na těchto cestách identifikují cykly. V tomto bodě se aplikace může zacyklit, pokud je vstupní automat typu DPDA a obsahuje v množině pravidel R pravidla, jež cyklicky navyšují zastoupení nevstupních symbolů v zásobníku.

Následně probíhá konstrukce cílového (simulujícího) automatu, přičemž postup tvorby respektuje vztah mezi typem původního (simulovaného) a cílového (simulujícího) modelu. Po dokončení této operace se výsledná interní data převedou zpět do strukturovaného zápisu a aplikace vygeneruje výstupní soubor ve formátu `.json`. Následovat bude i generování grafické reprezentace automatu ve formátu `.png`. Pokud uživatel nezadá jinak, tento celý

proces se automaticky zopakuje i pro nově vytvořený výstupní automat, jehož typ a třída přijímaných jazyků odpovídají původnímu vstupnímu automatu.

Jazyk `Python` byl zvolen především pro svou přehlednou syntaxi, širokou dostupnost knihoven určených k práci s datovými strukturami, snadné zpracování formátu `JSON` a existenci nástrojů pro generování grafických výstupů (v rámci této práce použita knihovna `pydot`[15]). Tato kombinace vlastností výrazně urychlila vývoj a zároveň umožnila vytvořit dobře čitelný, rozšiřitelný a snadno modifikovatelný kód, který je vhodný pro další výzkumné i praktické použití.

5.2 Vstupní konfigurační soubor

Jak bylo řečeno, aplikace přijímá vstupní konfigurační soubor ve formátu `.json`, který reprezentuje strukturovanou definici automatu. Vzhledem k tomu, že aplikace dokáže provádět oboustrannou transformaci, lze ji spustit se vstupním automatem buď typu DPDA, nebo typu OPDPDA. Kromě správně zvolených parametrů při spouštění konzolové aplikace je proto nezbytné mít korektně sestavený vstupní soubor. Vzhledem k rozdílům v definici modelů DPDAs a OPDPDAs vyžaduje každý z nich v konfiguračním souboru odlišnou strukturu popisu automatu.

Struktura pro definici vstupního modelu DPDA v konfiguračním souboru je následující:

```
{
  "set-of-states": [],
  "input-alphabet": [],
  "pushdown-alphabet": [],
  "rules": [],
  "start-state": "počáteční-stav-automatu",
  "start-pushdown-symbol": "počáteční-symbol-zásobníku",
  "set-of-end-states": []
}
```

Výpis 5.1: Struktura konfiguračního souboru pro DPDA

Struktura 5.1 reprezentuje strukturu konfiguračního souboru, kde:

- **set-of-states** je konečná množina vnitřních stavů
- **input-alphabet** je vstupní abeceda
- **pushdown-alphabet** je zásobníková abeceda
- **rules** je konečná množina pravidel
- **start-state** je počáteční stav
- **start-pushdown-symbol** je počáteční symbol zásobníku
- **set-of-end-states** je konečná množina koncových vnitřních stavů

přičemž

$$\text{input-alphabet} \cup \{\#\} \subseteq \text{pushdown-alphabet} \quad \text{a} \quad \text{set-of-end-states} \subseteq \text{set-of-states}$$

V části konfigurace `rules` jsou uvedena jednotlivá pravidla, která se zapisují následovně:

```
"rules": [
    {
        "from_state": "stav-z-množiny-stavů",
        "to_state": "stav-z-množiny-stavů",
        "index": <integer> >= 1,
        "non-terminal": "nevstupní-symbol-zásobníkové-abecedy",
        "change-to": "řetězec-symbolů"
    },...
]
```

Výpis 5.2: Struktura konfiguračních pravidel modelu DPDA

Struktura 5.2 reprezentuje strukturu pravidel automatu, kde

- **from-state** je vnitřní stav ze kterého bude při použití pravidla proveden skok
- **to-state** je vnitřní stav do kterého bude při použití pravidla proveden skok
- **index** je index přepisovaného nevstupního symbolu
- **non-terminal** je přepisovaný nevstupní symbol
- **change-to** je posloupnost symbolů na kterou bude nevstupní symbol přepsán

Struktura pro definici modelů OPDPDAs v konfiguračním souboru je následující:

```
{
  "set-of-states": [],
  "input-alphabet": [],
  "rules": [],
  "start-state": "počáteční-stav-automatu",
  "bottom-symbol": "symbol-dna",
  "initial-symbol": "počáteční-symbol-zásobníku",
  "end-states": []
}
```

Výpis 5.3: Struktura konfiguračního souboru pro OPDPDA

Struktura 5.3 reprezentuje strukturu konfiguračního souboru, kde:

- **set-of-states** je konečná množina vnitřních stavů
- **input-alphabet** je vstupní abeceda
- **rules** je konečná množina pravidel
- **start-state** je počáteční stav
- **bottom-symbol** je symbol označující dna zásobníků
- **initial-symbol** je počáteční vstupní symbol všech zásobníků
- **end-states** je konečná množina koncových vnitřních stavů

přičemž,

$$\text{bottom-symbol} \notin \text{input-alphabet}, \text{end-states} \subseteq \text{set-of-states a}$$

$$\text{initial-symbol} \in \text{input-alphabet}$$

V části konfigurace `rules` jsou uvedena jednotlivá pravidla, která se zapisují dvěma způsoby:

```
"rules": [
    {
        "from_state": "stav-z-množiny-stavů",
        "to_state": "stav-z-množiny-stavů",
        "index": <integer> >= 1,
        "pd_index": <integer> >= 1,
        "symbol": "přepisovaný-vstupní-symbol",
        "change": "řetězec-symbolů"
    },
    {
        "from_state": "stav-z-množiny-stavů",
        "to_state": "stav-z-množiny-stavů",
        "type": "EPSILON"
    },
    {
        "from_state": "stav-z-množiny-stavů",
        "to_state": "stav-z-množiny-stavů",
        "pd_index": <integer> >= 1,
        "type": "COPY",
        "change": "řetězec-symbolů"
    },
    {
        "from_state": "stav-z-množiny-stavů",
        "to_state": "stav-z-množiny-stavů",
        "type": "SET",
        "change": "konfigurace-zásobníků"
    }...
]
```

Výpis 5.4: Struktura konfiguračních pravidel modelu OPDPDA

Struktura 5.4 reprezentuje strukturu pravidel automatu, kde

- **from-state** je vnitřní stav ze kterého bude při použití pravidla proveden skok
- **to-state** je vnitřní stav do kterého bude při použití pravidla proveden skok
- **index** je index přepisovaného vstupního symbolu
- **pd_index** je index zásobníku, ve kterém bude provedena změna
- **symbol** je přepisovaný vstupní symbol
- **change** je posloupnost symbolů na kterou bude vstupní symbol rozepsán
- **type** udává typ abstraktního pravidla (každé se chová jinak a vyžaduje jiné parametry)

5.3 Struktura a moduly aplikace

Aplikace je distribuována jako Python knihovna v balíčku `dpda_transformer`, který je rozdělen na tři hlavní podbalíčky s jasně definovanou odpovědností:

- **core.py** Slouží jako vstupní bod celé aplikace. Zpracovává CLI parametry, volá funkce pro načítání a ukládání JSON konfigurací, generuje grafické výstupy (`.png`) a řídí proces analýzy vstupního simulovaného automatu a tvorby nového simulujícího modelu.
- **utils/** Obsahuje doplňkové nástroje:
  - `arg_parse.py` – parsování a validace přepínačů a argumentů příkazové řádky,
  - `generate_graph_image.py` – funkce pro kreslení DPDA a OPDPDA pomocí externí knihovny `pydot`,
  - `string_encoder.py` – převod libovolných řetězců na unikátní pseudo znaky,
  - `list_utils.py`, `string_utils.py` – pomocné operace nad seznamy a řetězci.
- **parser/** Zajišťuje konverzi mezi Python datovými strukturami a JSON formátem:
  - `json_load.py` – načtení a validace JSONu do slovníkové struktury `dict`,
  - `json_save.py` – zpětná serializace struktury `dict` do JSON souboru,
  - `json_create.py` – generování strukturované slovníkové podoby automatu z interních dat (pro oba modely jsou postupy rozdílné).
- **graph/** Hlavní jádro aplikace: moduly reprezentující stavové grafy a algoritmy nad nimi.
  - `module.py` – definice tříd `Graph` (DPDA), `PureGraph` (OPDPDA) a `Dfs` pro hledání cest a cyklů pomocí Dfs algoritmu,
  - `dpda_algo.py` – transformace DPDA → OPDPDA (zahrnuje hledání cest, silně souvislých komponent, analýzu chování a samotnou tvorbu cílového modelu),
  - `opdpda_algo.py` – transformace OPDPDA → DPDA (zahrnuje hledání cest, silně souvislých komponent, analýzu chování a samotnou tvorbu cílového modelu),
  - `create_module.py` – konstrukce objektu `Dfs` pro průzkum grafů a struktur `Graph` a `PureGraph` nad modely DPDA a OPDPDA

Dodržení programátorských norem a standardů

Vývoj balíčku `dpda_transformer` probíhal v souladu s běžně uznávanými programátorskými konvencemi pro jazyk Python:

- **PEP 8 – Style Guide for Python Code [14]:** Zdrojový kód respektuje pravidla formátování podle standardu PEP 8, zejména co se týče odsazování, pojmenovávání proměnných a strukturace kódu.
- **PEP 257 – Docstring Conventions (lokalizovaná forma) [2]:** Veřejné funkce a metody jsou opatřeny vícero-řádkovými dokumentačními komentáři ve formě docstringů. Tyto komentáře jsou psány v českém jazyce a obsahují popis vstupních argumentů, návratových hodnot a případně výjimek, které mohou být vyvolány. Tím je zajištěna přehledná a systematická dokumentace jednotlivých částí kódu.

- **Typové anotace:** Kód využívá typové anotace podle specifikace `typing`, což umožňuje snadnější statickou analýzu a zvyšuje srozumitelnost funkcí a jejich rozhraní.

Přehled souborové struktury

```
dpda_transformer/
 core.py
 utils/
    arg_parse.py
    generate_graph_image.py
    list_utils.py
    string_encoder.py
    string_utils.py
 parser/
    json_create.py
    json_load.py
    json_save.py
 graph/
     module.py
     create_module.py
     dpda_algo.py
     opdpda_algo.py
```

5.4 Vstupní parametry a spuštění

Aplikace je navržena pro běh na operačních systémech typu Unix (např. Linux) a vyžaduje instalaci Pythonu ve verzi 3.6 nebo novější. Celý projekt je implementován jako knihovna s názvem `dpda_transformer`, přičemž spouštění aplikace probíhá prostřednictvím modulu s hlavním řídicím skriptem.

Pro spuštění aplikace lze využít příkaz:

```
python3 -m dpda_transformer
```

nebo v případě přímého spuštění skriptu:

```
python3 dpda_transformer/core.py
```

Aplikace však ke správnému fungování vyžaduje zadání povinných argumentů příkazové řádky. Tyto argumenty umožňují specifikovat vstupní soubor a typ analyzovaného zásobníkového automatu:

- `-c`, `--conf` – určuje cestu ke vstupnímu souboru ve formátu JSON, který obsahuje popis konfiguračních dat vstupního automatu.
- `-d`, `--dpda` – přepínač informující, že vstupní automat je typu DPDA.
- `-p`, `--opdpda` – přepínač informující, že vstupní automat je typu OPDPDA.

Přepínače `-d` a `-p` se vzájemně vylučují a je tedy nutné zvolit právě jeden z nich, jinak nebude aplikace spuštěna. Ukázkový příkaz pro spuštění aplikace může vypadat například takto:

```
python3 -m dpda_transformer -c ./data/dpda_test.json -d
```

Tímto způsobem dojde k načtení konfiguračního souboru `dpda_test.json` a zahájení analýzy a transformace zadaného DPDA automatu.

Po úspěšném spuštění aplikace jsou vygenerovány výstupní automaty ve formátu konfiguračních `.json` souborů, které jsou uloženy do stejného adresáře, kde se nachází vstupní soubor. Současně jsou vytvořeny také grafické reprezentace těchto automatů ve formátu `.png`.

5.5 Výsledné hodnoty a testování

Pro aplikaci byla připravena sada konfiguračních vstupů reprezentujících automaty typu DPDA. Testovací soubory lze najít v adresáři `\tests` spolu s vygenerovanými automaty ve formátu JSON a PNG. Program bez problémů transformuje všechny korektně sestavené automaty.

Implementace však neobsahuje ošetření pro „absurdně“ nekorektní konfigurace automatů. Předpokládáme, že vstupní soubory odpovídají přehlednému a konzistentnímu zápisu (např. `test1.json`, `test2.json` apod.). Jak již bylo zmíněno v předchozí kapitole, tato práce se zaměřuje pouze na modely, které své přechody odvozují kombinací aktuálního stavu a stavu zásobníků.

Pokud je automat správně definován a přepsán do požadovaného formátu, aplikace bez potíží provádí obousměrné transformace i analýzy a na závěr vypisuje výsledná data. V případě, že se zpracovává vstupní model DPDA obsahující pravidla, která cyklicky rozšiřují zastoupení nevstupních symbolů v zásobníku, program se zacyklí, což je očekávané chování.

Test 1

Mějme vstupní konfigurační soubor `test1.json`, reprezentující hluboký zásobníkový automat M_{nDPDA}, který přijímá jazyk $L_{nDPDA} = \{a^i b^j c^k | i = j + k, j \leq k, i \geq 1\}$, generovaný stavovou gramatikou stupně 2. Po spuštění programu nám program vygeneruje automaty, přijímající ekvivalentní třídu jazyků A.1.

Test 2

Mějme vstupní konfigurační soubor `test2.json`, reprezentující hluboký zásobníkový automat M_{nDPDA}, který přijímá jazyk $L_{nDPDA} = \{a^i b^i c^i d^i | i \geq 1\}$, generovaný stavovou gramatikou stupně 2. Po spuštění programu nám program vygeneruje automaty, přijímající ekvivalentní třídu jazyků A.2.

Test 3

Mějme vstupní konfigurační soubor `test17.json`, reprezentující hluboký zásobníkový automat M_{nDPDA}, který přijímá jazyk $L_{nDPDA} = \{a^m c^v g^z h^z i^z j^z d^v e^v k^y t^y f^v b^m | m, v, z, y, v \geq 0\}$, generovaný stavovou gramatikou stupně 2. Po spuštění programu nám program vygeneruje automaty, přijímající ekvivalentní třídu jazyků A.3.

Test 4

Mějme vstupní konfigurační soubor `test24.json`, reprezentující hluboký zásobníkový automat M_{nDPDA}, který přijímá jazyk $L_{nDPDA} = \{\omega\omega|\omega \in \{a, b, c\}^*\}$, generovaný stavovou gramatikou stupně 2. Po spuštění programu nám program vygeneruje automaty, přijímající ekvivalentní třídu jazyků A.4.

Test 5

Mějme vstupní konfigurační soubor `test25.json`, reprezentující hluboký zásobníkový automat M_{nDPDA}, který přijímá jazyk $L_{nDPDA} = \{a^i b^i c^i d^i e^i f^i|i \geq 1\}$, generovaný stavovou gramatikou stupně 3. Po spuštění programu nám program vygeneruje automaty, přijímající ekvivalentní třídu jazyků A.5.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a zavést nový typ alternativního modelu vycházejícího z hlubokých zásobníkových automatů (DPDAs), prozkoumat jeho chování, vlastnosti, rozpoznávací sílu a tyto netriviální vlastnosti formálně dokázat a experimentálně ověřit pomocí speciálně navrženého a vyvinutého nástroje.

V úvodní části této práce byly nejprve představeny klíčové teoretické koncepty formálních jazyků, gramatik a automatů, následované složitějšími DPDAs, včetně stavových gramatik (SG), u nichž byly zkoumány jejich vlastnosti, chování a generativní či rozpoznávací síly.

Hlavním přínosem této práce je zavedení čistého hlubokého seřazeného n-zásobníkového automatu OPDPDA, který na rozdíl od klasických modelů DPDAs nevyužívá nevstupní symboly uvnitř n zásobníků, ale pracuje výhradně se symboly ze vstupní abecedy. Implicitní pravidla `pops` se aplikují pouze po dosažení koncových stavů, a díky variabilnímu počtu zásobníků, který přímo ovlivňuje třídu rozpoznávaných jazyků, je tato třída jazyků vlastní podmnožinou tříd jazyků přijímaných modely DPDAs. V této práci byla také formálně definována struktura modelů OPDPDAs, jejich konfigurace, typy přechodů a principy jejich použití s doplňujícími příklady.

V práci byly navrženy a v rámci nástroje i implementovány algoritmy pro konstrukci a transformaci automatů mezi modely OPDPDAs a DPDAs. Následující část se poté zabývala definicí přijímací síly modelů OPDPDAs, s formálně definovanými důkazy. Na základě těchto důkazů a výsledků experimentů bylo předvedeno a dokázáno, že třída jazyků rozpoznávaných modely OPDPDAs leží mezi regulárními jazyky (RGs) a jazyky přijímanými DPDAs, více 4.6.1. Důvodem je, že modely OPDPDAs kvůli nemožnosti nabytí nekonečného počtu pravidel a zásobníků nemohou simulovat pravidla modelů DPDAs a ZAs, která cyklicky rozšiřují zastoupení nevstupních symbolů v zásobnících, které mohou být dále samostatně přepisovány do potenciálně nekonečných posloupností.

Navržené algoritmy byly implementovány do podoby konzolové aplikace ve formě `Python` knihovny. Funkce této knihovny lze použít pro transformace mezi modely s následnými konstrukcemi strukturovaných reprezentací v zápisu `.json`, či generováním grafických reprezentací v podobě `.png` grafů. Pro experimentování s nástrojem byla vytvořena sada testovacích vstupů.

Budoucí výzkum

Pro budoucí výzkum se nabízí rozšíření modelů OPDPDAs o mechanismus postupného uvolňování znaků ze zásobníků i bez dosažení koncových stavů, tj. modifikace operace `pops`

a samotné definice modelu způsobem, aby bylo možné postupně vybírat a odstraňovat symboly během chodu automatu. Je zajímavé prozkoumat, zda takto upravený model, který systematičtěji pracuje se svými zásobníky, dokáže přijímat třídu jazyků, kterou rozpoznávají DPDAs s cyklickými pravidly rozšiřující zastoupení nevstupních symbolů v zásobníku, a které OPDPDAs rozpoznat nemohou.

Dále by bylo vhodné zkoumat behaviorální vlastnosti deterministických verzí OPDPDAs, neboť tato práce se zaměřila převážně na jejich nedeterministické verze.

Zajímavé by bylo také prozkoumat chování, vlastnosti a třídy přijímaných jazyků při omezení počtu použitelných zásobníků, během zpracování vstupních řetězců.

Literatura

[1] *The JSON Data Interchange Format* RFC 8259, Internet Engineering Task Force. 2017. Dostupné z: https://www.rfc-editor.org/rfc/rfc8259.html.

[2] Goodger, D. a Rossum, G. van. *PEP 257 – Docstring Conventions* https://peps.python.org/pep-0257/. 2001. Python Enhancement Proposal.

[3] Horáček, P.; Meduna, A. a Tomko, M. *Handbook of Mathematical Models for Languages and Computation*. The Institution of Engineering and Technology, 2020. 750 s. ISBN 978-1-78561-659-4. Dostupné z: https://www.fit.vut.cz/research/publication/12041.

[4] Horvat, G. a Meduna, A. On State Grammars. *Acta Cybernetica*, 1988, sv. 1988, č. 8, s. 237–245. ISSN 0324-721X. Dostupné z: https://www.fit.vut.cz/research/publication/6156.

[5] Hunter, T. *The Chomsky Hierarchy*. Hoboken, NJ: Wiley-Blackwell, 2021. 85–102 s. Dostupné z: https://timhunter.humspace.ucla.edu/papers/blackwell-chomsky-hierarchy.pdf.

[6] Kasai, T. An hierarchy between context-free and context-sensitive languages. *Journal of Computer and System Sciences*, 1970, sv. 4, č. 5, s. 492–508. ISSN 0022-0000. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0022000070800459.

[7] Linz, P. *An Introduction to Formal Languages and Automata, Fifth Edition*. 5th. USA: Jones and Bartlett Publishers, Inc., 2011. ISBN 9781449615529.

[8] Meduna, A. *Automata and Languages: Theory and Applications*. Springer Verlag, 2005. 892 s. ISBN 1-85233-074-0. Dostupné z: https://www.fit.vut.cz/research/publication/6177.

[9] Meduna, A. Deep pushdown automata. *Acta Informatica*, duben 2006, sv. 42, č. 8, s. 541–552. ISSN 1432-0525. Dostupné z: https://doi.org/10.1007/s00236-006-0005-0.

[10] Meduna, A. *Formal Languages and Computation*. Taylor & Francis Informa plc, 2014. 315 s. Taylor and Francis. ISBN 978-1-4665-1345-7. Dostupné z: https://www.fit.vut.cz/research/publication/10524.

[11] Meduna, A. a Zemek, P. *Regulated Grammars and Automata*. Springer US, 2014. 694 s. ISBN 978-1-4939-0368-9. Dostupné z: https://www.fit.vut.cz/research/publication/10498.

[12] Putala, M. *Částečně paralelní hluboké zásobníkové převodníky a jejich aplikace.* 2023. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů. Dostupné z: http://hdl.handle.net/11012/211031. [cit. 2025-01-02].

[13] Rossum, G. van a Python Development Team the. *Python Language Reference, version 3.10.4.* Python Software Foundation, 2022. Dostupné z: https://docs.python.org/3/.

[14] Rossum, G. van; Warsaw, B. a Coghlan, N. *PEP 8 – Style Guide for Python Code* https://peps.python.org/pep-0008/. 2001. Python Enhancement Proposal.

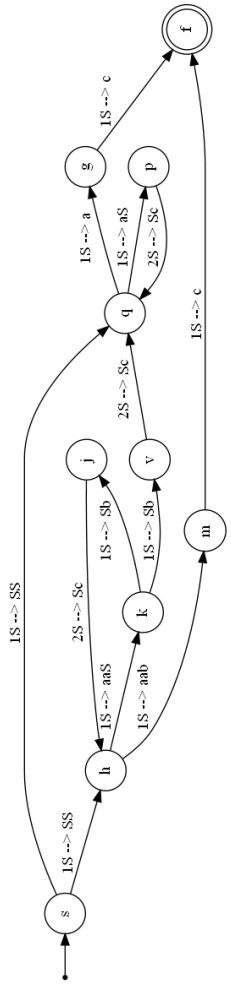
[15] Wilber, A. a Contributors. *Pydot: Python interface to Graphviz's Dot language.* 2017. Dostupné z: https://github.com/pydot/pydot.

[16] Češka, M.; Vojnar, T.; Smrčka, A. a Rogalewicz, A. *Teoretická informatika.* FIT VUT v Brně, aug 2020. Dostupné z: https://www.fit.vut.cz/study/course/TIN/public/Texty/TIN-studijni-text.pdf. Studijní text, TIN.

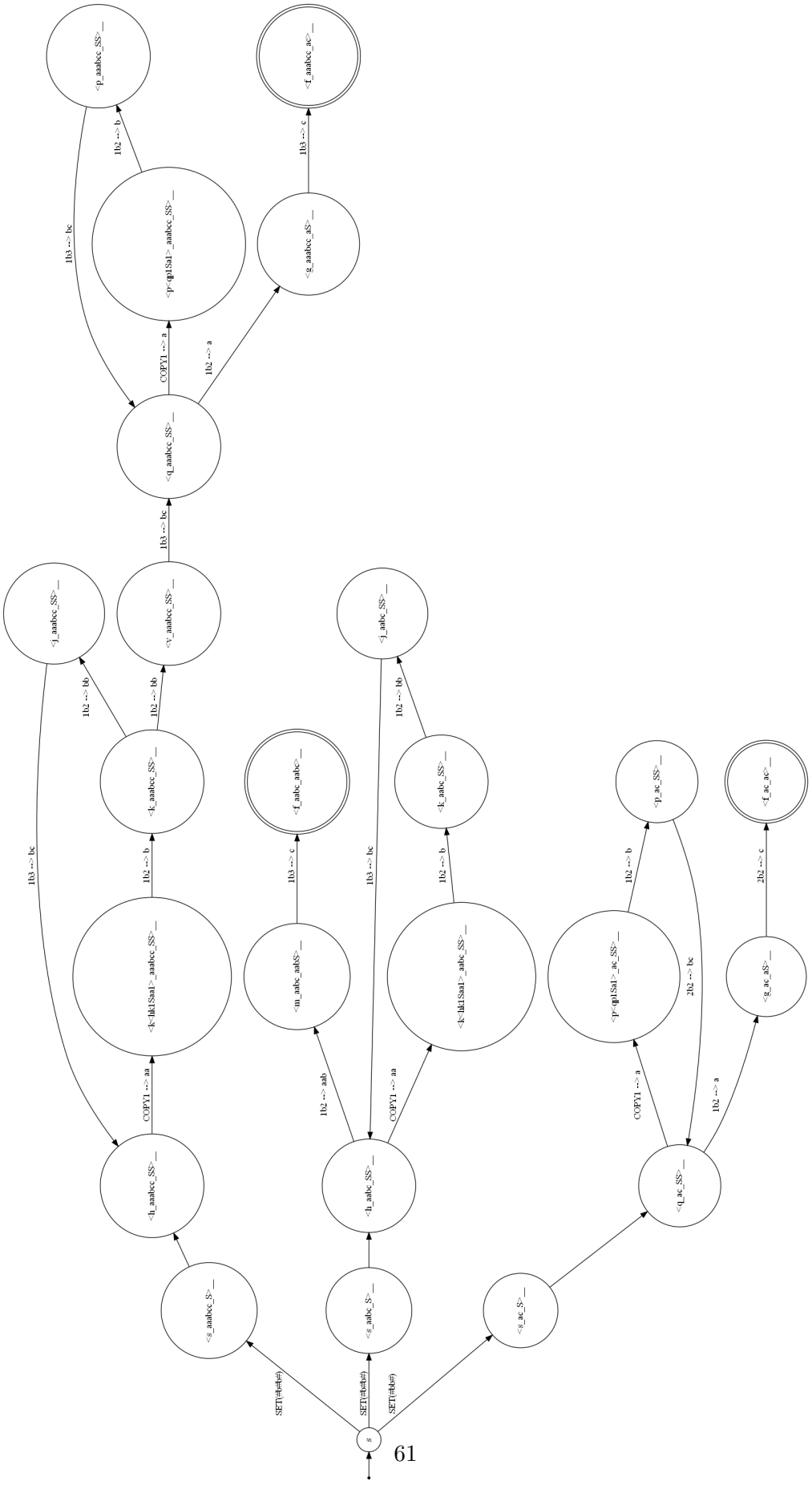
Příloha A

Testovací příklady

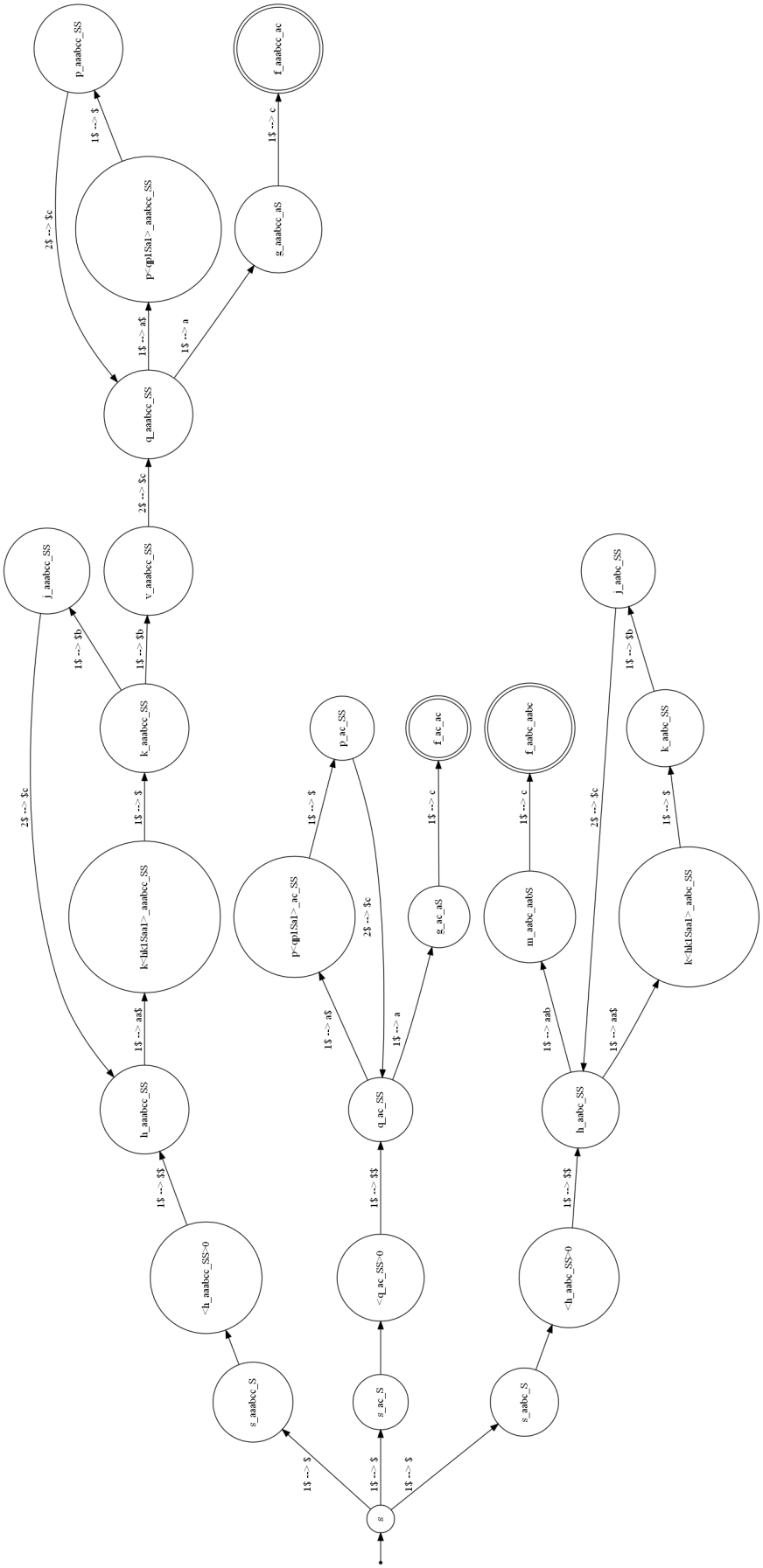
A.1 Příklad 1



Obrázek A.1: Grafická reprezentace DPDA modelu /test1.png

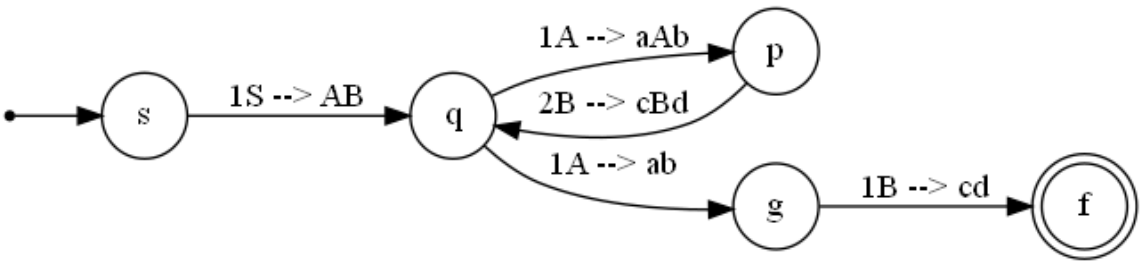


Obrázek A.2: Grafická reprezentace OPDPDA modelu /test1_toOPDPDA.png

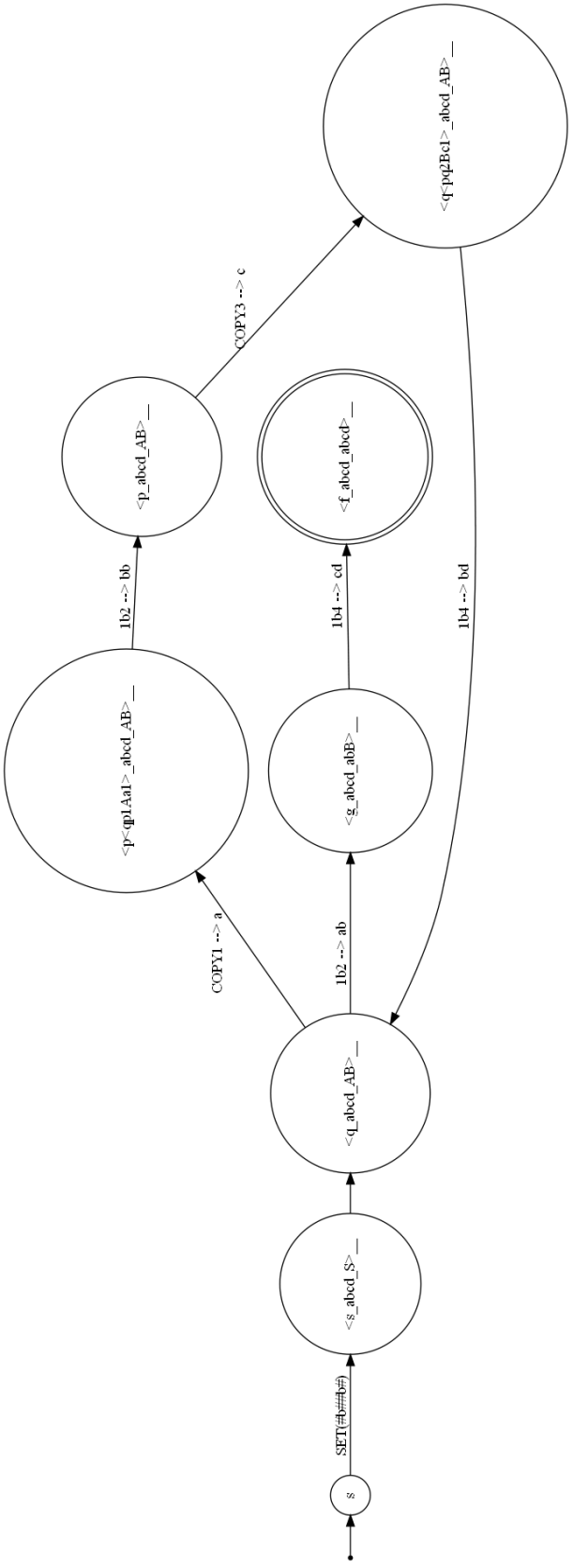


Obrázek A.3: Grafická reprezentace DPDA modelu /test1_toOPDPDA_toDPDA.png

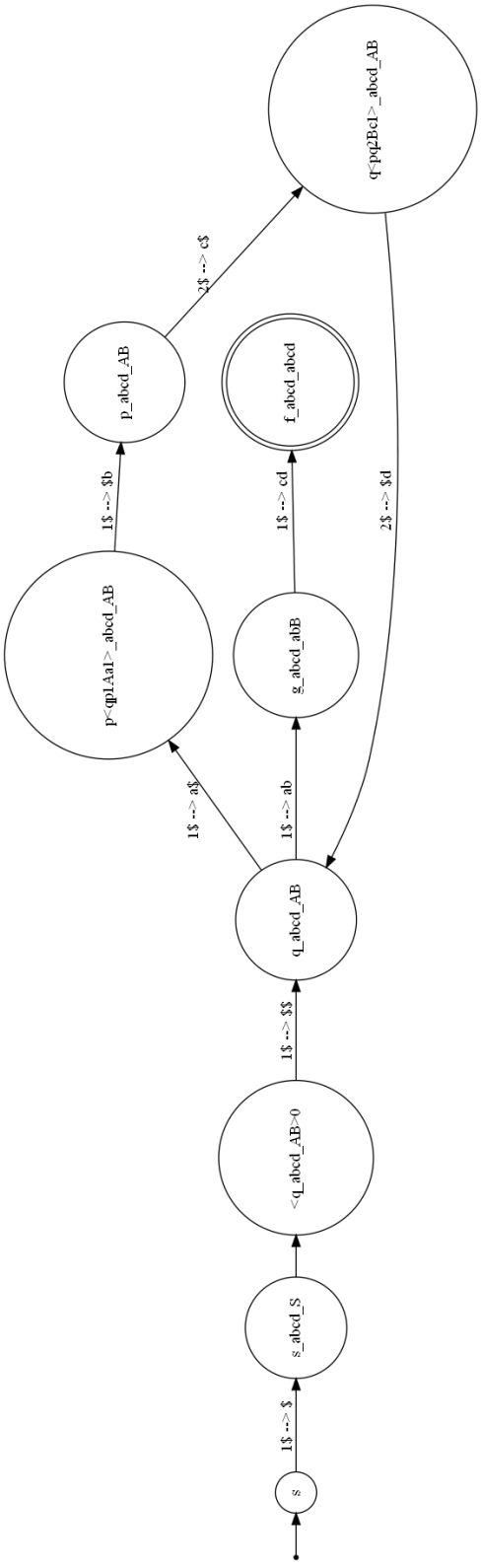
A.2 Příklad 2



Obrázek A.4: Grafická reprezentace DPDA modelu /test2.png

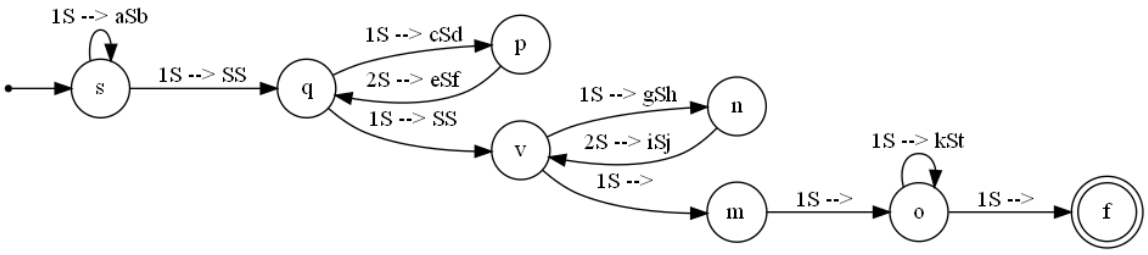


Obrázek A.5: Grafická reprezentace OPDPDA modelu /test2_toOPDPDA.png

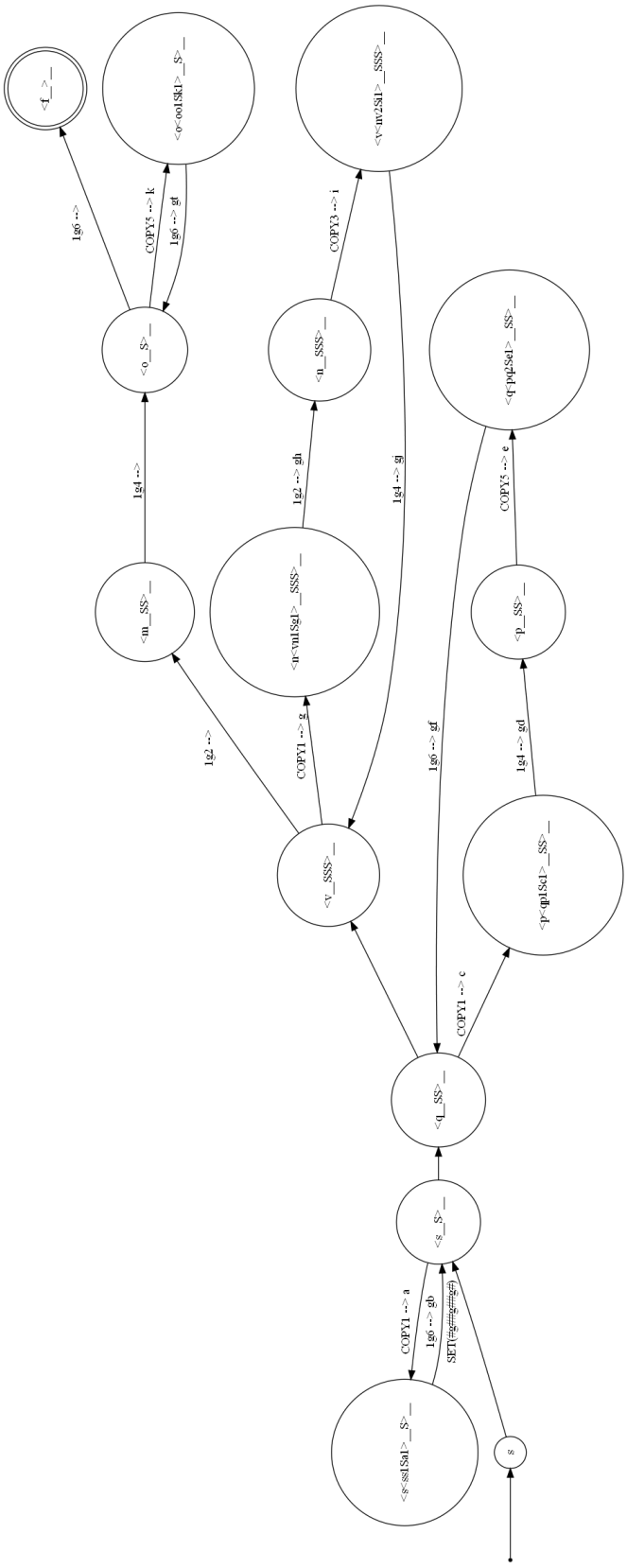


Obrázek A.6: Grafická reprezentace DPDA modelu /test2_toOPDPDA_toDPDA.png

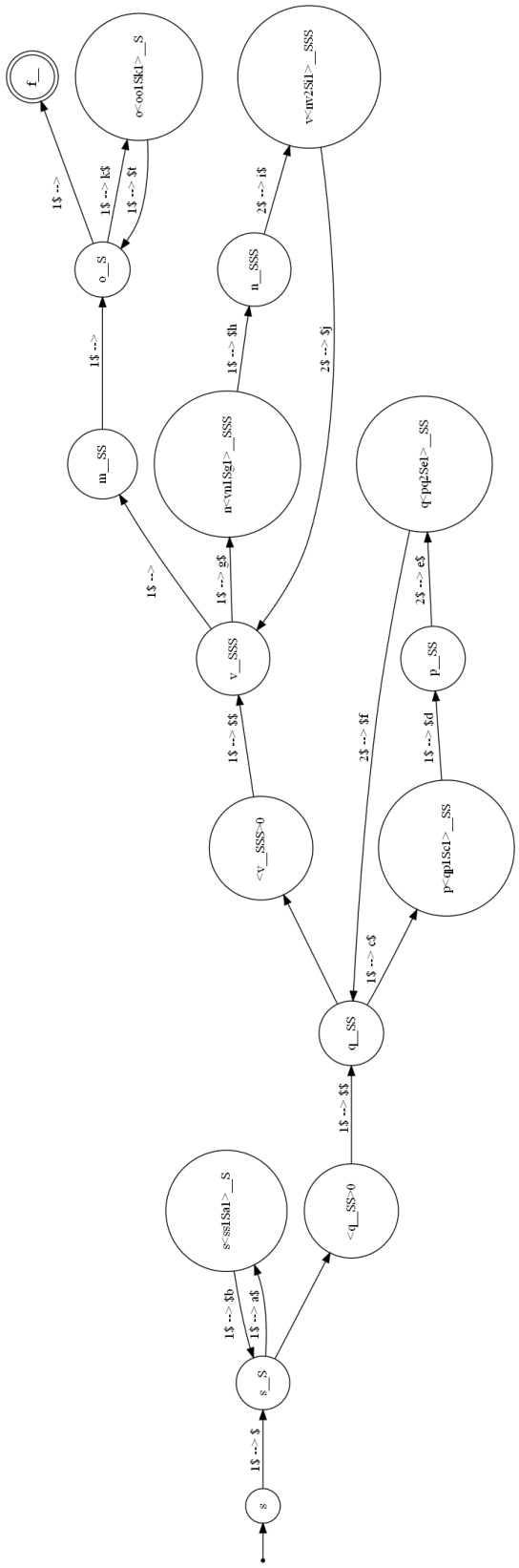
A.3 Příklad 3



Obrázek A.7: Grafická reprezentace DPDA modelu /test17.png

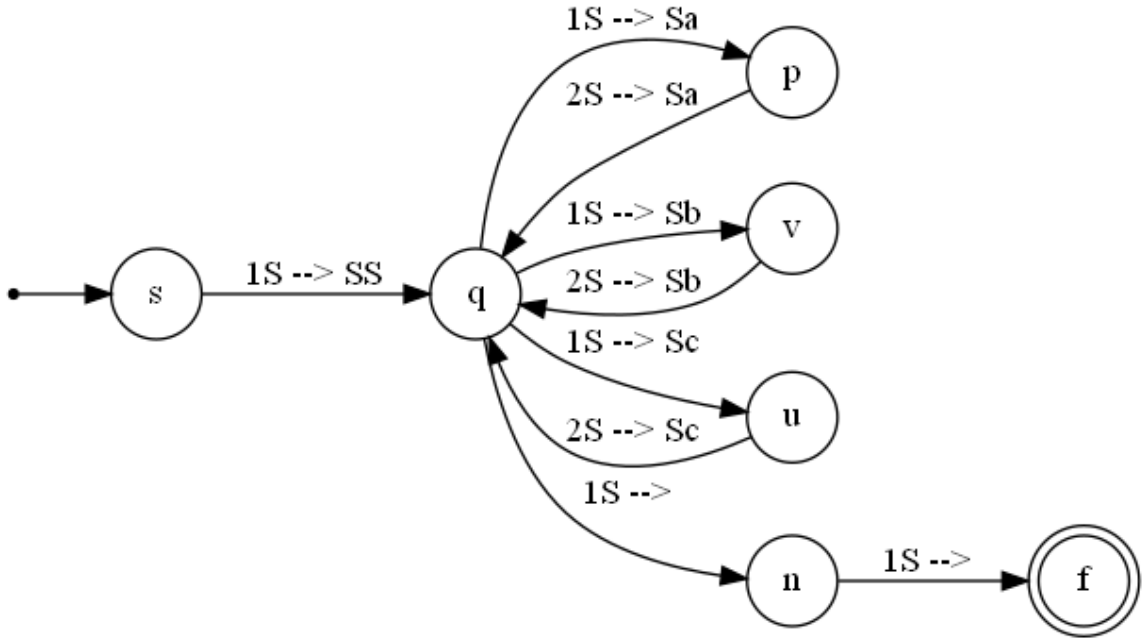


Obrázek A.8: Grafická reprezentace OPDPDA modelu /test17_toOPDPDA.png

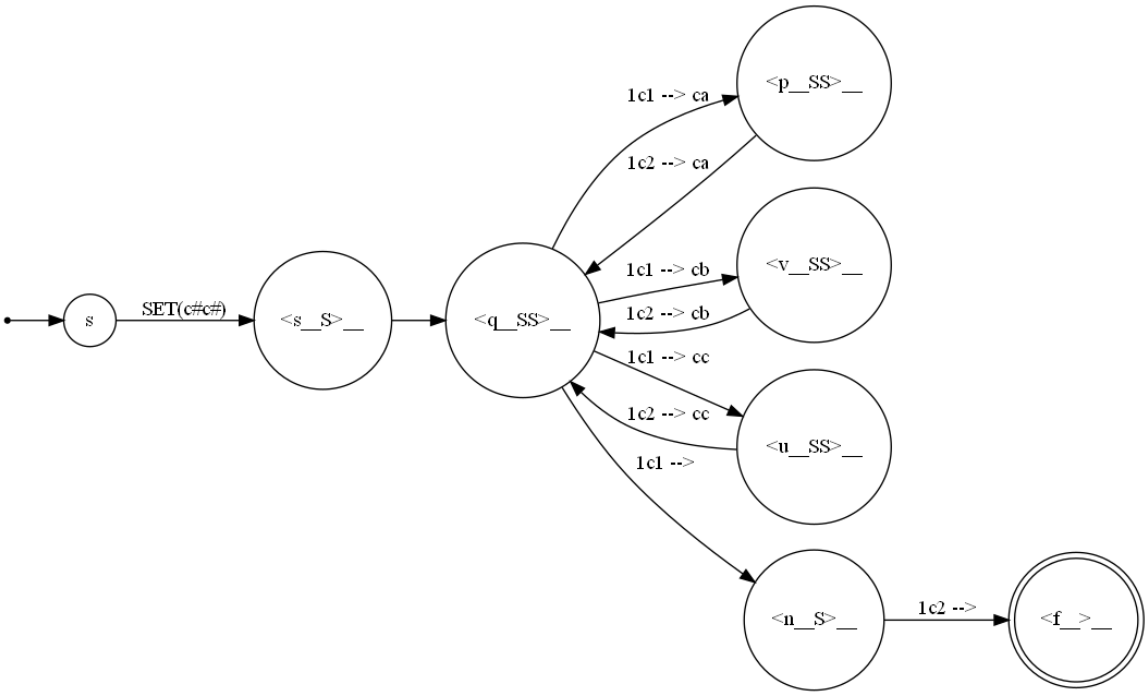


Obrázek A.9: Grafická reprezentace DPDA modelu /test17_toOPDPDA_toDPDA.png

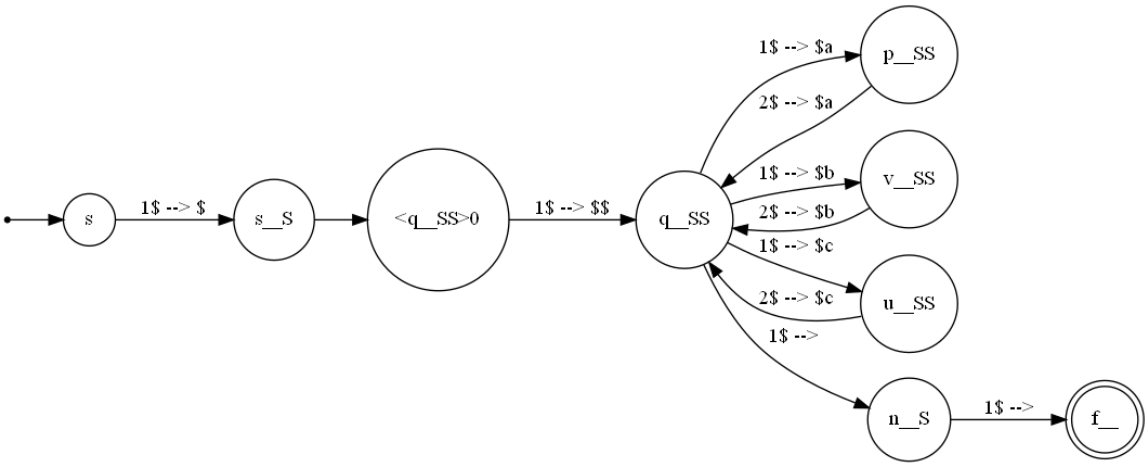
A.4 Příklad 4



Obrázek A.10: Grafická reprezentace DPDA modelu /test24.png

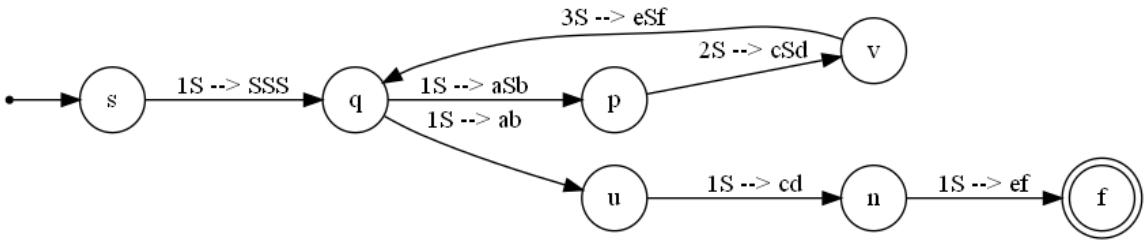


Obrázek A.11: Grafická reprezentace OPDPDA modelu /test24_toOPDPDA.png

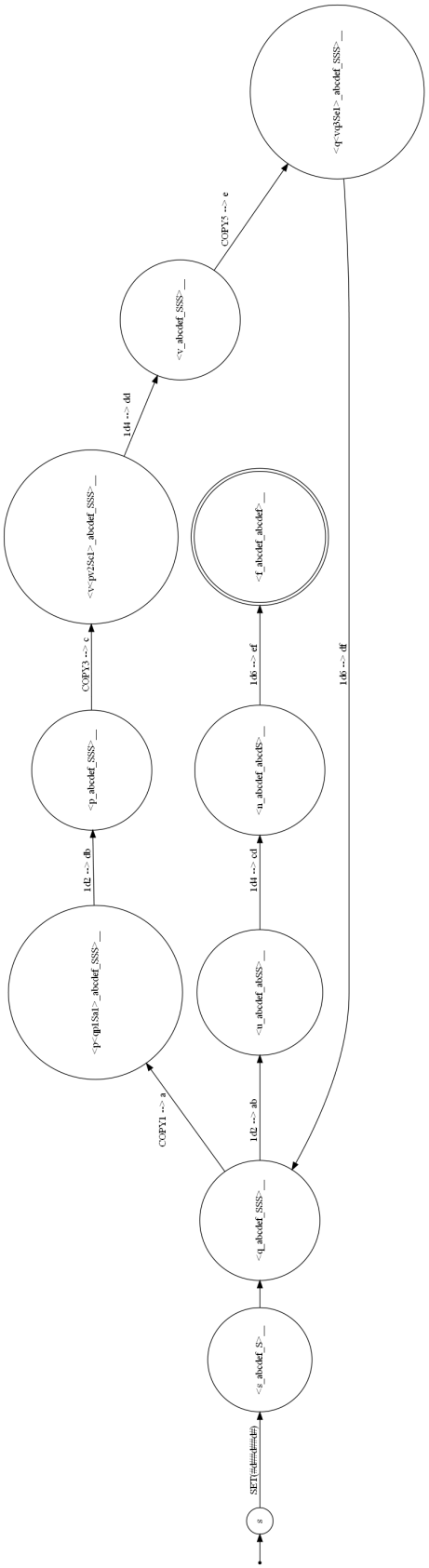


Obrázek A.12: Grafická reprezentace DPDA modelu /test24_toOPDPDA_toDPDA.png

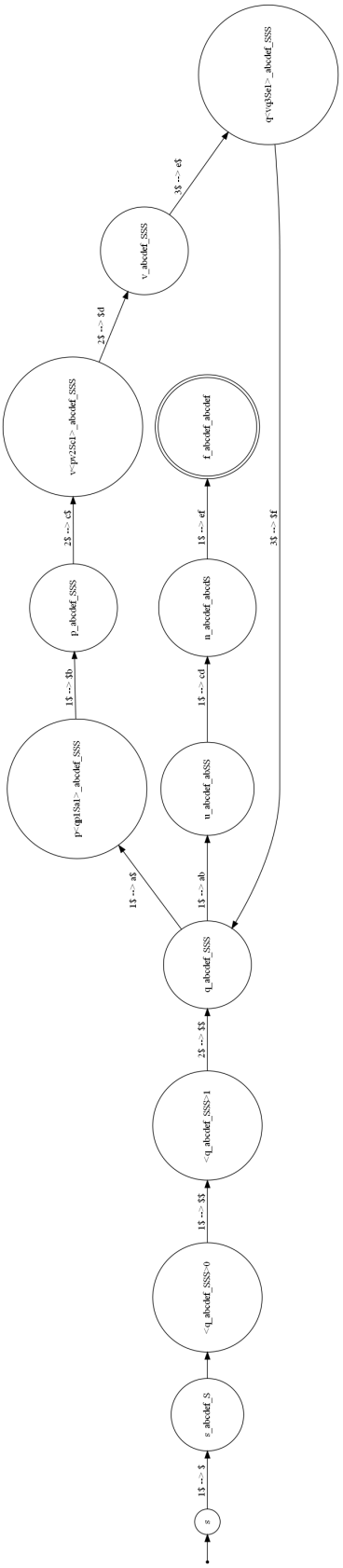
A.5 Příklad 5



Obrázek A.13: Grafická reprezentace DPDA modelu /test25.png



Obrázek A.14: Grafická reprezentace OPDPDA modelu /test25_toOPDPDA.png



Obrázek A.15: Grafická reprezentace DPDA modelu /test25_toOPDPDA_toDPDA.png