VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

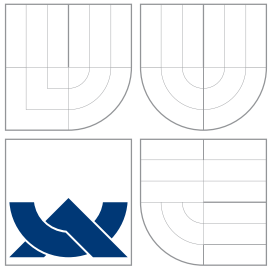
PŘEPISUJÍCÍ SYSTÉMY S OMEZENÝMI KONFIGURACEMI

DIZERTAČNÍ PRÁCE
PHD THESIS

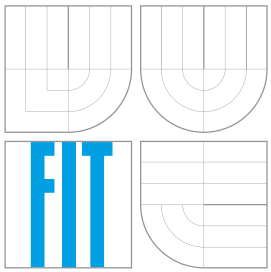
AUTOR PRÁCE
AUTHOR

Ing. ZBYNĚK KŘIVKA

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘEPISUJÍCÍ SYSTÉMY S OMEZENÝMI KONFIGURACEMI

REWRITING SYTEMS WITH RESTRICTED CONFIGURATIONS

DIZERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE — Ing. ZBYNĚK KŘIVKA
AUTHOR

VEDOUCÍ PRÁCE — Prof. RNDr. ALEXANDER MEDUNA, CSc.
SUPERVISOR

BRNO 2007

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Zbyněk Křivka

Bytem: Kšírova 193, Brno 619 00

Narozen/a (datum a místo): 3. června 1981 v Brně

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií

se sídlem Božetěchova 2, 612 66 Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - ☑ disertační práce
   - ☐ diplomová práce
   - ☐ bakalářská práce
   - ☐ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: Přepisující systémy s omezenými konfiguracemi

Vedoucí/ školitel VŠKP: Prof. RNDr. Alexander Meduna, CSc.

Ústav: Ústav informačních systémů

Datum obhajoby VŠKP:

VŠKP odevzdal autor nabyvateli v*:

- ☑ tištěné formě – počet exemplářů: 3
- ☑ elektronické formě – počet exemplářů: 2

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☑ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy
   - ☐ (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 15. června 2007

…….. ……
Nabyvatel Autor

Abstrakt

Tato teoretická dizertační práce zastřešuje pod pojmem přepisující systémy formální modely gramatik a automatů a zkoumá možnosti jejich kombinování. Zobecňuje pojem konfigurace pro označení vnitřního stavu přepisujících systémů a sjednocuje i některé další pojmy z oblasti gramatik a automatů. Dále klasifikuje různé přístupy k omezování přepisujících systémů s důrazem na omezování konfigurací a posloupností aplikovaných pravidel. Práce též představuje pojem dynamická složitost, který se zaměřuje na omezování vybraných metrik při procesu zpracovávání věty přepisujícím systémem.

Hlavní část práce představuje dva nové kombinované modely (#-přepisující systémy a redukující hluboké zásobníkové automaty) a jeden omezovaný systém (zásobníkové automaty s omezeným obsahem zásobníku). V případě #-přepisujících systémů bylo navíc detailněji studováno několik modifikací (n-pravě-lineární a zobecněné #-přepisující systémy inspirované Chomského hierarchií).

V závislosti na způsobu omezování prezentuje text výsledky ohledně vyjadřovacích schopností těchto formálních modelů. Demonstruje také úzký vztah nových i vybraných existujících kombinovaných, omezovaných a řízených přepisujících systémů. Hlavním výsledkem jsou nekonečné hierarchie tříd jazyků definované těmito přepisujícími systémy podle parametrů omezování (konečný index, n-limitovanost, hloubka).

V závěru jsou studovány vlastnosti těchto systémů s úzkou vazbou na praxi. Text zavádí dva z možných způsobů definice determinismu a kanoničnosti #-přepisujících systémů a demonstruje jejich vztah k potencionální praktické využitelnosti.

Klíčová slova

#-přepisující systém, determinismus, dynamická složitost, formální model, jazyk, konečný index, konfigurace, n-limitovanost, m-paralelní n-pravě-lineární jednoduchá maticová gramatika, omezování, popisná složitost, programovaná gramatika, redukující hluboký zásobníkový automat, stavová gramatika, řízený přepisující systém, vyjadřovací schopnost, třída jazyků, zásobníkový automat s omezeným obsahem zásobníku

Abstract

This theoretically oriented dissertation discusses rewriting systems, including various automata and grammars. It concentrates its attention upon their combination. More specifically, the central role of the present dissertation plays the general notion of a configuration as an instantaneous description of a rewriting system. Based upon various restrictions placed upon configurations and rewriting modes, the systems are classified and studied. Apart from this major topic, the dissertation also discusses dynamic complexity, which is based upon metrics placed upon the process of yielding strings.

As its fundamental topic, the dissertation discusses #-rewriting system, reducing deep pushdown automaton, and pushdown automata with restricted pushdowns. In addition, it studies some variants of #-rewriting systems, including n-right linear and generalized #-rewriting system. In general, the dissertation demonstrates how the generative power of the systems under discussion depends upon the restrictions placed upon them. In terms of dynamic complexity, it discusses a close relationship between various rewriting systems, including newly introduced systems. As its main result, the dissertation demonstrates several infinite hierarchies of language families defined by rewriting systems in dependency on their restrictions.

The conclusion demonstrates the application-important properties of the systems discussed in the dissertation. It sketches two new types of determinism and canonical rewriting; then, it demonstrates their potential practical usage.

Keywords

#-rewriting system, configuration, descriptional complexity, determinism, dynamic complexity, finite index, formal model, generative power, language, language family, n-limitation, m-parallel n-right-linear simple matrix grammar, programmed grammar, pushdown automaton with restricted pushdown content, reducing deep pushdown automaton, regulated rewriting system, restriction, state grammar

Citace

Zbyněk Křivka: Přepisující systémy s omezenými konfiguracemi, dizertační práce, Brno, FIT VUT v Brně, 2007

Přepisující systémy s omezenými konfiguracemi

Prohlášení

Prohlašuji, že jsem tuto dizertační práci vypracoval samostatně pod vedením prof. RNDr. Alexandera Meduny, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal. Pro ucelenost a systematičnost výkladu jsem uvedl i některé výsledky jiných autorů, zejména mého školitele. Konkrétně práce [35], [48], [50] a [53] byly publikovány mým školitelem. Dále byly ocitovány a náležitě označeny již známé výsledky z [12], [21] a [77]. Všechny hodnotné uvedené výsledky byly již dříve samostatně nebo spoluautorsky publikovány, což se odráží v citacích u jednotlivých výsledků.

Aktuální verze textu je navíc doplněna o opravy pravopisných chyb a technických překlepů, které nalezli oponenti dizertační práce.

.....
Zbyněk Křivka
9. října 2007

Poděkování

Ze srdce děkuji svému vedoucímu, profesoru Alexanderu Medunovi, za jeho ochotu, otevřenost, cenné rady a přátelský přístup, s nímž mě provázel mým doktorským studiem a těším se na pokračování naší spolupráce. Dále bych rád poděkoval docentu Petru Sosíkovi za jeho podrobnou a inspirativní oponenturu této práce.

Obsah

Část I

Současný stav

Kapitola 1

Úvod

Jednou z výrazně matematicky orientovaných oblastí informatiky je teorie formálních jazyků. Důležitost tohoto oboru podtrhuje skutečnost, že každý problém nebo úloha, která lze popsat, lze popsat pomocí nějakého jazyka. Základní princip tohoto popisu využívá fakt, že každé řešení lze zapsat formou věty nějakého jazyka. Díky tomu lze způsob řešení všech problémů unifikovat na otázku, zda věta reprezentující hledané řešení patří do jazyka, který popisuje všechna řešení daného problému. Z tohoto pohledu lze přistupovat k hledání řešení dvěma způsoby:

a) ověřujeme větu reprezentující řešení problému, zda patří do jazyka obsahujícího všechna řešení (procedurální přístup);

b) popisujeme/hledáme všechna řešení daného problému (deklarativní přístup).

Teorie formálních jazyků se zabývá hledáním způsobů, jak řešit problémy popsané formálně definovaným jazykem, a proto je pro nás přínosné naše problémy formálními jazyky specifikovat, neboť vlastnosti i způsoby automatického zpracování zjištěné pro příslušnou třídu jazyků, ve které se nachází i jazyk našeho problému, lze bezpracně převzít a využít. Mezi složitostí problému, složitostí popisujícího jazyka a složitostí jeho zpracování je bohužel přímá úměra. Například popis přirozeným jazykem s komplikovanou až nejednoznačnou sémantikou bude vždy hůře strojově zpracovatelný než například jednoduchý regulární výraz popisující identifikátor proměnné.

Jazyk, nositel informace či přímo možných řešení, je postaven na jednom z nejzákladnějších matematických pojmů — množině. Jelikož netriviální jazyk bývá často nekonečnou množinou, musíme mít k dispozici formální modely pro jeho konečný popis.

Formální modely slouží pro přesný matematický popis jazyků a lze je veskrze rozdělit do čtyř základních kategorií:

- množinový výčet, který je sice přesný, ale značně nepraktický
- množinový konstrukt využívající seznamu matematicky nebo slovně zapsaných požadavků a podmínek na prvky množiny, tedy věty jazyka, např. $\{w|\ w$ má délku dělitelnou 3, kde w je libovolná věta nad danou abecedou symbolů$\}$;
- využití algebraických operací pro specifikaci jazyka na základě jiných, již definovaných jazyků, např. $L = L_1 \cup L_2$, kde L_1, L_2 jsou jazyky;
- přepisující systémy, kdy definujeme algoritmický mechanismus, jak zpracovávat věty jazyka; tento přístup je pro praktické použití nejvýznamnější, a je mu proto v teorii formálních jazyků věnována vysoká pozornost; stejně tomu bude i v této práci.

Přepisující systém pracuje se svým vnitřním stavem prostřednictvím přepisování, což je operace nahrazení podřetězce vnitřního stavu systému jiným řetězcem symbolů. Přepisující systém je definován dvěma základními komponentami: úplnou abecedou a konečnou množinou pravidel. Úplná abeceda obsahuje všechny symboly, které se mohou vyskytnout na vstupu, výstupu nebo ve vnitřním stavu přepisujícího systému. Množina pravidel pak diktuje povolené změny, potažmo přepisy, vnitřního stavu systému.

Nejčastěji využívané a zkoumané přepisující systémy jsou gramatiky a automaty.

Z existujících gramatik si vezměme například významné bezkontextové gramatiky. Bezkontextová gramatika se skládá z úplné abecedy; abecedy terminálů, která pouze specializuje některé symboly úplné abecedy; startujícího symbolu; a konečné množiny přepisujících pravidel. Počínaje startujícím symbolem gramatika ve vnitřním stavu (tzv. větná forma) přepisuje pomocí pravidel jednotlivé neterminální symboly (symboly úplné abecedy, které nejsou terminální). Tímto přepisováním gramatika generuje řetězec, až dospěje do řetězce samých terminálů. Množina terminálních řetězců takto vygenerovaných určuje jazyk generovaný bezkontextovou gramatikou.

Automat naopak pracuje praktičtějším způsobem. Větu jazyka ověřuje postupným čtením a zpracováváním pomocí přechodů ve stavovém řízení samotného automatu. Říkáme, že gramatika větu generuje a automat ji přijímá. Jako příklad zástupce přepisujícího systému mezi automaty uvažujme konečný automat, který obsahuje: konečnou množinu stavů, Q, z nichž jeden stav je definován jako počáteční a některé jako koncové; abecedu vstupních symbolů, T, kde Q sjednoceno s T tvoří úplnou abecedu systému; a množinu pravidel, prostřednictvím nichž provádí automat přechody. Během přechodu provede změnu aktuálního stavu a přečtení jednoho vstupního symbolu. Pokud konečný automat provede podle svých pravidel posloupnost přechodů tak, že začne v počátečním stavu, přečte celý vstupní řetězec a skončí ve stavu koncovém, pak říkáme, že automat přijímá vstupní řetězec. Množina všech přijímaných řetězců tedy tvoří jazyk definovaný konečným automatem. Při pohledu na konečný automat jako na přepisující systém je vnitřní stav tvořen vstupním řetězcem a aktuálním stavem (tzv. konfigurace). Při vhodném zápisu konfigurace pak při každém přechodu přepisujeme podřetězec skládající se z aktuálního stavu a právě čteného symbolu ze vstupu na řetězec obsahující pouze cílový stav pravidla (přečtený vstupní symbol je tímto vymazán).

Kombinování gramatik a automatů, což jsou, jak jsme si právě demonstrovali, obojí přepisující systémy, vede bezesporu opět ke konstrukci dalšího přepisujícího systému. Tato práce se mimo jiné zaměřuje právě na kombinovaný typ přepisujících systémů, který je v teorii formálních jazyků relativně opomíjený, protože až donedávna se studovaly modifikace gramatik a automatů více méně zcela odděleně a nezávisle.

Při studiu kombinací automatů a gramatik je výhodné všechny pojmy a vlastnosti (např. konfigurace, aktivní a pasivní symbol, atd.) definovat v kontextu přepisujících systémů, abychom některý pojem či vlastnosti zbytečně neomezovali pouze na jednu z oblastí a spíše směřovali k unifikaci těchto dvou základních přístupů. Např. pojem konfigurace bude používán obecně pro označování popisu vnitřního stavu přepisujícího systému, takže i v případě gramatiky budeme pojem větné formy generalizovat na konfiguraci.

Kombinování automatů a gramatik i sjednocení pojmů budeme demonstrovat v rámci celé práce, ale především na dvou nových přepisujících systémech (kapitola 4): #-přepisujícím systému a redukujícím hlubokém zásobníkovém automatu.

Zaměření této dizertační práce je však ještě užší. Většina jazyků i tříd jazyků lze popsat či charakterizovat více formálními modely či konkrétněji přepisujícími systémy. Při výběru vhodného modelu/systému je proto třeba brát v úvahu jednu důležitou vlastnost — složitost. Složitost se

studuje prakticky od založení teorie formálních jazyků a obecně se dělí na dvě související větve: složitost popisná (neboli statická) a složitost prostorová a časová, jež má spíše dynamický charakter:

- popisná složitost (*Descriptional Complexity*) se zabývá prostorovou složitostí samotného popisu jazyka (např. kolik pravidel nebo neterminálních symbolů je dostačujících pro popis daného jazyka nebo třídy jazyků);
- prostorová a časová složitost (*Space/Time Complexity*) se naopak zaměřuje na efektivitu zpracování vět jazyků, takže význam pro praxi je ještě větší než v prvním případě.

Bohužel prostorová a časová složitost přistupuje k této charakterizaci formálních modelů značně hrubým způsobem, kdy pouze zjišťují třídy složitosti (např. asymptotické) daného modelu (např. nejhůře kubická časová složitost přijímání věty libovolného bezkontextového jazyka v závislosti na délce věty). Navíc v popisné i prostorové a časové složitosti je trend sice zlepšovat sledované metriky formálního modelu či jeho způsobu práce, ale pokud možno bez ovlivňování vyjadřovací síly zkoumaných modelů.

My se v této práci pokusíme o jiný přístup a pod větším drobnohledem se podíváme především na prostorovou složitost. Budeme omezovat některé specifické metriky vztahující se k prostorové složitosti, a pak budeme studovat získané třídy jazyků definované takto omezenými přepisujícími systémy. Za tímto účelem zavedeme v kapitole 3 kromě klasifikace omezování přepisujících systémů i pojem dynamická složitost.

Již v 70. letech 20. století se studovaly první řízené gramatiky (např. maticové nebo programované). Hlavní motivací pro jejich zavedení bylo získání silnějších vyjadřovacích prostředků, a to pouze díky malé modifikaci tradičních a jednoduchých bezkontextových gramatik. Při zobecnění pojmu řízení se dostaneme k pojmu omezování formálních modelů, protože ono řízení nám většinou tento model skutečně omezuje, např. výběr pravidel aplikovatelných v následujícím kroku u programovaných gramatik.

Omezování formálních modelů lze opět rozdělit například takto:

- omezování přepisujících systémů (látka, kterou budeme studovat podrobněji dále);
- omezování domény, nad kterou jazyk definujeme (např. místo symbolů abecedy můžeme použít slova konečného jazyka, nebo místo využití volného monoidu definujeme jazyk pomocí volné grupy atp.);
- ostatní (sem bychom mohli zařadit například požadavky na splňování různých algebraických vlastností, jako např. uzavřenost jazyka vůči substituci apod.)

Nejčastěji je studována právě první varianta, do které patří například (1) omezování přechodů mezi konfiguracemi nebo přímo (2) omezování konfigurací samotných. Druhý případ v kontextu kombinování přepisujících systémů potom v této práci tvoří cílovou oblast studia.

1.1 Přehled kapitol

Předkládaná dizertační práce je členěna do 4 částí, které se dále dělí na celkem 7 kapitol:

1. Současný stav (kapitoly 1, 2 a 3)
2. Nové formální modely a jejich omezování (kapitoly 4 a 5)

3. Relace k praxi (kapitola 6)

4. Závěr (kapitola 7)

Úvodní kapitola představuje práci jako celek, včetně hlavních motivačních aspektů.

V druhé kapitole jsou zopakovány všechny pojmy a definice (od základních po pokročilé), nejen z teorie formálních jazyků, používaných v následujícím textu. Kromě toho zobecníme formou konceptů pojmy z oblasti formálních jazyků jako formální model, instance a konfigurace. Formálněji se budeme věnovat definici jednoho sjednocujícího formálního modelu—přepisujícího systému—a jeho speciálním variantám: gramatikám a automatům. Zdůrazněme, že následující kapitoly již obsahují samotný výzkum autora (není-li explicitně řečeno jinak).

Třetí kapitola klasifikuje formální modely, potažmo přepisující systémy, se zaměřením na jejich omezování. Pojem omezování je zde chápán v obecnějším kontextu jako úzce spjatý s pojmem složitost. Omezování formálních modelů zde i z pohledu složitosti dělíme na statické a dynamické, případně ještě hybridní. Práce se soustředí na výsledky z oblasti hybridního a dynamického omezování přepisujících systémů, specificky se zaměřuje na omezování posloupnosti použitých přepisujících pravidel a omezování konfigurací. Na konci kapitoly je vyzdvižen vztah omezování konfigurací k časové a hlavně prostorové složitosti a vztah k praktickému fungování přepisujících systémů. Pro specifičtější způsob práce s prostorovou složitostí je zaveden pojem dynamické složitosti, který studuje prostorovou složitost s důrazem na konkrétní způsoby omezování konfigurací, jako je například limitovanost nebo konečný index. Samozřejmě hranice mezi omezováním dynamickým, statickým a především hybridním není vždy naprosto patrná, což je způsobeno principiální závislostí dynamické složitosti na té statické.

V kapitole 4 jsou definovány nové typy řízených přepisujících systémů, které budou ve zbytku práce podrobněji studovány z pohledu dynamické složitosti. Nejprve si představíme #-přepisující systém, což je řízený přepisující systém kombinující vlastnosti gramatik a automatů. Od gramatik přebírá generativní způsob vytváření vět jazyka a od automatů konečně-stavové řízení. Navíc ještě obsahuje statické omezení na pouze jediný symbol, jenž je možno přepisovat pomocí pravidel, která určují kolikátý tento symbol musí být od začátku konfigurace (tj. zleva). Kromě základního #-přepisujícího systému založeného na pravidlech bezkontextového tvaru představujeme také pravělineární a kontextové tvary pravidel. Dalším modelem je zásobníkový automat s omezeným obsahem zásobníku, kde máme zadán jazyk, který omezuje povolené obsahy zásobníku tohoto modelu, což je typická ukázka dynamického omezování. Poslední zavedený model je založen na redukující modifikaci hlubokých zásobníkových automatů.

Většina vlastností formálních jazyků se vztahuje na celou třídu těchto jazyků. Třída jazyků bývá velmi často specifikována právě formálním modelem, který je schopen popsat právě ty a jen ty jazyky patřící do dané třídy. Vyjadřovací síla formálních modelů, potažmo tříd jazyků těmito modely popisovaných, je tudíž klíčovou vlastností pro získání informací o dalších vlastnostech, které se k této třídě váží, a má proto význam tuto vlastnost přednostně studovat u všech nových formálních modelů. Nejinak tomu bude i v případě páté kapitoly, která shrnuje výsledky a vlastnosti nových i existujících formálních modelů (ověřeno konstrukčními důkazy), především týkající se omezování těchto modelů. Ve většině případů na základě omezení dynamické složitosti (např. n-limitovanost nebo konečný index) získáme nekonečné hierarchie jazyků, což je vlastnost významná pro důkladné studium vyjadřovacích schopností daného modelu. Nekonečné hierarchie modelů jsou v přiložených důkazech založeny na demonstrování ekvivalence s jiným formálním modelem, který tvoří nekonečnou hierarchii, a tím pádem získáme stejnou vlastnost i pro námi zkoumaný model.

Těžiště této práce je v čistě teoretické oblasti, ale i přesto se předposlední, šestá, kapitola zaměřuje na klíčové vlastnosti zavedených přepisujících systémů pro jejich případné praktické

využití. Jedná se o determinismus a kanonické přepisování, které jsou esenciální například při zpracování bezkontextových jazyků.

Závěrečná kapitola nastiňuje další vývoj práce a některé předpokládané budoucí výsledky, které byly v době psaní textu pouze rozpracovány. Kromě hypotéz se zmíníme také o nejpodstatnějších otevřených problémech.

1.2 Konvence

Za účelem pohodlnějšího psaní, ale především lepší čitelnosti a jednotnosti, je v textu kromě klasických definic také několik konvencí, které slouží pro lepší orientaci ve vytvořeném matematickém značení a dalších pasážích.

Konvence 1.1. Notace některých zápisů jsou převzaty nebo inspirovány knihami [48, 53] a [66, 67, 68].

Méně formální definice budeme označovat jako koncepty.

Konvence 1.2. Budeme se snažit vycházet ze zavedených českých pojmů ([9], [13]), ale mnoho pojmů nemá v českém jazyce jednoznačně ustálený výraz. S cílem lépe navázat na anglickou literaturu budeme tedy u některých pojmů uvádět v kulatých závorkách kurzívou i anglické ekvivalenty (*equivalents*).

Kapitola 2

Základní pojmy

V této sekci jsou definovány již existující formální modely, jejichž znalost je podmínkou pro studium dalších kapitol, které budou tyto pojmy již bez dalšího vysvětlování využívat. Nejzákladnější pojmy budou definovány maximálně stručně a pro hlubší studium i samotný úvod do této problematiky doporučuji nejprve některou z přehledových knih teorie formálních jazyků ([39, 48]) a překladačů ([1, 2, 3, 4]).

Pokročilé formální modely, které jsou často studovány až v rámci doktorského studijního programu a nebo v rámci výzkumu jsou popsány podrobněji, a to včetně příkladů ([12, 49, 64]).

2.1 Základní matematické definice

Pro úplné ujednocení pojmů zmiňme i několik naprosto základních z obecné matematiky ([71]), které se týkají především množin a relací. Kromě toho uveďme i několik značení, která budou využívána v textu práce.

Pojem množina budeme používat v intuitivním smyslu jako v tzv. naivním přístupu k množinám. Neformálně řečeno je *množina* Σ kolekce elementů převzatých z nějakého předspecifikovaného univerza. Náležitost elementu a do množiny Σ symbolicky značíme $a \in \Sigma$ a hovoříme o a jako o prvku množiny Σ. Naopak, pokud a není v Σ, píšeme $a \notin \Sigma$.

Kardinalita množiny Σ, $\mathrm{card}(\Sigma)$, je počet prvků[1] Σ. Množina, která nemá žádný prvek se nazývá *prázdná* a je značena $\emptyset$. Poznamenejme, že $\mathrm{card}(\emptyset) = 0$. Má-li Σ konečný počet prvků, pak je Σ *konečná množina*. Jinak je Σ *množinou nekonečnou*.

Konečnou množinu Σ lze nejjednodušeji specifikovat výpisem všech jejích prvků, tedy

$$\Sigma = \{a_1, a_2, \ldots, a_n\},$$

kde a_1 až a_n jsou všechny prvky množiny Σ. Nekonečnou množinu Ω obvykle popisujeme pomocí vlastnosti π tak, že Ω obsahuje všechny prvky z daného univerza, které splňují onu podmínku π (tzv. *množinový konstrukt*). Symbolický zápis dodržuje následující formát:

$$\Omega = \{a |\ \pi(a)\}\,.$$

Množiny, jejichž prvky jsou opět jiné množiny, většinou označujeme pojmem *třídy množin*, místo množiny množin.

[1] V literatuře se někdy používá také zápis $|\Sigma|$, který ale budeme používat výhradně pro délku posloupností.

Nechť jsou Σ a Ω dvě množiny. Σ je *podmnožina* Ω, symbolicky zapisováno jako $\Sigma \subseteq \Omega$, pokud každý prvek Σ také patří do Ω. Σ je *vlastní podmnožina* Ω, zapisováno jako $\Sigma \subset \Omega$, je-li $\Sigma \subseteq \Omega$ a Ω obsahuje prvek, který není obsažen v Σ. Jestliže $\Sigma \subseteq \Omega$ a $\Omega \subseteq \Sigma$, je Σ ekvivalentní Ω, značeno $\Sigma = \Omega$. *Potenční množina* množiny Σ, značíme jako 2^{Σ}, je množina všech podmnožin množiny Σ. Pro množiny Σ a Ω označme jejich sjednocení, průnik a rozdíl jako $\Sigma \cup \Omega$, $\Sigma \cap \Omega$, a $\Sigma - \Omega$. Tyto binární množinové operace definujme takto:

$$\Sigma \cup \Omega = \{a|\ a \in \Sigma \text{ nebo } a \in \Omega\},$$

$$\Sigma \cap \Omega = \{a|\ a \in \Sigma \text{ a zároveň } a \in \Omega\} \text{ a}$$

$$\Sigma - \Omega = \{a|\ a \in \Sigma \text{ a zároveň } a \notin \Omega\}.$$

Komplement množiny Σ nad univerzem U označujeme jako $\overline{\Sigma}$ a je definován jako $\overline{\Sigma} = U - \Sigma$.

Definice 2.1. Nechť $\mathbb{N}_0$ označuje množinu všech nezáporných celých čísel, pak značme $\mathbb{N} = \mathbb{N}_0 - \{0\}$. Dále nechť $K \subseteq \mathbb{N}_0$ je konečná množina. Nyní definujme $\max(K)$ jako nejmenší číslo k takové, že pro všechna $h \in K$, $k \geq h$, a $\min(K)$ jako největší číslo l takové, že pro všechna $h \in K$, $l \leq h$.

Nyní zopakujme pojmy z oblasti relací, funkcí a jejich vlastností.

Definice 2.2. Máme-li dva objekty a a b, pak dvojicí (a, b) značme intuitivně známým pojmem *uspořádaná dvojice* (*ordered pair*) skládající se z objektu a a b právě v tomto pořadí. Nechť máme dále množiny A a B. Nyní můžeme definovat *kartézský součin* (*Cartesian product*) množin A a B, značeno $A \times B$, jako

$$A \times B = \{(a, b)|\ a \in A,\ b \in B\}.$$

Binární relace (*binary relation*), nebo zkráceně pouze *relace*, ρ z množiny A do množiny B je libovolná podmnožina kartézského součinu $A \times B$. V symbolech tedy

$$\rho \subseteq A \times B.$$

Definiční obor (*domain*) relace ρ označovaný jako $\mathrm{Domain}(\rho)$ a *obor hodnot* (*range*) téže relace ρ, značeno $\mathrm{Range}(\rho)$, jsou definovány následovně:

$$\mathrm{Domain}(\rho) = \{a|\ (a, b) \in \rho \text{ pro nějaké } b \in B\}$$

a

$$\mathrm{Range}(\rho) = \{b|\ (a, b) \in \rho \text{ pro nějaké } a \in A\}.$$

Platí-li $A = B$, mluvíme o ρ jako o *relaci na* A. O relaci ρ na A říkáme (viz [71]), že je:

a) *reflexivní*, právě když $a\rho a$ pro libovolné $a \in A$;

b) *ireflexivní*, právě když $(a, a) \notin \rho$ pro všechna $a \in A$;

c) *symetrická*, právě když $a\rho b$ implikuje $b\rho a$ pro libovolné $a, b \in A$;

d) *antisymetrická*, právě když z $a\rho b$ a $b\rho a$ plyne $a = b$;

e) *tranzitivní*, právě když z $a\rho b$ a $b\rho c$ plyne $a\rho c$.

Reflexivní, antisymetrická a tranzitivní relace ρ na množině A se nazývá *uspořádání* a dvojice (A, ρ) se nazývá *uspořádaná množina*.

Mějme množinu A a uspořádání $\leq$ na A. Pokud pro libovolné dva prvky $a, b \in A$ platí, že $a \leq b$ nebo $b \leq a$, tak $\leq$ nazýváme *lineární uspořádání* a $(A, \leq)$ *lineárně uspořádána množina*.

Ireflexivní a tranzitivní relaci $<$ na množině A nazvěme *ostré uspořádání*, dvojici $(A, <)$ pak *ostře uspořádanou množinou*.

Definice 2.3. Buď $(X, \leq)$ uspořádaná množina, $A \subseteq X$. Říkáme, že $x \in X$ je *horní závora* (resp. *dolní*) množiny A, jestliže pro všechna $a \in A$ je $a \leq x$ (resp. $x \leq a$).

Dále prvek $x \in X$ nazveme *největším* (resp. *nejmenším*) prvkem v $(X, \leq)$, jestliže $y \leq x$ (resp. $x \leq y$) pro každé $y \in X$. Existuje-li mezi všemi horními závorami množiny A nejmenší, označujeme ji $\sup(A)$ a nazýváme *supremem* množiny A. Naopak, existuje-li největší dolní závora množiny A, označujeme ji $\inf(A)$ a nazýváme *infimem* množiny A.

Definice 2.4. *Funkce* (*function*) nebo též *zobrazení* (*mapping*) z A do B je relace ϕ z A do B taková, že pro každé $a \in A$,

$$\mathrm{card}(\{b|\ b \in B,\ (a,b) \in \phi\}) \leq 1.$$

Dále nechť ϕ značí funkci z A do B. Je-li $\mathrm{Domain}(\phi) = A$, ϕ je tzv. *totální funkce*, jinak ji nazveme *parciální*. Pokud pro každé $b \in B$, $\mathrm{card}(\{a|\ a \in A,\ (a,b) \in \phi\}) \leq 1$, ϕ je *injekce*. Pokud pro každé $b \in B$, $\mathrm{card}(\{a|\ a \in A,\ (a,b) \in \phi\}) \geq 1$, ϕ je *surjekce*. Je-li ϕ surjekce a zároveň injekce, mluvíme o *bijekci*.

Místo zápisu $(a,b) \in \rho$ se často používá $a \in \rho(b)$ a ještě častěji $a\rho b$. Jinými slovy, $(a,b) \in \rho$, $a \in \rho(b)$ a $a\rho b$ jsou obdobné zápisy vyjadřující totéž. Především poslední zápis budeme využívat velmi často. Je-li navíc relace ρ funkcí, používáme zápis tvaru $\rho(a) = b$.

Nechť ρ je relace nad množinou A. Pro $k \geq 1$ definujme *k-násobný součin* (*k-fold product*) relace ρ (někdy též k-tá mocnina ρ), ρ^k, pomocí rekurzivní metody takto: (1) $a\rho^1 b$, když a jen když $a\rho b$ a (2) $a\rho^k b$, když a jen když $a\rho c$ a $c\rho^{k-1} b$, pro nějaké c a $k \geq 2$. *Tranzitivní uzávěr* (*transitive closure*) relace ρ, ρ^+, je definován jako $a\rho^+ b$, když a jen když $a\rho^k b$ pro nějaké $k \geq 1$. *Reflexivně-tranzitivní* (*reflexive-transitive*) uzávěr relace ρ, ρ^*, je definován jako $a\rho^* b$, když a jen když $a\rho^+ b$ a zároveň $a\rho^* a$ pro každé $a \in A$.

2.2 Formální jazyky

Dále definujme pojmy a notace týkající se jazyků, tříd jazyků a operací nad jazyky.

Posloupnost je intuitivně známý pojem pro seznam prvků z nějakého univerza. Posloupnost je *konečná*, reprezentuje-li konečný seznam prvků, jinak je *nekonečná*. *Délka* konečné posloupnosti x, značená jako $|x|$, je počet elementů v x. Prázdnou posloupnost značíme jako ε, a je to posloupnost neobsahující žádný prvek, tedy $|\varepsilon| = 0$. Konečnou posloupnost obvykle reprezentujeme seznamem jejích prvků. Například uvažujme konečnou posloupnost x popsanou seznamem $x = 0, 1, 0, 0$. Tedy $|x| = 4$.

Abeceda T je konečná, neprázdná množina, jejíž prvky nazýváme *symboly*. Konečnou posloupnost symbolů z T nazývejme *řetězec* (nebo též *slovo*) nad T. Konkrétně ε pak nazýváme *prázdným řetězcem*. Pomocí T^* značíme množinu všech řetězců nad T, dále $T^+ = T^* - \{\varepsilon\}$. Libovolnou podmnožinu $L \subseteq T^*$ nazýváme *jazykem* nad T. Je-li L konečná množina řetězců, je L *konečný jazyk*, jinak je L *jazyk nekonečný*. Například T^* (nazývaný též univerzálním jazykem nad T) je nekonečným jazykem, zatímco $\emptyset$ a $\{\varepsilon\}$ jsou konečné. Uvědomme si, že $\emptyset \neq \{\varepsilon\}$ protože $\mathrm{card}(\emptyset) = 0 \neq \mathrm{card}(\{\varepsilon\}) = 1$. Pro konečný jazyk L označujme funkcí $\max(L)$ délku nejdelšího slova jazyka L. Analogicky s intuitivní definicí množiny budeme množiny, jejichž prvky jsou jazyky, nazývat *třídami jazyků*.

Konvence 2.1. Zaveďme si pomocnou konvenci pro vynechávání všech oddělovacích čárek v řetězcích. To znamená, že píšeme $a_1 a_2 \ldots a_n$ místo $a_1, a_2, \ldots, a_n$.

Definice 2.5. Nechť $x, y \in T^*$ jsou dva řetězce nad libovolnou abecedou T a nechť $L, K \subseteq T^*$ jsou dva jazyky nad T. Jelikož jsou jazyky vlastně speciální typy množin, lze na ně aplikovat všechny

operace dosud zavedené pro množiny. Konkrétně $L \cup K$, $L \cap K$ a $L - K$ označují sjednocení, průnik a rozdíl jazyků L a K. Nejpodstatnější operace je operace *konkatenace* (někdy jen katenace) řetězce x s řetězcem y značená jako xy. xy je pak řetězec získaný připojením y k x. Poznamenejme, že z algebraického pohledu jsou T^* a T^+ *volný monoid* a *volná semigrupa* generovaná operací konkatenace. Všimněme si, že pro každé $w \in T^*$, $w\varepsilon = \varepsilon w = w$. Konkatenace L a K, značená jako LK, je definována jako

$$LK = \{xy |\ x \in L,\ y \in K\}\,.$$

Kromě binárních operací převzatých z teorie množin zavedeme nad řetězci a jazyky také několik unárních, případně binárních operací.

Definice 2.6. Nechť T je abeceda a nechť $x \in T^*$ a $L \subseteq T^*$. *Komplement* jazyka L, značený jako $\overline{L}$, je definován jako $\overline{L} = T^* - L$.

Reverzace řetězce x budeme značit $\mathrm{rev}(x)$. Jedná se o řetězec x zapsaný v opačném pořadí, tedy je-li $x = a_1 a_2 \dots a_n$, kde $a_1, a_2, \dots, a_n$ jsou symboly řetězce x, pak $\mathrm{rev}(x) = a_n \dots a_2 a_1$.

Pro všechna $i \geq 0$ definujme i-tou *mocninu řetězce* x, značme x^i, pomocí rekurzivní definice: (1) $x^0 = \varepsilon$ a (2) $x^i = xx^{i-1}$ pro všechna $i \geq 1$. Všimněte si, že definice je založená na rekurzivní metodě. Uvažujme například i-tou mocninu x^i s $i = 3$ a demonstrujme tento rekurzivní definiční přístup. Podle druhé části definice $x^3 = xx^2$. Po opakované aplikaci druhé části definice na x^2 získáme $x^2 = xx^1$ a pro x^1 platí $x^1 = xx^0$. Rekurzivní definice mocniny řetězce je zakončena pravidlem (1), které definuje x^0 jako prázdný řetězec, tj. $x^0 = \varepsilon$. Tedy $x^1 = xx^0 = x\varepsilon = x$. Dále $x^2 = xx^1 = xx$. A nakonec $x^3 = xx^2 = xxx$. Pohodlným a výstižným matematickým zápisem pomocí rekurzivní metody definujeme i některé další pojmy, včetně i-té mocniny jazyka L, L^i, jež je definována jako (1) $L^0 = \{\varepsilon\}$ a (2) $L^i = LL^{i-1}$ pro každé $i \geq 1$. *Uzávěr* jazyka L, L^*, definujme jako

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

a *pozitivní uzávěr* jazyka L, L^+, jako

$$L^+ = \bigcup_{i \geq 1} L^i.$$

K tomu poznamenejme, že

$$L^+ = LL^* = L^*L$$

a

$$L^* = L^+ \cup \{\varepsilon\}.$$

Literatura také často místo uzávěru a pozitivního uzávěru jazyka může mluvit o reflexivně-tranzitivním a tranzitivním uzávěru jazyka.

Existuje-li $z \in T^*$ takové, že $xz = y$, pak x je tzv. předpona (*prefix*) řetězce y. Navíc, platí-li $x \notin \{\varepsilon, y\}$, je x dokonce *vlastní předpona* řetězce y. Funkcí $\mathrm{prefixes}(y)$ označujme množinu všech ppředpon řetězce y.

Analogicky je-li $z \in T^*$ a platí pro něj $zx = y$, tak x nazýváme *příponou* (*suffix*) řetězce y. Je-li k tomu $x \notin \{\varepsilon, y\}$, označujeme x *vlastní příponou* (*proper suffix*) řetězce y. Podobně $\mathrm{suffixes}(y)$ označuje množinu všech přípon řetězce y.

Pro libovolné $n \geq 0$ a $x \in T^*$ označuje $\mathrm{prefix}(x, n)$, resp. $\mathrm{suffix}(x, n)$ předponu, resp. příponu řetězce x délky n. Je-li $|x| < n$, pak je $\mathrm{prefix}(x, n) = \mathrm{suffix}(x, n) = x$.

Mějme řetězec $w \in T^*$ a množinu K, $\{\varepsilon\} \subseteq K \subseteq T$. $\mathrm{maxprefix}(w, K)$, resp. $\mathrm{maxsuffix}(w, K)$ označuje nejdelší předponu, resp. příponu řetězce w, jejíž všechny symboly jsou v K.

Existují-li řetězce $u, v \in T^*$ takové, že $uxv = y$, říkáme o x, že je podřetězcem či podslovem (*substring*, *subword*) řetězce y. Dále je-li $x \notin \{\varepsilon, y\}$, mluvíme o x jako o *vlastním podřetězci*, respektive *vlastním podslově* řetězce y. Množinu všech podřetězců řetězce y značíme $\mathrm{sub}(y)$.

Stojí za povšimnutí, že pro každé slovo w platí

$$\mathrm{prefixes}(w) \subseteq \mathrm{sub}(w),$$

$$\mathrm{suffixes}(w) \subseteq \mathrm{sub}(w),$$

a

$$\{\varepsilon, w\} \subseteq \mathrm{prefixes}(w) \cap \mathrm{suffixes}(w) \cap \mathrm{sub}(w).$$

Pokud w označuje neprázdné slovo, pak $\mathrm{first}(w)$ značí nejlevější symbol v řetězci w a $\mathrm{last}(w)$ nejpravější symbol v tomtéž řetězci. Dále pomocí $\mathrm{sym}(w, i)$ označujme i-tý symbol zleva v řetězci w pro $1 \leq i \leq |w|$. Pro libovolné slovo w, $\mathrm{alph}(w)$ označuje množinu všech symbolů, které se vyskytují ve slově w (např. $\mathrm{alph}(aAbAaBCc) = \{a, b, c, A, B, C\}$). Opět rozšiřme definici pro celý jazyk,

$$\mathrm{alph}(L) = \bigcup_{y \in L} \mathrm{alph}(y).$$

Pro dva řetězce x a y, kdy $|y| \geq 1$, označuje $\mathrm{occur}(y, x)$ počet výskytů y v x. Ještě obecnější tvar funkce $\mathrm{occur}(W, x)$, kde W je konečný jazyk a $\varepsilon \notin W$, označuje počet všech výskytů podřetězců řetězce x, které patří do W. Například $\mathrm{occur}(\{a, C\}, aAbAaBCc) = 3$.

Mějme dvě abecedy T a U. Totální funkci τ z T^* do 2^{U^*} taková, že $\tau(uv) = \tau(u)\tau(v)$ pro každé $u, v \in T^*$ nazveme *substitucí* (*substitution*) z T^* do U^*. Podle této definice, $\tau(\varepsilon) = \varepsilon$ a $\tau(a_1 a_2 \ldots a_n) = \tau(a_1)\tau(a_2)\ldots\tau(a_n)$, kde $a_i \in T$, $1 \leq i \leq n$, pro libovolné $n \geq 1$. Z toho plyne, že τ je plně specifikováno definováním $\tau(a)$ pro každý symbol $a \in T$ a určení substituce pro řetězec již plyne ze samotné definice. Totální funkce χ z T^* do U^* taková, že $\chi(uv) = \chi(u)\chi(v)$ pro každé $u, v \in T^*$ je *homomorfismus* (*homomorphism*) či zkráceně jen *morfismus* z T^* do U^*. Protože homomorfismus je speciální případ substituce, což plyne ze samotné definice, specifikujeme konkrétní morfismus χ analogicky ke specifikaci substituce τ.

Konvence 2.2. Symboly abeced budeme vybírat převážně z malých a případně i velkých písmen začátku latinské abecedy (např. a, b, c, A, B, C). Naopak řetězce nad abecedou budeme většinou označovat malými písmeny z konce latinské abecedy (např. u, v, w, x, y, z) nebo výjimečně malá písmena z počátku řecké abecedy (např. $\alpha, \beta, \gamma, \delta$). Pro číselné proměnné a konstanty se s oblibou využívají písmena prostředka abecedy jako i, j, k, m, n.

Následuje poslední a trochu komplikovanější definice operace multipodřetězce libovolného řetězce. V podstatě se jedná o vymazání konečného množství podřetězců v daném řetězci a výsledek této operace označíme za multipodřetězec (podobné pokročilejší operace nad řetězci jsou studovány například v [20, 54]).

Definice 2.7. Nechť Σ je abeceda a $x, y_i, z_j \in \Sigma^*$, kde $1 \leq i \leq n$, $0 \leq j \leq n$, pro nějaké $n \in \{0, 1, \ldots, |x|\}$ a $x = z_0 y_1 z_1 \ldots y_n z_n$. Potom $y_1 y_2 \ldots y_n$ nazýváme *multipodřetězec* (*multisubstring*) řetězce x. Dále nechť $\mathrm{multisub}(x)$ označuje množinu všech multipodřetězců řetězce x.

Pro dokončení definic základních pojmů ještě uveďme pojem n-tice a pojmy s ním související.

Definice 2.8. Nechť $A_1, A_2, \ldots, A_n$ jsou množiny a nechť máme libovolné objekty $a_1, a_2, \ldots, a_n$, kde $a_i \in A_i$ pro všechna $1 \leq i \leq n$, $n \geq 1$. Pokud tyto objekty utvoří konečnou neprázdnou posloupnost $(a_1, a_2, \ldots, a_n)$, budeme tuto posloupnost nazývat *n-tice* (*n-tuple*) a její prvky *komponenty* (*components*).

Nechť S označuje n-tici $(a_1, a_2, \ldots, a_n)$. Komponentu a_i, $1 \leq i \leq n$, nazveme i-tou komponentou n-tice S. Pak pro vybrání i-té komponenty z n-tice S definujme funkci component(S, i), takto component$(S, i) = a_i$.

2.3 Koncept formálního modelu

V následující sekci uvedeme sadu konceptů i definic pro sjednocení pojmů ve snaze unifikovat přístup k teorii automatů a teorii gramatik.

Literatura se většinou tomuto přístupu vyhýbá, protože bývá úzce zaměřena jen na jednu ze zmíněných oblastí teorie formálních jazyků. Naopak v této práci se zaměřujeme na možnost kombinace gramatik a automatů, a proto hledání společných rysů jde ruku v ruce se společným pojmenováním pojmů, které mají stejný základ v obou teoriích.

Na druhou stranu to, že snaha o jistý druh unifikace a sjednocování přístupu k různých oblastem i aspektům teorie formálních jazyků není jev naprosto ojedinělý, dokazují například [14], [53] a [78].

Nyní si popišme koncept (neformální definici) pojmu formální model.

Obecná charakterizace tohoto pojmu mimo teorii formálních jazyků je (citujeme www.google.com):

Formální model je matematická reprezentace studovaného konceptu. Formální modely jsou základem pro jakýkoli návrh jazyka a pro rigorózní vědecké studium obecně.

V této práci chápeme pojem formální model jako matematický mechanismus, který přesně popisuje způsob, jakým jeho instance definují věty, potažmo množiny vět, tedy jazyky. Formálním modelem můžeme zaštiťovat mechanismus definice jazyka prostřednictvím algebraických operací a vlastností (tzv. deklarativní přístup), a nebo procedurální přístup, který využívají přepisující systémy (například automaty a gramatiky).

Koncept 2.1. *Formální model* je mechanismus, který pomocí modelu a sémantiky modelu definuje třídu jazyků.

Například bezkontextové gramatiky, zásobníkové automaty, Turingovy stroje ([48]) nebo L-systémy ([64]) jsou formální modely.

Koncept 2.2. *Instance formálního modelu* je konkretizace formálního modelu, která definuje konkrétní jazyk. Tomuto jazyku budeme říkat *jazyk definovaný instancí formálního modelu.*

Instancí formálního modelu je například konkrétní gramatika nebo automat, který definuje již zcela konkrétní jazyk (např. kontextová gramatika generující jazyk $\{a^{2^n} \mid n \geq 1\}$ nebo Turingův stroj přijímající jazyk $\{b(a^i b)^{2n} \mid i \geq 1,\ n \geq 1\}$).

V rámci zjednodušeného pojmenovávání se můžeme setkat se situací, kdy pojem formální model označuje spíše instanci formálního modelu, což bude plynout z kontextu.

2.3.1 Přepisující systémy

Pro teorii formálních jazyků je bezesporu nejdůležitějším typem formálních modelů přepisující systém, který zaštiťuje algoritmický přístup ke zpracování vět jazyka. Přepisující systém pomocí přepisování částí vnitřního stavu/paměti provádí výpočet přesně podle stanovených pravidel konkrétního modelu. Svou obecností je vhodný pro sjednocení světa generativních gramatik i světa akceptujících automatů ([68]).

Definice 2.9. *Přepisující systém* je dvojice, $H = (\Sigma, R)$, kde Σ je abeceda a R je konečná relace na Σ^*. Σ nazýváme úplnou abecedou systému H. Prvky R nazýváme pravidla systému H, takže R označujeme za množinu pravidel systému H.

Konvence 2.3. Každé pravidlo $(x, y) \in R$ zapisujme jako $x \to y$. Pro zkrácení často označujeme $x \to y$ pomocí návěští r, tedy $r\colon x \to y$, a místo $r\colon x \to y \in R$ si dovolujeme někdy zjednodušeně psát $r \in R$. Pro $r\colon x \to y \in R$, x a y reprezentují levou stranu a pravou stranu pravidla r, značeno lhs(r) a rhs(r). R^* značí množinu všech posloupností pravidel z R. Pomocí $\rho \in R^*$ lze zkráceně vyjádřit, že ρ je posloupnost skládající se z $|\rho|$ pravidel z R. Analogicky s definicí řetězců budeme i v řetězcích pravidel vynechávat čárky mezi jednotlivými komponentami. Tedy místo $\rho = r_1, r_2, \ldots, r_n$, budeme psát ρ jako $r_1 r_2 \ldots r_n$. Pro explicitní vyjádření, že Σ a R jsou komponentami H, budeme psát Σ_H a R_H.

V podobných přepisujících systémech často definujeme některé pojmy zcela stejně nebo analogicky. Pro lepší přehlednost tedy zavedeme implicitní definice některých pojmů zobecněných pro přepisující systémy, a tudíž platící pro jejich různé varianty.

Definice 2.10. Nechť $H = (\Sigma, R)$ je přepisující systém. *Přepisující relace* na Σ^* je označena pomocí $\Rightarrow$ a definována tak, že pro každé $u, v \in \Sigma^*$, $u \Rightarrow v$ v systému H, když a jen když existuje pravidlo $x \to y \in R$ a $w, z \in \Sigma^*$ takové, že $u = wxz$ a $v = wyz$. Nechť $u, v \in \Sigma^*$. Jestliže $u \Rightarrow v$ v H, budeme říkat, že H přímo přepisuje u na v. Jako obvykle pro každé $n \geq 0$, n-násobný součin relace $\Rightarrow$ je značen jako $\Rightarrow^n$. Pokud $u \Rightarrow^n v$, H přepisuje u na v v n krocích, tzv. n-kroký přepis. Dále, tranzitivně-reflexivní uzávěr a tranzitivní uzávěr relace $\Rightarrow$ klasicky zapisujeme $\Rightarrow^*$ a $\Rightarrow^+$. Jestliže $u \Rightarrow^* v$, jednoduše řekneme, že H přepisuje u na v, a jestliže $u \Rightarrow^+ v$, H přepisuje u na v netriviálním způsobem.

Někdy je zapotřebí explicitně specifikovat pravidla použitá během přepisování. Předpokládejme, že H provede $u \Rightarrow v$ tak, že $u = wxz$, $v = wyz$ a H nahradí x řetězcem y aplikací pravidla $r\colon x \to y \in R$. Abychom tuto aplikaci pravidla symbolicky vyjádřili, píšeme $u \Rightarrow v\ [r]$ nebo detailněji $wxz \Rightarrow wyz\ [r]$ v H a říkáme, že H přímo přepisuje uxv na uyv podle r. Obecněji, nechť n je nezáporné celé číslo, $w_0, w_1, \ldots, w_n$ posloupnosti, kde $w_i \in \Sigma^*$, $0 \leq i \leq n$, a $r_j \in R$ pro $1 \leq j \leq n$. Pokud $w_{j-1} \Rightarrow w_j\ [r_j]$ v H pro $1 \leq j \leq n$, H přepisuje w_0 na w_n v n krocích podle pravidel $r_1 r_2 \ldots r_n$, symbolicky zapsáno jako $w_0 \Rightarrow^n w_n\ [r_1 r_2 \ldots r_n]$ v H ($n = 0$ v případě, že $w_0 \Rightarrow^0 w_0\ [\varepsilon]$). Zápisem $u \Rightarrow^* v\ [\rho]$, kde $\rho \in R^*$, vyjadřujeme, že H provádí $u \Rightarrow^* v$ za použití ρ; $u \Rightarrow^+ v\ [\rho]$ má analogický význam ($\rho \neq \varepsilon$). Samozřejmě kdykoli je specifikování aplikovaných pravidel nadbytečné, můžeme od něj upustit a zkráceně psát $u \Rightarrow v$, $u \Rightarrow^n v$ a $u \Rightarrow^* v$.

Konvence 2.4. Chceme-li explicitně vyjádřit, pomocí kterého přepisujícího systému byl přepisující krok proveden, zapíšeme ho pomocí symbolu pro krok s označením systému v levém dolním indexu. Například krok $x \Rightarrow y$ v H můžeme zapsat jako $x \Rightarrow_H y$.

Konvence 2.5. Pokud nebude u definice nového přepisujícího systému specifikováno rozšíření jeho relace přepisujícího kroku na tranzitivní a reflexivně-tranzitivní uzávěry, bude se implicitně uvažovat obecná definice 2.10.

Protože mezi přepisující systémy neřadíme deklarativní způsoby popisu jazyka (například množinový zápis, neboli tzv. množinový konstrukt či výčet), má smysl zavést pojem pro označení a popis celkového vnitřního stavu instance přepisujícího systému během zpracovávání konkrétní věty konkrétního jazyka. Konečné posloupnosti komponent zachycující vnitřní stav modelu budeme říkat konfigurace. Pojem konfigurace je hojně používán nejen v teorii automatů, ale také u gramatických systémů ([11], [27], druhý díl [64]) a často také kombinovaných systémů (např. [16] a [21]). V této práci jej chápeme ještě obecněji a budeme pojem konfigurace používat pro označení vnitřního stavu jakéhokoliv přepisujícího systému (včetně klasické gramatiky).

Koncept 2.3. *Konfigurace instance přepisujícího systému* (nebo zkráceně konfigurace přepisujícího systému nebo jen konfigurace) je dostatečně určující vnitřní stav přepisujícího systému a všech jeho výpočetních částí, který vypovídá o míře zpracovanosti vstupu (je-li nějaký), využití či stavu interní paměti (je-li přítomna) a případně i záznamu dosavadního výstupu systému. Matematicky je konfigurace přepisujícího systému obecně n-tice komponent s definovanou délkou, kde $n \geq 1$.

Mějme konfiguraci $c = (\Gamma_1, \Gamma_2, \ldots, \Gamma_n)$. Délku konfigurace pak definujme jako

$$|c| = |\Gamma_1| + |\Gamma_2| + \cdots + |\Gamma_n|.$$

Jelikož je konfigurace c posloupností, lze její prvky nazývat komponentami a i-tou komponentu vybírat funkcí component(c, i).

Komponenty konfigurací přepisujícího systému můžeme dělit na konečné a nekonečné. Konečné komponenty jsou nejčastěji forma konečně-stavového řízení přepisujícího systému (například konečné automaty nebo stavové gramatiky).

Speciálním případem je konfigurace v klasických gramatikách, kde je $n = 1$. Takové konfiguraci někdy říkáme *větná forma* a nebudeme ji uzavírat do kulatých závorek.

Tímto postupným zjemňováním pohledu na způsob definování třídy, jazyka a věty jsme obdrželi pojmy, které použijeme pro sjednocení společných vlastností a pohledů na svět gramatik a automatů, které se většinou studují odděleně.

Konvence 2.6. Nakonec doplňme, že všechny přepisující systémy v této práci předpokládají za základ bezkontextový tvar pravidel, tj. jeden symbol bývá přepisován (bez ohledu na kontext) obecně řetězcem symbolů, pokud není explicitně uvedeno jinak. Tyto systémy jsou dále případně doplněny o řízení či jiný způsob omezování (viz kapitola 3).

Ve studovaných přepisujících systémech se většinou vyskytují symboly, u kterých požadujeme, aby se nevyskytovaly ve větách tímto systémem definovaného jazyka. V generativních systémech je zvykem je nazývat neterminály a v akceptačních nevstupní symboly (a v konkrétnějších např. nevstupní zásobníkové symboly). V případě zobecnění na libovolný přepisující systém (viz definice 2.9) budeme takovýto symbol nazývat *proměnnou* (podobně jako v [73]). Ostatní symboly úplné abecedy nazveme *pasivními*. Tuto klasifikaci ještě upřesněme v konceptu 2.4.

Koncept 2.4. O symbolu úplné abecedy přepisujícího systému budeme říkat, že je:

a) *proměnná*, pokud lze tento symbol v konfigurace systému v některém kroku přepsat obecně na jiný symbol či řetězec (například neterminální symbol nebo nevstupní zásobníkový symbol).

   1) *aktivní*, pokud se jedná o proměnnou přepsatelnou v aktuální konfiguraci systému. Většinou nás bude zajímat konkrétní výskyt aktivního symbolu.
   2) *potenciálně-aktivní*, pokud proměnná není aktivní symbol, ale v některé budoucí konfiguraci (vzniklé přepisováním aktuální konfigurace) se může stát aktivním symbolem.

b) *pasivní*, pokud je to symbol úplné abecedy, který není proměnná. Většinou se jedná o symboly, ze kterých se skládají věty zpracovávané přepisujícím systémem (například terminální symboly nebo vstupní symboly).

Je dobré si všimnout, že proměnné v jednom modelu již nemusí patřit mezi proměnné v modelu jiném (například čistá gramatika nebo L-systém versus bezkontextová gramatika). Obecně řečeno, některé symboly dokonce mohou upravovat svou roli v průběhu výpočtu. Například neterminální symbol n-limitované gramatiky může být aktivní a v jiném kroku jen potenciálně-aktivní.

Konvence 2.7. Mějme přepisující systém $H = (\Sigma, R)$ a pravidlo $r \in R$. V následujícím textu (definicích a důkazech) budou, v případě, že to zlepší čitelnost, pravidla označena unikátním návěštím. Navíc v případě nutnosti budeme pravidla ohraničovat pomocí závorek $\lfloor$ a $\rfloor$. Pravidlo bude tedy v plném tvaru vypadat následovně:

$$\lfloor r \colon \mathrm{lhs}(r) \to \mathrm{rhs}(r) \rfloor$$

a v případě, že to nebude k újmě na čitelnosti, budeme jej značit zkráceně $r \colon \mathrm{lhs}(r) \to \mathrm{rhs}(r)$ nebo jen $\mathrm{lhs}(r) \to \mathrm{rhs}(r)$. Máme-li zavedeno pravidlo s návěštím, definujme také funkci $\mathrm{lab}(\lfloor r \colon \mathrm{lhs}(r) \to \mathrm{rhs}(r) \rfloor) = r$ pro zjištění návěští zadaného pravidla. Funkce $\mathrm{Lab}(P) = \{\mathrm{lab}(r) \mid r \in P\}$ pak bude sloužit pro získání množiny návěští z P, kde $P \subseteq R$. V případě dostatečné jednoznačnosti bude možné v konstrukčních důkazech zaměňovat pravidlo a jeho unikátní návěští.

Studium přepisujících systémů různých typů a kombinací vede k různým typům modelů, z nichž některé definují stejné jazyky. Instance formálních modelů, které definují stejné jazyky budeme nazývat ekvivalentními instancemi modelů a obecně formální modely definující stejnou třídu jazyků budeme označovat za ekvivalentní, tj. stejně mocné či stejně silné.

2.4 Definice základních přepisujících systémů

V této sekci se budeme věnovat všeobecně známým a důležitým formálním modelům z teorie formálních jazyků. Nejprve si probereme nejpodstatnější zástupce gramatik a v další podsekci i automatů. Vše zakončíme krátkým připomenutím pojmu Chomského hierarchie jazyků (gramatik a automatů).

2.4.1 Gramatiky

Tato sekce opakuje základní typy gramatik a automatů, jejichž znalost je nezbytná pro další kapitoly této dizertační práce.

Konkrétně budeme hovořit především o bezkontextových, kontextových a neomezených gramatikách a vztazích mezi nimi. V této souvislosti zmíníme i veledůležitou Chomského hierarchii jazyků. Při definicích jednotlivých gramatik začneme od nejobecnější verze a postupným omezováním pravidel budeme dostávat další a další typy gramatik, aniž bychom museli měnit základní definici gramatiky nebo mechanismus derivačních (přepisujících) kroků.

Připomeňme, že v gramatikách místo konfigurace používáme raději pojem větná forma a místo aktivních symbolů (viz koncept 2.4) budeme hovořit o neterminálních symbolech, a místo pasivních symbolů o terminálech.

Definice 2.11. *Neomezená gramatika* (*phrase-structure grammar*) je čtveřice

$$G = (\Sigma, T, P, S),$$

kde

Σ je *úplná abeceda*,

T je množina *terminálů* ($T \subset \Sigma$),

$P \subseteq \Sigma^*(\Sigma - T)\Sigma^* \times \Sigma^*$ je konečná relace,

$S \in \Sigma - T$ je *axiom* gramatiky G (nebo též počáteční neterminál).

Symboly z rozdílu $\Sigma - T$ označujeme jako *neterminály*. Dále je každá uspořádaná dvojice $(x, y) \in P$ nazývaná *pravidlo* (*production* či *rule*), symbolicky zapisované ve formě:

$$x \to y \in P.$$

Podle svého obsahu je tedy množina P nazývána *množinou pravidel* v G. Mějme pravidlo $p\colon x \to y \in P$, pak ustanovme, že $\mathrm{lhs}(p) = x$ a $\mathrm{rhs}(p) = y$. Přepisující relace se v G nazývá *přímé derivace* (*direct derivation*) a je to binární relace na Σ^* označovaná symbolem $\Rightarrow_G$ a definovaná následujícím způsobem. Nechť $p\colon x \to y \in P$, $u, v, z_1, z_2 \in \Sigma^*$ a $u = z_1 x z_2$, $v = z_1 y z_2$; pak,

$$u \Rightarrow_G v\ [p].$$

Pokud nehrozí nejednoznačnost, zjednodušíme zápis $u \Rightarrow_G v\ [p]$ na $u \Rightarrow_G v$. Dále $\Rightarrow_G^k$ označuje k-násobný součin relace $\Rightarrow_G$. Pomocí $\Rightarrow_G^+$ a $\Rightarrow_G^*$ označujeme tranzitivní uzávěr relace $\Rightarrow_G$ a tranzitivní a reflexivní uzávěr relace $\Rightarrow_G$. Platí-li $S \Rightarrow_G^* x$ pro nějaké $x \in \Sigma^*$, tak x nazýváme *větná forma* (*sentential form*).

Existuje-li derivace $S \Rightarrow_G^* w$, kde $w \in T^*$, pak derivaci $S \Rightarrow_G^* w$ říkáváme *úspěšná derivace* (*successful derivation*) v G. *Jazyk generovaný gramatikou* G, značený jako $L(G)$, je definován jako

$$L(G) = \{w \in T^* |\ S \Rightarrow_G^* w\}.$$

V některé literatuře jsou neomezené gramatiky také často definovány jako gramatiky s následující formou pravidel:

$$xAy \to xuy,$$

kde $u, x, y \in \Sigma^*$, $A \in \Sigma - T$ (viz [17]). Obě definice jsou zaměnitelné ve smyslu popisu stejné třídy jazyků oběma těmito formálními modely, které tímto definují třídu *rekurzivně spočetných jazyků* (*recursively enumerable languages*), značenou **RE**.

Definice 2.12. *Kontextová gramatika* (*context-sensitive grammar*) je neomezená gramatika,

$$G = (\Sigma, T, P, S),$$

taková, že každé pravidlo z P je následujícího tvaru

$$xAy \to xuy,$$

kde $A \in \Sigma - T$, $u \in \Sigma^+$, $x, y \in \Sigma^*$. *Kontextový jazyk* je potom jazyk generovaný kontextovou gramatikou a třídu kontextových jazyků značíme **CS**.

Definice 2.13. *Bezkontextová gramatika* (*context-free grammar*) je neomezená gramatika,

$$G = (\Sigma, T, P, S),$$

taková, že každé pravidlo $x \to y \in P$ splňuje podmínku, že $x \in \Sigma - T$. Bezkontextový jazyk je pak analogicky jazyk generovaný bezkontextovou gramatikou. Třída bezkontextových jazyků je označována jako **CF**.

Definice 2.14. *Lineární gramatika* (*linear grammar*) je neomezená gramatika,

$$G = (\Sigma, T, P, S),$$

taková, že každé pravidlo $x \to y \in P$ splňuje podmínku, že $x \in \Sigma - T$ a $\mathrm{occur}(\Sigma - T, y) \leq 1$. Takto definovaná lineární gramatika poté generuje lineární jazyk a třídu lineárních jazyků značme **LIN**.

Definice 2.15. *Regulární gramatika* (*regular grammar*), respektive pravě-lineární (*right-linear grammar*) je neomezená gramatika,

$$G = (\Sigma, T, P, S),$$

taková, že každé pravidlo $x \to y \in P$ splňuje $x \in \Sigma - T$ a $y \in T \cup T(\Sigma - T)$, respektive pro pravě-lineární pravidla platí, že $x \in \Sigma - T$ a $y \in T^* \cup T^*(\Sigma - T)$. Regulární, respektivě pravě-lineární gramatika definuje regulární, respektive pravě-lineární jazyk. Třída obou typů těchto jazyků je označována jako třída regulárních jazyků a budeme ji značit **REG**.

Konvence 2.8. V případě, že to bude žádoucí pro lepší čitelnost a jednoznačnost (např. v případě práce se dvěmi instancemi formálního modelu najednou), budou komponenty formálního modelu $M = (K_1, K_2, \ldots, K_n)$ dodatečně indexovány. Tedy, $M = (K_{1\,M}, K_{2\,M}, \ldots, K_{n\,M})$.

2.4.2 Automaty

Vedle gramatik je druhou podstatnou oblastí teorie formálních jazyků teorie automatů. Automaty jsou výpočetní stroje, které přijímají věty jazyků místo jejich generování, a proto jsou velmi významné pro praxi.

Definice 2.16. *Konečný automat* je pětice $M_{FA} = (Q, T, R, s, F)$, kde Q je konečná množina stavů, T je konečná vstupní abeceda, $R \subseteq Q \times (T \cup \{\varepsilon\}) \times Q$ je konečná množina pravidel tvaru $pa \to q$, kde $p, q \in Q$ a $a \in T \cup \{\varepsilon\}$, $s \in Q$ je počáteční stav a $F \subseteq Q$ je množina koncových stavů. Při pohledu na konečný automat jako na přepisující systém je úplná abeceda $\Sigma = Q \cup T$.

Konfigurace automatu M je dvojice tvaru (p, w), kde $p \in Q$ a $w \in T^*$. Díky disjunktnosti Q a T můžeme tuto konfiguraci zapisovat pohodlněji jako řetězec pw. Mějme dvě konfigurace u, v z M. M provádí přepisující krok neboli *přechod* z konfigurace $u = (p, aw)$ do konfigurace $v = (q, w)$, kde $p, q \in Q$, $a \in T \cup \{\varepsilon\}$, $w \in T^*$, je-li v R pravidlo $pa \to q$. Tento přechod neboli akceptační krok mezi u a v značíme symbolicky jako $u \Rightarrow v$. Jako u ostatních přepisujících systémů i zde definujme $u \Rightarrow v$, $u \Rightarrow^n v$ s $n \geq 0$ a $u \Rightarrow^* v$, kde $u, v \in \Sigma^*$ (viz definice 2.10). Jestliže $sw \Rightarrow^* f$ v M, kde $w \in T^*$, $s \in Q$, $f \in F$, M přijímá w. Množina všech řetězců, které M přijímá, tvoří jazyk přijímaný automatem M, zapisovaný jako $L(M)$.

Třídu jazyků přijímaných konečnými automaty značme $\mathcal{L}(\mathbf{FA})$.

Vyjadřovací schopnosti tohoto formálního modelu popisuje následující věta převzatá z [48].

Věta 2.1. $\mathcal{L}(\mathbf{FA}) = \mathbf{REG}$.

Kromě inspirace při studiu kombinovaných přepisujících systémů využijeme tento typ automatů pro omezování obsahu konfigurací také v sekci 5.1.4.

Definice 2.17. *Zásobníkový automat* je sedmice $M_{PDA} = (Q, T, \Gamma, R, s, S, F)$, kde Q je konečná množina stavů, $T \subseteq \Gamma$ je konečná vstupní abeceda, Γ je konečná zásobníková abeceda, $R \subseteq \Gamma^* \times Q \times (T \cup \{\varepsilon\}) \times \Gamma^* \times Q$ je konečná množina pravidel, $s \in Q$ je počáteční stav, $S \in \Gamma$ je počáteční symbol na zásobníku a $F \subseteq Q$ je množina koncových stavů. Q, T, Γ jsou po dvojicích navzájem disjunktní. Pravidla zapisujeme ve tvaru $\gamma pa \to yq$, což odpovídá $(\gamma, p, a, y, q) \in R$, kde $\gamma \in \Gamma^*$, $p, q \in Q$, $a \in T \cup \{\varepsilon\}$ a $y \in \Gamma^*$.

V některé literatuře ([13]) se zásobníkový automat podle definice 2.17 nazývá *rozšířený*. Dále se u automatů běžně neuvádí úplná abeceda Σ mezi seznamem komponent, protože se většinou rozděluje na více jak dvě abecedy (např. na Q, T a Γ) (podobně jako třeba u stavových gramatik, viz níže).

Definice 2.18. *Konfigurace zásobníkového automatu* je trojice $(\alpha, p, w) \in \Gamma^* \times Q \times T^*$, kde α je celý obsah zásobníku, p označuje aktuální stav a w zatím nezpracovaný vstupní řetězec. Přechodem, neboli akceptačním krokem, potom nazýváme binární relaci $\Rightarrow$ na $\Gamma^* \times Q \times T^*$, pro kterou platí, že $(z\gamma, p, aw) \Rightarrow (\beta\gamma, q, w)$ právě když existuje v R pravidlo $zpa \to \beta q$, kde $z, \gamma, \beta \in \Gamma^*$, $p, q \in Q$, $a \in T$ a $w \in T^*$.

Aplikaci pravidla zásobníkového automatu na aktuální konfiguraci $(\gamma\beta, p, aw)$ lze interpretovat následovně: Pokud je p aktuální stav, a aktuální vstupní symbol, γ je podřetězec na vrcholu zásobníku a $\gamma pa \to yq \in R$, pak tento zásobníkový automat M_{PDA} může přečíst symbol a, změnit vrchol zásobníku z γ na y a nakonec změnit stav z p na q. Poznamenejme, že pokud je v pravidle tvaru $zpa \to \beta q$, $a = \varepsilon$, tak toto pravidlo ze vstupu nečte žádný symbol.

Standardním způsobem rozšíříme relaci $\Rightarrow$ na $\Rightarrow^n$ pro $n \geq 0$. Pak na základě $\Rightarrow^n$ definujeme $\Rightarrow^+$ a $\Rightarrow^*$ jako tranzitivní a tranzitivně-reflexivní uzávěry relace $\Rightarrow$.

Jazyk přijímaný zásobníkovým automatem M, $L(M)$, definujme jako $L(M) = \{w |\ w \in T^*, (S, s, w) \Rightarrow (\varepsilon, f, \varepsilon),\ f \in F\}$. Dále třídu takovýchto jazyků značme $\mathcal{L}(\mathbf{PDA})$.

2.4.3 Chomského hierarchie jazyků, gramatik a automatů

Pro třídy jazyků definovaných v předchozích definicích platí následující důležitá věta, která zachycuje Chomského hierarchii jazyků.

Věta 2.2 (viz [48])**.** $\mathbf{REG} \subset \mathbf{LIN} \subset \mathbf{CF} \subset \mathbf{CS} \subset \mathbf{RE}$.

Pro připomenutí vztahů mezi základními gramatikami a definovanými automaty doplňme ještě, že

$$\mathbf{REG} = \mathcal{L}(\mathbf{FA}) \subset \mathbf{CF} = \mathcal{L}(\mathbf{PDA}).$$

2.5 Řízené přepisující systémy

V této sekci definujeme pokročilejší formální modely. Hlavní pozornost bude věnována řízeným gramatikám (např. programovaná, stavová a další). Jelikož tyto modely již nebývají běžnou součástí vzdělaní každého informatika, jejich výklad bude koncipován podrobněji než doposud (včetně příkladů s komentářem).

2.5.1 Způsoby řízení

Řízené gramatiky ve většině případech staví na pravě-lineárních nebo nejčastěji bezkontextových gramatikách, u nichž jistým způsobem diktují či omezují aplikovatelnost některých pravidel nad rámec klasické definice výpočetního kroku v Chomského gramatikách:

- použití relace pro provázání předchozích a následných použití pravidel (např. programovaná gramatika, maticová gramatika)
- doplnění gramatiky o prvky automatů jako třeba množinu stavů, které omezují aplikovatelnost daného pravidla jenom na případy, kdy se konfigurace nachází ve správném aktuálním stavu (např. stavová gramatika)
- rozptýlená kontrola kontextu ve větné formě, tzn. na rozdíl od kontextových gramatik provádíme kontrolu napříč celou větnou formou a ne pouze v bezprostředním okolí přepisovaného symbolu (např. gramatiky s nahodilým kontextem nebo s rozptýleným kontextem)

- paralelní verze přepisování v konfiguraci (nejen přepis všech aktivních symbolů, ale například i přepis všech stejných symbolů v konfiguraci apod.; například L-systémy, čisté gramatiky, indické gramatiky)
- v některých případech lze i omezování formálních modelů chápat jako řízení, což diskutuje celá kapitola 3

2.5.2 Řízené gramatiky

Při popisu nebezkontextových problémů si většinou vystačíme i s modely jednoduššími než jsou kontextové gramatiky, nebo dokonce Turingovy stroje. Proto byly v 70. letech minulého století hojně zaváděny různé modifikace bezkontextových gramatik, které si kladly za cíl právě zvýšit mocnost formálního modelu, a přesto zbytečně nezkomplikovat jeho pravidla ani jejich výběr pro aplikování. Na několika typech modifikovaných gramatik, které jednotně označujeme jako *řízené gramatiky*, staví i tato práce.

Ve všech případech (pokud není řečeno jinak) uvažujeme řízené gramatiky založené na pravidlech bezkontextového tvaru, takže jádro takového pravidla obsahuje na levé straně právě jeden neterminál a na straně pravé libovolný řetězec. Speciálním případem jsou pak modely, kdy zakazujeme na pravé straně pravidla prázdný řetězec. Jedná se o formální modely bez *vymazávajících pravidel*.

Konvence 2.9. Kromě základních tříd jazyků (např. z Chomského hierarchie) budeme všechny ostatní třídy jazyků (většinou definované řízenými a/nebo omezovanými formálními modely) značit podle následujícího schématu

$$\mathcal{L}_X(\mathbf{Y}, Z),$$

kde X je způsob omezování (například konstanta udávající konečný index; index X může být zapsán i zleva), Y udává základní formální model (například CF pro bezkontextovou gramatiku) a nakonec Z specifikuje dodatečné omezení (například typ omezujícího jazyka, kontrola výskytu apod.).

Je-li Y základní formální model, pak $Y - \varepsilon$ značí ten samý formální model, ale bez vymazávajících pravidel (pokud samozřejmě má pro daný model smysl uvažovat definici vymazávajících pravidel).

Programované gramatiky

Programovaná gramatika (z roku 1969, viz [62]) specifikuje dodatečné podmínky kladené na úspěšné provedení derivačních kroků, kde diktuje množinu povolených následných pravidel po aplikaci určitého pravidla. V případě nemožnosti aplikovat žádné pravidlo z předepsané podmnožiny se uchyluje k využití některého pravidla v zotavovací množině pravidel.

Definice 2.19. *Programovaná gramatika* (viz strana 28 v [12]) je čtveřice $G = (\Sigma, T, P, S)$, kde všechny komponenty mají stejný význam jako u bezkontextové gramatiky. Všechna pravidla jsou tvaru:

$$\lfloor r\colon A \ \rightarrow x, R, F \rfloor,$$

kde $A \in \Sigma - T$, $x \in \Sigma^*$, $R, F \subseteq \mathrm{Lab}(P)$, $q\colon A \rightarrow v$ je bezkontextové jádro pravidla, R nazýváme množinou následných pravidel (*success field*) a F zotavovací množinou pravidla r (*failure field*). Pokud je alespoň u jednoho pravidla zotavovací množina neprázdná, mluvíme o tzv. *programované gramatice s kontrolou výskytu* (*appereance checking*). V opačném případě místo neustálého psaní prázdné množiny množinu F ze zápisu pravidel vynecháváme.

Derivační krok při použití pravidla $\lfloor q\colon A \to v, R, F \rfloor$ gramatiky G je analogický derivačnímu kroku v bezkontextové gramatice (definice 2.13). Navíc je ale potřeba kontrolovat řízení pomocí množin R a F.

R označuje množinu pravidel (resp. návěští pravidel), z nichž lze vybírat následující pravidlo k aplikaci v G.

V situaci, kdy není aplikovatelné žádné pravidlo z R, zůstane větná forma nezměněna a G bude pokračovat libovolným pravidlem, jehož návěští je v F.

Standardním způsobem definujme $\Rightarrow_G^m$, kde $m \geq 0$, $\Rightarrow_G^+$ a $\Rightarrow_G^*$. Jazyk generovaný programovanou gramatikou G, $L(G)$ je definován jako $L(G) = \{w \in T^* |\ S \Rightarrow_G^* w\}$.

Příklad 2.1. Uvažujme následující programovanou gramatiku bez kontroly výskytu, která bude generovat kontextový jazyk, čímž budeme demonstrovat zvýšení vyjadřovacích schopností tohoto formálního modelu oproti bezkontextovým gramatikám.

Mějme $G = (\Sigma, T, P, S)$, kde úplná abeceda obsahuje všechny v pravidlech použité symboly, tj. $\Sigma = \{S, A, B, C, a, b, c\}$; terminální symboly podle konvence píšeme malými písmeny, a jsou to $T = \{a, b, c\}$; počátečním neterminálem je S; množina pravidel P obsahuje tyto pravidla předepsaného tvaru:

1: $S \to ABC, \{2, 5\}$
2: $A \to aA, \{3\}$
3: $B \to bB, \{4\}$
4: $C \to cC, \{2, 5\}$
5: $A \to a, \{6\}$
6: $B \to b, \{7\}$
7: $C \to c, \{7\}$

Ve složených závorkách u každého pravidla je určena množina návěští pravidel, která mohou být použita po aplikaci daného pravidla. Například pokud aplikuje první pravidlo, tak jako další krok je možné aplikovat pouze druhé nebo páté pravidlo a žádné jiné.

Například větu *aabbcc* generuje následující derivace (derivační posloupnost) $S \Rightarrow ABC$ [1] $\Rightarrow aABC$ [2] $\Rightarrow aAbBC$ [3] $\Rightarrow aAbBcC$ [4] $\Rightarrow aabBcC$ [5] $\Rightarrow aabbcC$ [6] $\Rightarrow aabbcc$ [7]. Po zamyšlení se nad propojením jednotlivých pravidel vidíme, že pravidla 2, 3, 4 a 5, 6, 7 jsou prováděna ve skupinách a pouze po aplikaci pravidla 1 a 4 se provádí rozhodnutí, se kterou z těchto skupin pokračovat. Jazyk generovaný touto gramatikou G je $L(G) = \{a^n b^n c^n |\ n \geq 1\}$.

Příklad 2.2. Nyní si ukažme složitější programovanou gramatiku, která má sice méně pravidel, ale využívá navíc kontrolu výskytu, tedy obsahuje u některých pravidel neprázdnou zotavovací množinu.

$G = (\{S, A, a\}, \{a\}, P, S)$, kde množina pravidel úplného tvaru (včetně zotavovacích množin) je následující:

1: $S \to AA, \{1\}, \{2, 3\}$
2: $A \to S, \{2\}, \{1\}$
3: $A \to a, \{3\}, \emptyset$

Rozbor této programované gramatiky s kontrolou výskytu bude demonstrovat, že generuje kontextový jazyk $L(G) = \left\{a^{2^n} |\ n \geq 1\right\}$. Začátek je dán počátečním neterminálem. Hned po aplikaci prvního pravidla ovšem zjistíme, že znova použít první pravidlo, tak jak to máme předepsáno, nelze. Nyní přichází do hry zotavovací množina, protože jsme se ocitli ve slepé uličce. Větná forma zůstane nezměněna a následující pravidlo pro další aplikaci se vybírá ze zotavovací množiny prvního pravidla mezi 2 a 3. Toto rozhodnutí stanoví, zda pokračovat v generování symbolů a, nebo derivaci

ukončit a všechny neterminály přepsat na terminály.

$$S \Rightarrow AA\ [1]\ \Rightarrow^2\ SS\ [22]\ \Rightarrow\ SAA\ [1]\ \Rightarrow\ AAAA\ [1]\ \Rightarrow\ AaAA\ [3]\ \Rightarrow^3\ aaaa\ [333]$$

Opakovaná aplikace druhého pravidla pouze přejmenuje všechny neterminály A na S. Z definice programované gramatiky plyne, že druhé pravidlo se tomuto úkolu nemůže vyhnout přes zotavovací množinu, dokud existuje nějaké A k přejmenování.

Poslední pravidlo pouze vytváří řetězec terminálů z vygenerovaných neterminálů, a proto má prázdnou zotavovací množinu.

Třídy jazyků definovaných programovanými gramatikami, programovanými gramatikami bez vymazávajících pravidel, programovanými gramatikami s kontrolou výskytu, resp. programovanými gramatikami s kontrolou výskytu a bez vymazávajících pravidel značíme $\mathcal{L}(\mathbf{P})$, $\mathcal{L}(\mathbf{P}, CF - \varepsilon)$, $\mathcal{L}(\mathbf{P}, CF, ac)$, resp. $\mathcal{L}(\mathbf{P}, CF - \varepsilon, ac)$.

Gramatiky s rozptýleným kontextem

Kontextové gramatiky podle definice 2.12 kontrolují bezprostřední okolí přepisovaného symbolu. Rozptýlený kontext, jak plyne z názvu, však sousednost s přepisovaným symbolem nevyžaduje. Kontrolované kontextové symboly se musí vyskytovat kdekoli ve větné formě (u zakazujícího kontextu se naopak nesmí kontextové symboly vyskytovat nikde ve větné formě). Existují i zobecněné varianty, které místo kontextových symbolů připouští celé řetězce.

Definice 2.20. *Gramatika s nahodilým kontextem* ([74] nebo viz strana 30 v [12]) je čtveřice $G = (\Sigma, T, P, S)$, kde významy komponent jsou opět totožné s bezkontextovou gramatikou a všechna pravidla jsou tvaru:

$$\lfloor q\colon A\ \rightarrow x, P, F \rfloor,$$

kde $A \in \Sigma - T$, $x \in \Sigma^*$, $P, F \subseteq (\Sigma - T)$, P nazýváme povolující množina neterminálů (*permitting set*) a F zakazující množina neterminálů (*forbidding set*). Aplikace pravidla q na větnou formu se provede, jestliže výchozí větná forma obsahuje všechny neterminály z množiny P a žádný neterminál z množiny F. Samotný derivační krok je proveden analogicky s bezkontextovou gramatikou. V případě, že alespoň jedno pravidlo má neprázdnou zakazující množinu, mluvíme o gramatice s nahodilým kontextem s kontrolou výskytu. V opačném případě gramatiku označujeme jako gramatiku s nahodilým kontextem.

Příklad 2.3. Ukažme si gramatiku s nahodilým kontextem $G = (\Sigma, T, P, S)$ pro jazyk $L(G) = \{a^n b^n c^n |\ n \geq 1\}$, kde pravidla jsou:

$\lfloor 1\colon S \rightarrow ABC, \emptyset, \emptyset \rfloor$
$\lfloor 2\colon A \rightarrow aA', \{B\}, \emptyset \rfloor$, $\lfloor 3\colon B \rightarrow bB', \{C\}, \emptyset \rfloor$, $\lfloor 4\colon C \rightarrow cC', \{A'\}, \emptyset \rfloor$
$\lfloor 5\colon A' \rightarrow A, \{B'\}, \emptyset \rfloor$, $\lfloor 6\colon B' \rightarrow B, \{C'\}, \emptyset \rfloor$, $\lfloor 7\colon C' \rightarrow C, \{A\}, \emptyset \rfloor$
$\lfloor 8\colon A \rightarrow a, \{B\}, \emptyset \rfloor$, $\lfloor 9\colon B \rightarrow b, \{C\}, \emptyset \rfloor$, $\lfloor 10\colon C \rightarrow c, \emptyset, \emptyset \rfloor$.

Třídy jazyků definované gramatikami s rozptýleným kontextem značíme analogicky programovaným gramatikám.

Stavové gramatiky

Stavové gramatiky lze řadit k nejstarším snahám o kombinování gramatik a automatů. V podstatě se jedná o bezkontextovou gramatiku obohacenou o konečně-stavové řízení převzaté z konečných automatů.

Definice 2.21. *Stavová gramatika* (viz [21]) je pětice $G = (V, W, T, P, S)$, kde V je abeceda terminálních a neterminálních symbolů, T, S mají stejný význam jako u bezkontextové gramatiky (viz definice 2.13), W je konečná množina *stavů* a P je konečná podmnožina relace $(W \times (V - T)) \times (W \times V^+)$.

Místo čtveřice $(q, A, p, v) \in P$ zapisujeme pravidla ve tvaru $(q, A) \to (p, v) \in P$. Pro každý řetězec $z \in V^*$ definujme množinu ${}_G states(z) = \{q \mid (q, B) \to (p, v) \in P$, kde $B \in (V-T) \cap \text{alph}(z)$, $v \in V^+$, $q, p \in W\}$. Je-li $r\colon (q, A) \to (p, v) \in P$, $x, y \in V^*$, ${}_G states(x) = \emptyset$, pak G může provést *derivační krok* z (q, xAy) do (p, xvy), symbolicky zapisujeme jako $(q, xAy) \Rightarrow (p, xvy)$ $[r]$ v G. Pokud navíc existuje pozitivní celé číslo n splňující podmínku $\text{occur}(V - T, xA) \leq n$, říkáme, že $(q, xAy) \Rightarrow (p, xvy)$ $[r]$ je derivace *n-limitovaná*; píšeme $(q, xAy)\ {}_n\Rightarrow\ (p, xvy)$ $[r]$.

Jazyk generovaný gramatikou G, $L(G)$ definujeme jako $L(G) = \{w \in T^* \mid (q, S) \Rightarrow^* (p, w)$, $q, p \in W\}$. $\mathcal{L}(\mathbf{ST})$ označuje třídu všech jazyků definovaných stavovými gramatikami.

Dále pro každé $n \geq 1$ definujeme *n-limitovaný jazyk* daný gramatikou G, $L(G, n) = \{w \in T^* \mid (q, S)\ {}_n\Rightarrow^*\ (p, w)$, $q, p \in W\}$. Třídu těchto jazyků značme ${}_n\mathcal{L}(\mathbf{ST}) = \{L \mid L = L(G, k)$, $1 \leq k \leq n$, G je stavová gramatika$\}$, kde $n \geq 1$. Dále ${}_\infty\mathcal{L}(\mathbf{ST}) = \bigcup_{n \geq 1}^{\infty} {}_n\mathcal{L}(\mathbf{ST})$.

Všimněme si definice tvaru pravidel, která u stavových gramatik vylučuje vymazávající pravidla. Úplná abeceda v tomto systému obsahuje i stavy, $\Sigma = V \cup T \cup W$.

Neformálně popsáno, stavová gramatika využívá jeden řídící a jeden omezující mechanismus:

1. řízení pomocí konečně-stavového řízení: Každé pravidlo je obohaceno o výchozí a cílový stav, které, podobně jako např. u konečných automatů, diktují v jakém aktuálním stavu je pravidlo aplikovatelné a do jakého cílového stavu se jeho aplikací systém dostane. Proto je také nutné rozšířit konfiguraci tohoto modelu z pouhé větné formy na dvojici obsahující aktuální stav a aktuální větnou formu (tzv. *větná komponenta konfigurace*).

2. omezení na aplikaci nejlevějšího možného přepisu: Přímo definice stavové gramatiky zajišťuje, že v aktuálním stavu musí být aplikován přepis neterminálu nejvíce vlevo ve větné komponentě konfigurace. Přepisovaný neterminál však nemusí být nutně nejlevější, ale dostačuje, pokud pro žádný jiný neterminál vyskytující se nalevo od neterminálu přepisovaného neexistuje pro aktuální stav přepisující pravidlo (matematicky zajištěno podmínkou ${}_G states(x) = \emptyset$ pro výchozí konfiguraci (q, xAy) každého výpočetního kroku.

V každé konfiguraci jsou tedy obsaženy dva aktivní symboly (stav a nejlevější přepsatelný neterminál) a libovolný počet pasivních a potenciálně-aktivních symbolů.

Příklad 2.4. Nejprve si ukažme klasický nebezkontextový jazyk $\{a^n b^n c^n \mid n \geq 1\}$ generovaný pomocí stavové gramatiky $G = (V, W, T, P, S)$, kde komponenty V, W, T jsou lehce odvoditelné z množiny pravidel P:

1: $(s, S) \to (s, AC)$
2: $(s, A) \to (p, aAb)$
3: $(p, C) \to (s, cC)$
4: $(s, A) \to (q, ab)$
5: $(q, C) \to (q, c)$

Jednoduchá ukázka derivace ve stavové gramatice, která je 2-limitovaná, což znamená, že nikdy nepracuje více jak s druhým neterminálem zleva, následuje:

$$\begin{array}{lcll} (s,S) & \Rightarrow & (s,AC) & [1] \\ & \Rightarrow & (p,aAbC) & [2] \\ & \Rightarrow & (s,aAbcC) & [3] \\ & \Rightarrow & (q,aabbcC) & [4] \\ & \Rightarrow & (q,aabbcc) & [5] \end{array}$$

Příklad 2.5. Další příklad stavové gramatiky ukazuje trochu jiný jazyk $\{ww|\ w \in T^*\}, T \in \{a,b\}$, který však také není bezkontextový. Uvádíme již pouze výčet pravidel:

1: $(s,S) \rightarrow (p,AB)$
2: $(s,A) \rightarrow (p,aA)$
3: $(p,B) \rightarrow (s,aB)$
4: $(s,A) \rightarrow (p,bA)$
5: $(q,B) \rightarrow (s,bB)$
6: $(s,A) \rightarrow (f,a)$
7: $(f,B) \rightarrow (s,a)$
8: $(s,A) \rightarrow (g,b)$
9: $(g,B) \rightarrow (s,b)$

Funkčnost gramatiky demonstrujme na generování řetězce *abbabb*: $(s,S) \Rightarrow (s,AB)$ [1] $\Rightarrow (p,aAB)$ [2] $\Rightarrow (s,aAaB)$ [3] $\Rightarrow (q,abAaB)$ [4] $\Rightarrow (s,abAabB)$ [5] $\Rightarrow (g,abbabB)$ [8] $\Rightarrow (s,abbabb)$ [9]. Princip se velmi podobá předchozímu příkladu 2.4, protože i zde si pomocí stavů zajišťujeme provedení jisté sekvence kroků, kde každý krok je prováděn na jiném místě větné komponenty konfigurace. Dále si pomocí stavů také pamatujeme informaci, který symbol byl již vygenerován v prvním podřetězci w, abychom stejný symbol mohli vygenerovat i v druhém podřetězci w, a tím zaručili jejich shodnost.

Připomeňme, že Kasai ([21]) ve svém článku dokázal důležité věty týkající se stavových gramatik. Nyní si je (bez důkazů) pro lepší pochopení následných výsledků uvedeme.

Věta 2.3. $\mathcal{L}(\mathbf{ST}) = \mathbf{CS}$.

Důsledek 2.4. ${}_{\infty}\mathcal{L}(\mathbf{ST}) \subset \mathcal{L}(\mathbf{ST})$.

Všimněme si, že pro každé $n \geq 1$, ${}_{n}\mathcal{L}(\mathbf{ST}) \subseteq {}_{n+1}\mathcal{L}(\mathbf{ST})$, což plyne z definice stavové gramatiky.

Věta 2.5. *Pro všechna* $n \geq 1$, ${}_{n}\mathcal{L}(\mathbf{ST}) \subset {}_{n+1}\mathcal{L}(\mathbf{ST})$.

m-paralelní n-pravě-lineární jednoduché maticové gramatiky

Již od 70. let 20. století se v teorii formálních jazyků studuje povaha a vlastnosti paralelismu. Kolem roku 1975 byl zaveden nový přepisující systém kombinující paralelní a řízené paralelní přepisování: (1) paralelismus známý například z n-paralelních pravě-lineárních gramatik ([60], [61]) nebo z L-systémů ([63]) a (2) jednoduché maticové gramatiky ([18]).

Tento systém pak přepisuje v jednom kroku $m \cdot n$ neterminálů tak, že paralelně aplikuje m n-pravě-lineárních jednoduchých matic, kde matice je posloupnost pravidel aplikovaných atomicky v jediném přepisujícím kroku.

Definice 2.22. Pro $m, n \geq 1$, *m-paralelní n-pravě-lineární jednoduchá maticová gramatika* (m-*parallel n-right-linear simple matrix grammar*) (zkráceně m-*Pn*-G, viz [77]) je $(mn+3)$-tice

$$G = (N_{11}, \ldots, N_{1n}, \ldots, N_{m1}, \ldots, N_{mn}, T, S, P),$$

kde

N_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ jsou po dvojicích vzájemně disjunktní *abecedy neterminálů*,

T je *abeceda terminálů*,

$S \notin N \cup T$ je počáteční symbol, kde $N = N_{11} \cup \cdots \cup N_{mn}$, a

P je konečná množina *maticových pravidel*.

Maticové pravidlo může mít jednu z následujících tří forem:

(i) $[S \to X_{11} \ldots X_{mn}]$, $X_{ij} \in N_{ij}$, $1 \leq i \leq m$, $1 \leq j \leq n$,

(ii) $[X_{i1} \to \alpha_{i1}, \ldots, X_{in} \to \alpha_{in}]$, $X_{ij} \in N_{ij}$, $\alpha_{ij} \in T^*$, $1 \leq j \leq n$, pro nějaké i, $1 \leq i \leq m$, a

(iii) $[X_{i1} \to \alpha_{i1} Y_{i1}, \ldots, X_{in} \to \alpha_{in} Y_{in}]$, $X_{ij}, Y_{ij} \in N_{ij}$, $\alpha_{ij} \in T^*$, $1 \leq j \leq n$, pro nějaké i, $1 \leq i \leq m$.

Derivační krok pro m-*Pn*-G je definován následovně:
Pro $x, y \in (N \cup T \cup \{S\})^*$ a m-*Pn*-G G, $x \Rightarrow y$, když a jen když

(A) buď $x = S$ a $[S \to y] \in P$,

(B) nebo $x = y_{11} X_{11} \ldots y_{mn} X_{mn}$, $y = y_{11} x_{11} \ldots y_{mn} x_{mn}$, kde $y_{ij} \in T^*$, $X_{ij} \in N_{ij}$, $1 \leq i \leq m$, $1 \leq j \leq n$, a $[X_{i1} \to x_{i1}, \ldots, X_{in} \to x_{in}] \in P$, $1 \leq i \leq m$.

Je-li $x, y \in (N \cup T \cup \{S\})^*$ a $m \geq 0$, pak $x \Rightarrow^m y$, když a pravě když existuje posloupnost $x_0 \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_m$, $x_0 = x$, $x_m = y$. Pak říkáme, že $x \Rightarrow^+ y$, když a jen když existuje $m > 0$ takové, že $x \Rightarrow^m y$ a $x \Rightarrow^* y$, když a jen když buď $x = y$, nebo $x \Rightarrow^+ y$.

Alternativně definujme obvyklým způsobem tranzitivní $\Rightarrow^+$ a reflexivně-tranzitivní uzávěr $\Rightarrow^*$ relace přímé derivace $\Rightarrow$.

Jazyk generovaný m-*Pn*-G G označujeme $L(G)$ a definujeme jej jako $L(G) = \{x \mid S \Rightarrow^* x, x \in T^*\}$. Jazyk $L \subseteq T^*$ je *m-paralelní n-pravě-lineární jednoduchý maticový jazyk* (m-*Pn*-L), když a jen když existuje m-*Pn*-G G taková, že $L = L(G)$. Třídu jazyků definovaných m-*Pn*-L budeme značit $\mathcal{R}^m_{[n]}$.

Položíme-li $n = 1$, dostáváme *m-paralelní pravě-lineární gramatiku*. Naopak položíme-li $m = 1$, získáme *n-pravě-lineární jednoduchou maticovou gramatiku*, která nás bude v kapitole 5 zajímat nejvíce.

Kapitola 3

Omezování konfigurací

V této kapitole nejprve shrneme různé druhy omezování formálních modelů, resp. přepisujících systémů, a v její další části se zaměříme na omezování konfigurací, které tvoří stěžejní téma této práce.

Jednou z nejpočetnějších oblastí omezování formálních modelů jsou řízené gramatiky a automaty. Již kolem 70. let minulého století byly zaváděny první řízené gramatiky ([21, 62, 74, 77]). Jejich hlavní myšlenkou je, že staví na jednoduché bezkontextové gramatice, již vhodným způsobem doplňují o různé řídící mechanismy, které jistým způsobem řídí aplikaci pravidel. V zásadě lze místo řízení také hovořit o omezování, protože daný řídící mechanismus omezuje původní volnost formálního modelu.

Řízené automaty se začaly studovat mnohem později a v menší míře než gramatiky a lze říci, že větší pozornosti se jim dostává až v poslední době ([36, 23, 50, 52]).

3.1 Způsoby omezování přepisujících systémů

Klasifikace omezování (potažmo i řízení) přepisujících systémů nemá žádnou všeobecně uznávanou podobu, a tak nelze prohlašovat předkládané rozdělení jako ultimativní. Inspirujme se například v [12, 53].

Proveďme základní klasifikaci omezování formálních modelů s důrazem na přepisující systémy:

1. statické (omezování komponent přepisujících systémů; vztah k popisné složitosti)
2. dynamické (omezování výpočtu/zpracování věty; vztah k časové a prostorové složitosti)
   a) omezování konfigurací
   b) omezování posloupnosti aplikovaných pravidel
3. hybridní (nejasné rozhraní mezi statickým a dynamickým omezováním)

Při hlubším zamyšlení se nad konkrétními typy omezování přepisujících systémů dojdeme k závěru, že kvůli silné provázanosti popisu modelu a jeho funkčnosti (způsobu práce) lze provádět mezi statickou a dynamickou charakterizací omezování transformace. Tyto transformované popisy omezování pak ale působí velmi těžkopádně a jsou hůře pochopitelné, takže budeme možnost převodu mezi jednotlivými způsoby omezování ignorovat. V případech, kdy bude způsob omezení na rozhraní mezi dynamickým a statickým omezováním, budeme mluvit o hybridním omezování (nebude-li to nezbytně nutné, budeme se snažit tomuto pojmu vyhýbat).

Uveďme například povolení/zákaz vymazávajících pravidel, který by mohl patřit jak do statického omezování, tak do dynamického, kdy klademe omezení na monotónnost posloupnosti délek konfigurací při zpracování libovolné věty.

Závěrem je, že tato klasifikace je závislá na způsobu definování zkoumaného omezování přepisujícího systému. My se budeme snažit pracovat výhradně s definicemi, které jsou již v teorii formálních jazyků ustálené.

Poznamenejme, že popis omezování přepisujících systémů lze snad ve všech případech transformovat na jistý typ omezujícího jazyka (ten omezuje buď posloupnost pravidel, tvary konfigurací či jiné parametry přepisujícího systému) (viz např. pozorování 3.1).

Protože, obzvlášť v některých speciálních případech, je tato unifikace způsobu omezování značně těžkopádná, pokusíme se o podrobnější klasifikaci omezování posloupností pravidel i konfigurací v následujících dvou sekcích.

3.2 Typy omezování posloupnosti aplikovaných pravidel

U tohoto typu omezování klademe dodatečné podmínky (většinou k jednotlivým pravidlům) na aplikovatelnost konkrétních pravidel. Vzhledem k obrovskému množství existujících (a stále vznikajících) řízených přepisujících systémů si nemůžeme troufnout tuto klasifikaci uzavřít. Mezi nejčastější přístupy patří omezování:

- řízené tzv. řídícím jazykem (např. řízené gramatiky a řízené automaty, viz [49] a [23]);
- závislost na předchozích aplikovaných pravidlech (maticové, programované gramatiky);
- konečně-stavové řízení (stavové gramatiky, #-přepisující systémy);
- výskytem nebo absencí podřetězců (či symbolů) v přepisovaných konfiguracích (např. gramatiky s rozptýleným kontextem, (zobecněné) podmínkové gramatiky);
- vymezením pouze části aktuální nebo libovolné konfigurace, kde povolíme/zakážeme přepisování (např. n-limitovanost, kterou si blíže představíme níže);
- a další.

3.2.1 n-limitovanost

Tento typ omezování přechodů mezi jednotlivými konfiguracemi klade požadavek na práci s aktivními symboly, jež se vyskytují mezi prvními n proměnnými v řetězci, který je j-tou komponentou konfigurace ξ formálního modelu, což je vyjádřeno označením component(ξ, j), kde $j \in \{1, 2, \ldots m\}$ a m je počet komponent konfigurace (konfigurace je m-tice). Odtud plyne požadavek, aby přepisovací systém s tímto omezením pracoval s proměnnými (např. neterminály nebo nevstupními zásobníkovými symboly).

n-limitovanost je tedy omezování způsobu vybírání použitého pravidla tak, aby se vždy přepsal jeden z n nejlevějších výskytů libovolných symbolů, které jsou obecně v konfiguraci přepsatelné. Většinou se toto omezení aplikuje na potenciálně nekonečnou komponentu konfigurace (např. větnou komponentu konfigurace nebo zásobník).

Definice 3.1. Mějme přepisující systém $H = (\Sigma, R)$. Označme konečnou množinu proměnných $N \subseteq \Sigma$ a component$(w, j) \in \Sigma^*$, kde j je konstanta vyjadřující pro daný přepisující systém pořadí významné komponenty konfigurace, na kterou omezení n-limitovanosti aplikujeme.

Přechod mezi dvěma konfiguracemi ξ_1 a ξ_2, component$(\xi_1, j) = uAz \Rightarrow uvz =$ component(ξ_2, j) nazveme n-limitovaný, pokud occur$(N, u) \leq n - 1$ a A je aktivní symbol. Posloupnost přechodů je pak n-limitovaná, je-li každý její přechod n-limitovaný. Jazyk, jehož všechny věty mohou vzniknout n-limitovanou posloupností přechodů, nazveme n-limitovaný.

Nechť X je formální model, pak pro každé $n \geq 1$ je ${}_n\mathcal{L}(\mathbf{X})$ třída n-limitovaných jazyků definovaných modelem X.

Pozorování 3.1. Ukažme si, jak lze jednoduše vyjádřit například n-limitovanost v pojmech omezování posloupností pravidel i konfigurací:

a) n-limitovanost znamená, že povolíme přepisování pouze omezeného počtu výskytů proměnných zleva v konfiguraci nebo v její části;

b) n-limitovanost znamená, že v konfiguraci omezíme počet výskytů aktivních symbolů jistou konstantou a přidáme požadavek na jejich výskyt nejlevěji v konfiguraci (či její části) bez ohledu na pasivní symboly (nalevo od posledního aktivního symbolu se však nesmí vyskytovat žádný potenciálně-aktivní symbol).

Jedním z nejdůležitějších praktických využití omezení n-limitovanosti v oblasti gramatik je ekvivalence třídy bezkontextových jazyků a třídy bezkontextových jazyků generovaných bezkontextovou gramatikou pracující nejlevějším (levým, kanonickým) způsobem a znamená to, že bezkontextová gramatika může pracovat nejlevějším způsobem, tj. 1-limitovaně, při derivování věty jazyka bez ovlivnění generativní síly.

Navíc lze n-limitovanost používat jako zobecnění pojmu levé derivace (někdy nazývaná kanonická), a to i pro svět automatů (viz definice 4.8). Kanoničnost přepisování některých systémů budeme podrobněji studovat v kapitole 6.

Konvence 3.1. Jelikož mnoho z definovaných pojmů pracuje s pojmy jako n-tý symbol zleva a podobně a jelikož všechny tyto pojmy lze zobecnit i pro definici z pravé strany, budeme možnost zprava opomínat v případě, že bychom získali naprosto totožné či analogické vlastnosti. Tato konvence se týká pojmů jako n-limitovanost, levá derivace, i-tý symbol v řetězci atd.

3.3 Typy omezování konfigurací

Pro omezování konfigurací je využití některého omezujícího jazyka více přirozené než u posloupností pravidel:

- Omezující jazyk může být:
  - konečný (V případě aplikace omezení na celou konfiguraci (ne pouze na některou část) se jedná o velmi restriktivní omezení vedoucí k rapidní degradaci vyjadřovací síly většiny systémů, až na třídu konečných jazyků.);
  - nekonečný (pro zachování elegance a efektivity omezování by omezující jazyk neměl být příliš komplikovaný, např. pravě-lineární, lineární nebo nejvýše bezkontextový).

Přesto další rozdělení omezování konfigurací může mít následující podobu:

- omezování počtu výskytů některých symbolů, např.: množiny (nebo jejich libovolná kombinace, viz koncept 2.4)
  - pasivních symbolů (značně netradiční varianta)
  - proměnných (obecně aktivní dohromady s potencionálně-aktivními symboly; nejdůležitější varianta vedoucí na omezení nazývané konečný index (viz definice 3.2))
  - aktivních symbolů
  - potencionálně-aktivních symbolů
- jiné dělení omezování podle počtu výskytů symbolů je na:
  - konečné (většinou zadané konstantou)
  - nekonečné (omezování nějakou funkcí většinou závislou na délce konfigurace nebo délce věty, např. omezování pracovního prostoru viz definice 3.3)

Díky závislosti omezování konfigurací, mimo jiné také na množině přepisujících pravidel, lze aplikovat dva způsoby zajištění omezování konfigurací (obecně ortogonální k předchozí klasifikaci):

- implicitní omezování konfigurací — ze samotné definice formálního modelu nebo z množiny pravidel modelu plyne, že se při validních aplikacích pravidel nemůže vyskytnout konfigurace, která by po přepisu z počáteční konfigurace danému omezení nevyhovovala (např. #-přepisující systém indexu k s vhodně stanovenou množinou pravidel, viz definice 4.1);
- explicitní omezování konfigurací — přepisující systém umožňuje porušit omezení, a je tedy nutné dodržování omezení kontrolovat při každém kroku do nové konfigurace a v případě, že by došlo k porušení omezení v cílové konfiguraci, krok nepovolit (např. bezkontextová gramatika konečného index k).

V následujících dvou podsekcích definujme dvě významné omezování konfigurací: konečný index a omezený pracovní prostor.

3.3.1 Konečný index

Konečný index je omezující mechanismus zavedený v 70. letech minulého století, jenž byl důkladně studován při aplikování na mnoho různých formálních modelů: bezkontextovou gramatikou počínaje a řízenými přepisujícími systémy všeho druhu konče.

Neformálně řečeno nám konečný index konstantou k, $k \in \mathbb{N}$, omezuje počet výskytů proměnných v konfiguraci. Pro gramatiky to například znamená, že větná část konfigurace nebude v žádném kroku obsahovat více než k neterminálů. Následuje zobecněná definice pro přepisující systém (založná na kapitole o konečném indexu v [12]).

Definice 3.2. Nechť $H = (\Sigma, R)$ je přepisující systém, $N \subseteq \Sigma$ abeceda proměnných (např. neterminálů nebo nevstupních symbolů), $T \subseteq \Sigma$ abeceda pasivních symbolů (např. terminálů nebo vstupních symbolů) a σ počáteční konfigurace systému H. Mějme posloupnost přechodů mezi konfiguracemi $D\colon \sigma = w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_r = w$, $r \geq 1$ v H, kde komponenta ve w určující systémem definovaný řetězec (větu jazyka) je rovna řetězci pasivních symbolů. Tedy $\text{component}(w, j) \in T^*$, kde j je konstanta vyjadřující pro H pořadí dané komponenty konfigurace, tj. značí významnou komponentu konfigurace, nad kterou omezování na konečný index aplikujeme. Definujeme $\text{Ind}(D, H) = \max(\{\text{occur}(N, \text{component}(w_i, j)) \mid 1 \leq i \leq r\})$. Pro $\text{component}(w, j) \in$

T^* definujme $\mathrm{Ind}(w, H) = \min(\{\mathrm{Ind}(D, H) \mid D$ je posloupnost definující řetězec w systémem $H\})$. *Index přepisujícího systému* H značme $\mathrm{Ind}(H) = \sup(\{\mathrm{Ind}(w, H) \mid \mathrm{component}(w, j) \in L(H)\})$. Pro jazyk L ve třídě jazyků $\mathcal{L}(\mathbf{X})$ generované přepisujícím systémem nějakého typu X, pak definujeme $\mathrm{Ind}_X(L) = \inf(\{\mathrm{Ind}(H) \mid L(H) = L,\ H$ je typu $X\})$. Nechť $\mathcal{L}(\mathbf{X})$ je třída jazyků, pak označme $\mathcal{L}_k(\mathbf{X}) = \{L \mid L \in \mathcal{L}(\mathbf{X}), \mathrm{Ind}_X(L) \leq k\ \}$ pro libovolné $k \geq 1$ a $\mathcal{L}_{fin}(\mathbf{X}) = \bigcup_{n \geq 1} \mathcal{L}_k(\mathbf{X})$.

Definice uvedená výše je někdy označována jako *slabá*, protože vyžaduje omezení pouze té nejekonomičtější cesty k větě. Po zpřísnění podmínky dostáváme tzv. *silnou* (striktní) variantu konečného indexu, kdy požadujeme omezení každé možné konfigurace dosažitelné z počáteční konfigurace.

Konvence 3.2. Mluvíme-li o konkrétním konečném indexu k, tak slovo „konečném“ vynecháváme. Tedy například přepisující systém je indexu 3.

Fakt, že zavedení omezení na konečný index způsobuje podstatnou ztrátu generativní síly, potvrzuje následující dvojice rovnic (3.1) a (3.2) (převzato z [12]), která porovnává vztahy mezi řízenými gramatikami bez a s konečným indexem.

$$\mathbf{CF} \subset \mathcal{L}(\mathbf{RC}) \subseteq \mathcal{L}(\mathbf{P}) \subset \mathcal{L}(\mathbf{P}, CF - \varepsilon, ac) \subseteq \mathbf{CS} \subset \mathcal{L}(\mathbf{P}, CF, ac) = \mathbf{RE} \tag{3.1}$$

$$\mathcal{L}_{fin}(\mathbf{RC}) = \mathcal{L}_{fin}(\mathbf{P}) = \mathcal{L}_{fin}(\mathbf{P}, CF - \varepsilon, ac) = \mathcal{L}_{fin}(\mathbf{P}, CF, ac) \tag{3.2}$$

Žádná z tříd uvedená v rovnici (3.2) navíc není porovnatelná s třídou kontextových jazyků bez konečného indexu, protože platí $\{a^n b^n c^n \mid n \geq 1\} \in \mathcal{L}_{fin}(\mathbf{P})$ a $\mathcal{L}(\mathbf{CF}) - \mathcal{L}_{fin}(\mathbf{P}) \neq \emptyset$. Základním větám a vlastnostem formálních modelů (výhradně gramatik) s konečným indexem je věnována celá třetí kapitola knihy [12], kde lze najít i alternativní definici konečného indexu, konkrétně pro gramatiky.

Nakonec krátké porovnání těchto dvou základních typů omezení (n-limitovanost a konečný index k):

Věta 3.1. *Nechť X je libovolný přepisující systém definující třídy jazyků $\mathcal{L}(\mathbf{X})$, $\mathcal{L}_k(\mathbf{X})$ a ${}_k\mathcal{L}(\mathbf{X})$, pak platí $\mathcal{L}_k(\mathbf{X}) \subseteq {}_k\mathcal{L}(\mathbf{X})$.*

Důkaz věty 3.1. Věta plyne z definice obou omezení (indexu n a n-limitovanosti), kdy konečný index je podstatně restriktivnější než n-limitovanost.

Závěrem podotkněme, že je možné aplikovat omezení na konečný index nejen na svět gramatik, ale i na svět automatů. Především v případě zásobníkového automatu bychom mohli získat zajímavé výsledky, které budou pravděpodobně také velmi blízké výsledkům z oblasti gramatik.

3.3.2 Pracovní prostor

Definice *pracovního prostoru* (*Workspace*) pracuje s gramatickým pojmem derivace, který však lze bez problémů nahradit obecnějším pojmem posloupnost konfigurací přepisujícího systému (definice 3.3 se inspiruje stranou 15 v [12]).

Definice 3.3. Mějme přepisující systém $H = (\Sigma, R)$ (libovolného typu, např. neomezenou gramatiku) s úplnou abecedou Σ, abecedou pasivních symbolů $T \subseteq \Sigma$ a počáteční konfigurací σ. Nechť ξ je konfigurace a $\mathrm{component}(\xi, j)$ značí komponentu z dané konfigurace, na kterou se vztahuje omezení na pracovní prostor ($j \in \{1, 2, \ldots, m\}$, je-li konfigurace m-tice). Dále máme posloupnost konfigurací $D\colon \sigma \Rightarrow w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$, $n \geq 1$. Definujeme $WS(w_n, D) = \max(\{|\mathrm{component}(w_i, j)| \mid 0 \leq i \leq n\})$ a $WS(x, H) = \min(\{WS(x, D) \mid D$ posloupnost konfigurací

vedoucí ke konfiguraci ξ v H, že component$(\xi, j) = x\}$). $WS(x, H)$ označuje *pracovní prostor* potřebný pro zpracování x v systému H. Existuje-li konstanta $k \geq 0$ taková, že pro všechny věty $x \in L(H) - \{\varepsilon\}$ platí $WS(x, H) \leq k \cdot |x|$, pak má přepisující systém H *omezený pracovní prostor.*

K tomuto typu omezení se vztahují následující dvě důležité věty (převzato z [12]).

Věta 3.2. *Má-li gramatika G typu 0 omezený pracovní prostor, pak je $L(G)$ kontextový jazyk.*

Věta 3.3. *Bezkontextová gramatika má omezený pracovní prostor implicitně již ze své definice.*
Důkaz věty 3.3. Víme, že bezkontextové gramatiky jsou ekvivalentní s bezkontextovými gramatikami bez vymazávajících pravidel, takže uvažujme výlučně bezkontextové gramatiky bez vymazávajících pravidel. Pak ovšem existuje jisté minimální $k \geq \max(\{\text{rhs}(p)|\ p \in P\})$ a nejkratší vygenerovaný řetězec je délky 1 (po aplikování konečného počtu pravidel). Potom ovšem $|w| \leq |x|$, protože řetězec nelze zkrátit. Důkaz lze nalézt také v [12].

3.4 Složitost formálních modelů

Složitost se zabývá kvantifikováním a klasifikováním nároků na paměť a čas jak při zpracovávání vět jazyků, tak i uchovávání konečných popisů formálních modelů, které jazyk definují (přehled lze nalézt v [56] a populárněji shrnuto v [8]).

V duchu dynamického a statického omezování přepisujících systémů se studium složitosti v rámci teorie formálních jazyků dělí na dvě větve:

1. časová a prostorová složitost (dynamického charakteru; více praktická)
2. popisná složitost (statického charakteru; spíše teoretická)

Koncept 3.1. Prostorová a časová (*Space Complexity* a *Time Complexity*) složitost říká, jak efektivně instance formálního modelu zpracovává větu jazyka, potažmo jazyk. Konkrétní hodnoty potom získáváme pomocí ujednoceného, dostatečně obecného formálního modelu (nejčastěji Turingův stroj, viz [48]). Časová složitost udává konkrétní počet přechodů/kroků pro zpracování vět, většinou v závislosti na jejich délce. Maximální počet potřebných paměťových buněk (např. na pásce Turingova stroje) pro práci modelu je zase klíčový údaj pro prostorovou složitost.

Pro celé třídy jazyků nebo formálních modelů pak mluvíme o třídách složitosti, které jsou navíc většinou vyjádřeny pouze asymptoticky (tj. omezené shora, zdola či z obou stran nějakou matematickou funkcí závislou na délce věty).

Opomeňme nyní praktické pojmy časové a prostorové složitosti, které bývají většinou aplikovány na výpočetně úplné modely, jako například Turingův stroj, RAM model (*Random Access Memory*) či moderní programovací jazyk, a podívejme se na existující pojem *popisné složitosti.*

Koncept 3.2. *Popisná složitost* (*Descriptional Complexity* nebo *Syntactic Complexity*) sdružuje různé metriky charakterizující efektivitu uchovávání konečného popisu formálního modelu. Nejčastěji se jedná o kardinality jednotlivých komponent přepisujícího systému nebo jejich význačných podmnožin, jež obsahují prvky jisté komponenty s nějakou komplikovanější podmínkou než pro případ základního modelu, ze kterého vycházíme (např. počet pravidel s kontextovou podmínkou, vycházíme-li z bezkontextových pravidel).

Popisná složitost je v teoretické informatice velmi zajímavé a populární téma (např. [5, 41, 43, 44, 45, 46] a celý soubor výsledků najdeme v knihách [12] a [53]). Umožňuje nám měřit a porovnávat

efektivitu popisů prostřednictvím formálních modelů, respektive přepisujících systémů. Většinou se autoři článků při redukci popisné složitosti zaměřují pouze na jednu komponentu (jeden rozměr). O to výjimečnější a cennější jsou pak výsledky redukující více komponent najednou (viz [51]).

3.4.1 Klasifikace složitosti formálních modelů

Ve zbytku sekce se podívejme na alternativní klasifikaci složitosti formálních modelů a zaveďme dynamickou (běhovou) variantu popisné složitosti, tzv. *dynamickou složitost*, která má již bližší vztah k praktické prostorové složitosti, ale není tak obecná. Míry jako konečný index, n-limitovanost a pracovní prostor lze potom studovat pod záštitou dynamické složitosti, která má bližší vztah k přepisujícím systémům a teorii formálních jazyků než obecná prostorová a časová složitost.

Následující odstavce zavádějí dva druhy klasifikace relevantní pro tuto práci.
Dva přístupy:

1. praktický — využívá metrik jako počet instrukcí procesoru a buněk v paměti nutných pro zdárný výpočet algoritmu v závislosti na vstupu (např. jeho délce). Tento přístup je do značné míry závislý na zvoleném formálním modelu, a především jeho praktické implementaci (použité hardwarové platformě apod.).

2. teoretický — je značně robustnější a nevyžaduje zvolení způsobu implementace, ale pouze samotného formálního modelu či jeho konkrétní instance (např. konkrétní gramatika). Sledují se vlastnosti ve vztahu k záznamu samotné instance formálního modelu i jeho provádění jako počet pravidel, determinismus, počet neterminálů, maximální velikost konfigurace a další. Nevýhodou tohoto přístupu je problematické porovnávání složitosti nekompatibilních formálních modelů, respektive existence či neexistence různých měr v různých modelech.

Ortogonální pohled na složitost rozděluje přístupy ke složitosti podle toho, jestli nás zajímá spíše efektivnější popis instance modelu, nebo efektivnější zpracování.

1. popisná složitost — statický charakter; např. velikost komponent pro popis modelu a jejich složitost (počet pravidel, neterminálů, maximální délka pravidel, velikost LR tabulky nebo složitost aplikace nejnáročnějšího pravidla)

2. dynamická složitost — dynamický (běhový) charakter sledující ukazatele během samotného provádění (práce) konkrétní instance modelu; např. konečný index přijímání konkrétní věty, minimální/maximální velikost pracovního prostoru (viz definice 3.3), složitost výběru následného pravidla pro aplikaci nebo rychlost růstu stavového prostoru při simulaci nedeterministického chování pomocí algoritmu navracení (*backtracking*).

V práci se budeme zabývat pouze teoretickým přístupem ke složitosti. Přesto je vztah k praktické složitosti často zřejmý.

Zbytek textu se snaží o zesystematičtění přístupu k omezování konfigurací formálních modelů podle právě nastíněných kritérií a jejich kombinací. Mnoho konkrétních měr této tzv. dynamické složitosti již bylo více či méně studováno, ovšem bez zařazení do širšího kontextu, o které se snaží tato práce.

Část II

Nové formální modely a jejich omezování

Kapitola 4

Definice

Nové formální modely zavedené v rámci této práce mají několik společných rysů. Ve dvou případech se jedná o přepisující systémy, které nelze zcela striktně řadit buď mezi gramatiky, nebo automaty: #-přepisující systémy (sekce 4.1) a redukující hluboké zásobníkové automaty (sekce 4.2 resp. 4.2.2). Třetí model se zabývá řízením zásobníkového automatu prostřednictvím přímého omezování jeho konfigurací.

Samotné dílčí výsledky, které byly o těchto modelech dokázány, jsou uvedeny spolu se všemi vypracovanými důkazy. Ve většině případech diskutujeme vyjádřovací sílu daného formálního modelu (kapitola 5 a částečně 6). Mezi matematicky nejvýznamnější důsledky těchto ekvivalencí tříd jazyků patří tvorba nekonečných hierarchií závislých na omezování těchto modelů (např. index k nebo n-limitovanost).

4.1 #-přepisující systémy

V celé následující sekci budeme hovořit o klíčovém formálním modelu této práce, včetně několika zkoumaných modifikací. Výsledky týkající se tohoto přepisujícího systému tvoří esenciální část textu.

Zavedeme a budeme studovat tzv. #-přepisující systémy, které reprezentují generativní řízený formální model. #-přepisující systémy kombinují vlastnosti automatů a gramatik. Z oblasti gramatik používají metodu zpracování věty jazyka—generování věty. Od konečných automatů si model vypůjčuje konečně-stavové řízení a vynechání neterminálních symbolů ze své definice.

Největší část studia tohoto nového přepisujícího systému je věnována omezování na konečných index a dalším přirozeným modifikacím (především podsekce 4.1.3 a 4.1.4).

Poprvé byl koncept publikován na studentské soutěži [30] a [31]. Detailní rozpracování hlavních výsledků ([37]) pak tvořilo základ pro mezinárodní časopiseckou publikaci [34].

4.1.1 Motivace

V teorii formálních jazyků je většina jazyk-definujících formálních modelů založených na přepisujících systémech, které jsou reprezentovány buď gramatikami, nebo automaty.

Připomeňme, že gramatiky jazyk generují a automaty jej akceptují, což tvoří zásadní rozdíl těchto dvou přístupů k jazykům. Vezměme například bezkontextovou gramatiku G (viz rigorózní definice 2.13). G obsahuje abecedu terminálů a abecedu neterminálů, z nichž jeden neterminál je

označen jako počáteční symbol. Ze startujícího neterminálu začneme postupným přepisováním jednotlivých neterminálů generovat větnou formu, což je řetězec nad úplnou abecedou (tj. terminálními i neterminálními symboly). Posloupnost přepisování s počátkem ve startujícím neterminálu a koncem ve větné formě, která obsahuje pouze terminální symboly, se nazývá derivace věty jazyka a někdy říkáme, že G generuje větu jazyka. Množina terminálních takto vygenerovaných řetězců určuje jazyk generovaný gramatikou G.

Z teorie automatů můžeme připomenout například konečný automat M (viz definice 2.16), který má konečnou množinu stavů, z nichž jeden stav je definován jako počáteční (startující). Některé stavy navíc označíme jako koncové, a ty budou mít potom při práci automatu speciální význam. M pracuje prostřednictvím provádění přechodů. Během přechodu provede změnu aktuálního stavu (konečná vnitřní paměť modelu) a přečtení jednoho vstupního symbolu. Pokud automat M provede podle jeho pravidel posloupnost přechodů tak, že začne v počátečním stavu, přečte celý vstupní řetězec a skončí ve stavu koncovém, pak říkáme, že M přijímá vstupní řetězec. Množina všech přijímaných řetězců tedy tvoří jazyk definovaný automatem M.

Mezi formálními modely však existují také jazyk-definující přepisující systémy, které obsahují a kombinují vlastnosti jak gramatik, tak automatů (viz [6], [21] a [55]). Tento přístup k definování jazyk-definujícího přepisujícího systému je podstatně méně častý, než by předpovídala přirozenost této modifikace. Podobně v duchu této práce, kdy se snažíme demonstrovat možnosti kombinování světa automatů a gramatik, definujme nový formální model, tzv. #-přepisující systém (viz především [34]). Opravdu, na jednu stranu je tento model generativní stejně jako gramatiky, a na druhou jako automaty používá konečně-stavové řízení.

Inspirací pro vytvoření nového typu přepisujícího systému bylo, kromě kombinování přístupu gramatik a automatů, také následující:

- operace rozdělování řetězců často vídaná například v biologii ([57], [72]), kterou námi navržený přepisující systém zjednodušeně simuluje prostřednictvím zavedení jediného přepisovatelného symbolu, který z hlediska rozdělování řetězce tvoří hranici obou rozdělených částí. Tento symbol značme # a nazývejme *hranice* (*bounder*). Hranice tedy rozděluje větnou část konfigurace systému na konečný počet částí, které jsou pak tvořeny výhradně pasivními symboly;

- čisté gramatiky, kde se nepoužívají neterminální symboly—na rozdíl od čistých gramatik my povolujeme jedinpu proměnnou, tzv. hranice, takže nelze provádět přepis libovolného symbolu v konfiguraci. Dále na toto omezení můžeme nahlížet jako na omezení popisné složitosti, kdy omezíme význačnou podmnožinu úplné abecedy systému na jeden jediný symbol. Přesněji řečeno, jediný význačný symbol (#) úplné abecedy se nesmí vyskytovat v žádné větě definovaného jazyka;

- konečné automaty pro konečně-stavové řízení;

- bezkontextové gramatiky pro jednoduchý tvar přepisujícího jádra každého pravidla #-přepisujícího systému, tj. jednu hranici přepisujeme na libovolný řetězec, včetně prázdného;

- n-limitovanost aplikovaná na každé pravidlo zvlášť, kdy vyžadujeme možnost přepsání pouze několika nejlevějších přepsatelných symbolů (proměnných, v našem případě hranic). U #-přepisujícího systému tuto podmínku začleníme přímo do definice tvaru pravidla. Každé pravidlo mimo jiné určí, na kolikátou hranici ve větné části konfigurace zleva může být aplikován předepsaný přepis.

Nyní následuje definice #-přepisujícího systému. Než-li k ní přistoupíme, znova zdůrazněme, že základní podoba systému využívá pravidla bezkontextového tvaru obohacená o konečně-stavové řízení a omezování přepisování nejlevějších hranic prostřednictvím n-limitovanosti. Nejedná se však o klasickou n-limitovanost aplikovanou na celou konfiguraci, ale vztahující se ke každému pravidlu zvlášť (vyjádřeno číselným pořadím aktivní hranice).

Budeme-li hovořit o #-přepisujícím systému s jiným typem přepisovacích pravidel než bezkontextovým, bude to vždy explicitně řečeno.

4.1.2 Definice

Definice 4.1. Nechť $\mathbb{N}$ je množina všech kladných celých čísel (viz definice 2.1). *Bezkontextový #-přepisující systém* (*context-free #-rewriting system* (CF#RS) nebo zkráceně jen #-přepisující systém) je čtveřice $H = (Q, \Sigma, s, R)$, kde Q je konečná množina stavů, Σ je úplná abeceda obsahující speciální symbol, který budeme nazývat *hranice* (*bounder*) a zapisovat jako #, $Q \cap \Sigma = \emptyset$, $s \in Q$ je počáteční stav a

$$R \subseteq Q \times \mathbb{N} \times \{\#\} \times Q \times \Sigma^*$$

je konečná relace, jejímž členům říkejme *pravidla*. Pravidlo $(p, n, \#, q, x) \in R$, kde $n \in \mathbb{N}$, $q, p \in Q$ a $x \in \Sigma^*$ zapisujme jako $r\colon p\,{}_n\# \to q\ x$, kde r je unikátní návěští, které lze v případě jednoznačnosti vypustit.

Konfigurace systému H je dvojice z kartézského součinu $Q \times \Sigma^*$. Nechť χ je množina všech konfigurací systému H. Nechť $(p, u\#v), (q, uxv) \in \chi$ jsou dvě konfigurace, které budeme pro jednoduchost zapisovat bez závorek v jediném řetězci jako $pu\#v$, $quxv$, kde $p, q \in Q$, $u, v, x \in \Sigma^*$. H provádí *výpočetní krok* z $pu\#v$ do $quxv$ pomocí pravidla $r\colon p\,{}_n\# \to q\ x$, když $\mathrm{occur}(\#, u) = n - 1$, symbolicky psáno $pu\#v \Rightarrow quxv\ [r]$ v H nebo krátce $pu\#v \Rightarrow quxv$.

Standardním způsobem rozšíříme $\Rightarrow$ na $\Rightarrow^m$ (m-kroký výpočet), pro $m \geq 0$, a dále na tranzitivní uzávěr $\Rightarrow^+$ (netriviální výpočet) a tranzitivně-reflexivní uzávěr $\Rightarrow^*$ (výpočet).

Jazyk derivovaný systémem H, $L(H)$, definujeme jako

$$L(H) = \{w \mid s\# \Rightarrow^* qw,\ q \in Q, w \in (\Sigma - \{\#\})^*\}.$$

Poznámka 4.1. Podíváme-li se blíže na tvar pravidla v CF#RS ($r\colon p\,{}_n\# \to q\ x$), zjistíme, že symbol hranice (#) se v něm na levé straně vyskytuje vždy, a nemá tudíž informační hodnotu. V případě kontextového tvaru pravidel je nutné uvádění na levé straně řetězec k přepsání. Pro zachování jednotného tvaru je přepisovaná hranice uváděna i na levé straně pravidel bezkontextových a pravě-lineárních (viz sekce 4.1.3) #-přepisujících systémů.

Definice 4.2. Pro #-přepisující systém definujme konečný index na základě definice 3.2 pro obecný přepisující systém, kde položme $N = \{\#\}$, $T = \Sigma - \{\#\}$ a typ přepisujícího systému je samozřejmě #-přepisující systém, tj. $X = \mathrm{CF\#RS}$.

Jinými slovy, je-li k kladné celé číslo, pak #-přepisující systém H je indexu k, jestliže pro každou konfiguraci $x \in \chi$ platí, že $s\# \Rightarrow^* qy = x$ implikuje $\mathrm{occur}(\#, y) \leq k$. Poznamenejme, že H indexu k nemůže dospět do konfigurace, která by obsahovala více jak k hranic (#), čímž se #-přepisující systém indexu k liší například od programovaných gramatik konečného indexu, které mohou derivovat i řetězce obsahující více jak k neterminálů (z těchto větných forem pak již ale nelze vygenerovat větu jazyka s indexem k).

Nechť $\mathcal{L}_k(\mathbf{CF\#RS})$ označuje třídu jazyků definovaných bezkontextovými #-přepisujícími systémy indexu k a $\mathcal{L}_k(\mathbf{P})$ třídu jazyků definovaných programovanými gramatikami indexu k (podle definice 3.2).

Definice 4.3. Pravidlo #-přepisujícího systému nazveme vymazávající (*erasing rule*), jestliže jeho pravá strana obsahuje cílový stav a prázdný řetězec. Například $p\,{}_2\# \rightarrow q\ \varepsilon$.

Příklad 4.1. Mějme bezkontextový #-přepisující systém $H = (\{s, p, q, f\}, \{a, b, c, \#\}, s, R)$, kde R obsahuje pravidla:

1: $s\,{}_1\# \rightarrow p\ \#\#$

2: $p\,{}_1\# \rightarrow q\ a\#b$

3: $q\,{}_2\# \rightarrow p\ \#c$

4: $p\,{}_1\# \rightarrow f\ ab$

5: $f\,{}_1\# \rightarrow f\ c$

$L(H) = \{a^n b^n c^n \mid n \geq 1\}$, kdy $Ind(H) = 2$; tedy #-přepisující systém je indexu 2.
Následuje příklad generování řetězce *aaabbbccc*: $s\# \Rightarrow p\#\#$ [1] $\Rightarrow qa\#b\#$ [2] $\Rightarrow$
$pa\#b\#c$ [3] $\Rightarrow qaa\#bb\#c$ [2] $\Rightarrow paa\#bb\#cc$ [3] $\Rightarrow faaabbb\#cc$ [4] $\Rightarrow faaabbbccc$ [5].

Příklad 4.2. Mějme bezkontextovou gramatiku $G = (\{S\}, \{a, b, a', b'\}, S, P)$, kde P obsahuje pravidla:

1. $S \rightarrow SS$
2. $S \rightarrow \varepsilon$
3. $S \rightarrow aSa'$
4. $S \rightarrow bSb'$

$L(G) = D_2$ tzv. Dyckův jazyk pro dva druhy závorek a platí, že $L(G) \in \mathbf{CF} - \mathcal{L}_{fin}(\mathbf{CF}\#\mathbf{RS})$ (viz [12]).

Z příkladu 4.1 navíc $L(H) \in \mathcal{L}_{fin}(\mathbf{CF}\#\mathbf{RS}) - \mathbf{CF}$, tj. $L(H) \notin \mathbf{CF}$, takže třída bezkontextových jazyků a třída definovaná #-přepisujícími systémy s konečným indexem jsou neporovnatelné.

Nejdůležitější výsledky (věty 5.3 a 5.19) o #-přepisujících systémech se týkají omezení konfigurací na konečný index (viz definice 3.2). Již nyní lze předeslat, že na základě konečného indexu budou tyto systémy tvořit nekonečnou hierarchii jazyků, což je matematicky velmi zajímavá vlastnost, která silně vypovídá o schopnostech různých konkrétních instancí přepisujícího systému. Při podrobnějším pohledu je zde také úzký vztah ke složitosti (jak prostorové, tak časové), a to především z nově představeného pohledu dynamické složitosti.

4.1.3 Založené na pravě-lineárních pravidlech

Jedna z nejpřímočařejších modifikací každého přepisujícího systému je změna povoleného tvaru jádra přepisovacích pravidel, podobně jako to vůči bezkontextovým gramatikám dělají ostatní gramatiky Chomského hierarchie.

Jako speciální případ #-přepisujících systémů nyní zavedeme a budeme studovat pravě-lineární variantu #-přepisujících systémů. Jak již naznačuje název systému, jádro přepisovacích pravidel bude založeno na pravidlech pravě-lineárních gramatik (viz definice 2.15).

Pravě-lineární pravidla mají tu vlastnost, že nikdy nezvyšují počet proměnných (v našem případě hranic) v konfiguraci, tj. pro libovolné pravidlo r platí, že $\mathrm{occur}(\#, \mathrm{rhs}(r)) \leq 1$. Proto je nutné rozdělení provést hned prostřednictvím definování počáteční konfigurace tvaru $s\#^n$, kde s je počáteční stav systému a n je libovolné kladné celé číslo. Počet hranic pak může v průběhu

výpočtu již pouze klesat při aplikaci pravidel tvaru $p\,{}_i\# \to q\ \alpha$, kde $\alpha \in (\Sigma - \{\#\})^*$, a tím se přibližovat k výsledné větě jazyka.

Tyto systémy budou tvořit v závislosti na konstantě n nekonečnou hierarchii definovanou například n-pravě-lineárními jednoduchými maticovými gramatikami (viz sekce 5.1.1). Prostřednictvím dalších výsledků bude také nastíněn vztah k třídě pravě-lineárních jazyků.

Definice 4.4. Nechť $H = (Q, \Sigma, s, R)$ je bezkontextový #-přepisující systém, $n \in \mathbb{N}$ a navíc relace R splňuje

$$R \subseteq Q \times \mathbb{N} \times \{\#\} \times Q \times ((\Sigma - \{\#\})^*\{\#\} \cup (\Sigma - \{\#\})^*),$$

pak H nazývejme *n-pravě lineární #-přepisující systém* (*n-right-linear #-rewriting system*, zkráceně n-RLIN#RS).

Pravidlo $(p, i, \#, q, x) \in R$, kde $i \in \mathbb{N}$, $i \leq n$, $p, q \in Q$ a $x \in \{\alpha\#\} \cup \alpha$, $\alpha \in (\Sigma - \{\#\})^*$. Pravidlo většinou budeme zapisovat v přehlednější formě jako $r\colon p\,{}_i\# \to q\ x$, kde r je jeho unikátní návěští, které může být případně vynecháno.

Pojmy konfigurace, výpočetní krok, m-kroký výpočet ($m \geq 0$), netriviální výpočet a výpočet jsou definovány analogicky jako u bezkontextového #-přepisujícího systému (viz definice 4.1). Výjimku tvoří počáteční konfigurace σ, která je definována jako $\sigma = s\#^n$.

Jazyk generovaný n-RLIN#RS H, $L(H)$, je definován jako

$$L(H) = \{w \mid s\#^n \Rightarrow^* qw,\ q \in Q, w \in (\Sigma - \{\#\})^*\}.$$

Nechť n je celé kladné číslo a σ je počáteční konfigurace n-pravě lineárního #-přepisujícího systému H. H je *indexu* n, pokud každá konfigurace $x = qy$ splňuje podmínku, že $\sigma \Rightarrow^* qy$ implikuje $\mathrm{occur}(\#, y) \leq n$. Všimněme si, že H indexu n nemůže nikdy vygenerovat řetězec obsahující více jak n hranic (#), takže n-RLIN#RS H je vždy indexu nejvýše n.

Nechť $n \in \mathbb{N}$. $\mathcal{L}(n\text{-}\mathbf{RLIN\#RS})$ označuje třídu jazyků definovanou n-pravě-lineárními #-přepisujícími systémy.

Definice 4.5. Výpočetní krok nazveme #-vymazávající (*#-erasing*), je-li v tomto kroku přepsána # na řetězec terminálů nebo prázdný řetězec.

Nechť d je n-kroký výpočet v H, pro libovolné $n \geq 0$. Pomocí d_i, resp. ${}_t d_i$ budeme značit i-tý výpočetní krok v posloupnosti kroků d, resp. i-tý výpočetní krok přepisující zleva t-tou #. t nazýváme *stupeň kroku* d_i. Výpočet d označujeme za *úspěšný*, pokud d popisuje výpočet z počáteční konfigurace do koncové konfigurace (q, w), kde $w \in (\Sigma - \{\#\})^*$.

Příklad 4.3. 3-RLIN#RS $H_2 = (\{s, p, q, r, t\}, \{a, b, c, \#\}, s, R_2)$, kde R_2 obsahuje pravidla

1: $s\,{}_1\# \to p\ a\#$

2: $p\,{}_2\# \to q\ b\#$

3: $q\,{}_3\# \to s\ c\#$

4: $s\,{}_1\# \to r\ a$

5: $r\,{}_1\# \to t\ b$

6: $t\,{}_1\# \to t\ c$

Například H_2 provádí výpočet $aabbcc$ následovně: $s### \Rightarrow pa### [1] \Rightarrow qa\#b## [2] \Rightarrow sa\#b\#c\# [3] \Rightarrow raab\#c\# [4] \Rightarrow taabbc\# [5] \Rightarrow taabbcc$ [6].

Je zřejmé, že H z příkladu 4.1 je indexu 2 a H_2 v tomto příkladu je indexu 3. Oba systémy ale popisují tentýž jazyk $L(H) = L(H_2) = \{a^n b^n c^n \mid n \geq 1\}$.

Představili jsme n-pravě-lineární #-přepisující systém, který vznikl přirozenou modifikací bezkontextového #-přepisujícího systému. Třídy jazyků charakterizované touto variantou přepisujícího systému tvoří nekonečné hierarchie tříd jazyků v závislosti na počtu hranic v počáteční konfiguraci. Důkaz nekonečné hierarchie využije m-paralelní n-pravě-lineární jednoduché maticové gramatiky, které budou ještě dále zjednodušeny na n-pravě-lineární jednoduché maticové gramatiky.

Další podstatný výsledek se bude vztahovat k třídě regulárních jazyků. Pokud totiž n-pravě-lineární #-přepisující systém omezíme jistou obecnou podmínkou na cyklické přepisování hranic, získáme omezení, které bude degradovat vyjadřovací sílu právě na úroveň regulárních jazyků. Využití věty 5.8 bude demonstrováno i na odvození jednodušších závěrů, které by bylo nutno jinak dokazovat zdlouhavějším způsobem.

Oba základní výsledky týkající se n-pravě-lineárních #-přepisujících systémů jsou uvedeny v sekci 5.1.1 ve větách 5.7 a 5.8.

4.1.4 Založené na zobecněných pravidlech

Předchozí dvě podsekce představily #-přepisující systémy založené na bezkontextových a pravě-lineárních pravidlech, která vždy během jednoho výpočetního kroku přepisují jediný výskyt hranice. Samozřejmě lze tyto systémy dále zobecnit tak, že během výpočetního kroku budou přepisovat celý řetězec obsahující alespoň jednu hranici. Jak se pak změní vyjadřovací síla takto generalizovaného #-přepisujícího systému?

V této sekci budeme diskutovat zobecněnou verzi #-přepisujících systémů, které obsahují kontextová pravidla. Prezentované výsledky nás přesvědčí, že tímto zobecněním v případě omezení na konečný index nezískáme mocnější formální model. Získáme tak pouze další charakterizaci pro dobře známou nekonečnou hierarchii, jež je tvořena programovanými gramatikami konečného indexu.

Původní verze #-přepisujícího systému je založena na pravidlech tvaru $p\,{}_i\# \to q\ \gamma$, kde p, q jsou stavy, i je kladné celé číslo a γ je řetězec nad úplnou abecedou. Pomocí tohoto pravidla systém přepisuje i-tou # řetězcem γ a zároveň mění aktuální stav z p na q. Nyní diskutujme zobecněnou verzi #-přepisujícího systému, který bude používat pravidla tvaru $p\,{}_i\alpha\#\beta \to q\ \alpha\gamma\beta$, kde α a β jsou řetězce a nazýváme je levý a pravý kontext; ostatní symboly mají stejný význam jako ve výchozím typu systému. Toto zobecněné pravidlo je možno aplikovat na hranici, pokud se tato hranice vyskytuje uprostřed α-β kontextu; v ostatních aspektech se zobecněné #-přepisující systémy od bezkontextových neliší.

Definice 4.6. Zobecněný #-přepisující systém (*generalized #-rewriting system*, zkráceně G#RS) je čtveřice $H = (Q, \Sigma, s, R)$, kde Q, Σ a s mají naprosto stejný význam jako u bezkontextového #-přepisujícího systému a konečná relace R splňuje

$$R \subseteq Q \times \mathbb{N} \times \Sigma^*\{\#\}\Sigma^* \times Q \times \Sigma^*.$$

Pravidla jsou většinou psána ve tvaru $r\colon p\,{}_i\alpha\#\beta \to q\ \alpha\gamma\beta \in R$, kde r je unikátní návěští, $i \in \mathbb{N}$, $q, p \in Q$ a $\alpha, \beta, \gamma \in \Sigma^*$, kde α a β jsou *levý* a *pravý kontext* pravidla r.

Konfigurace zobecněného systému H je dvojice z $Q \times \Sigma^*$. Výpočetní krok je definován trochu složitěji než u CF#RS. Nechť $pu\alpha\#\beta v$, $qu\alpha\gamma\beta v$ jsou dvě konfigurace, $p, q \in Q$, $u, v, \alpha, \beta, \gamma \in \Sigma^*$, $i \in \mathbb{N}$ a $\mathrm{occur}(\#, u\alpha) = i - 1$. Pak H provádí *výpočetní krok* z $pu\alpha\#\beta v$ do $qu\alpha\gamma\beta v$ použitím

pravidla $r\colon p\,{}_i\alpha\#\beta \to q\ \alpha\gamma\beta$; symbolicky píšeme $pu\alpha\#\beta v \Rightarrow qu\alpha\gamma\beta v$ $[r]$ v H. Pokud chceme explicitně vyjádřit pouze pozici hranice, která byla ve výpočetním kroku přepsána, píšeme $pu\alpha\#\beta v\,{}_i\Rightarrow qu\alpha\gamma\beta v$ v H.

Analogicky jako u bezkontextového #-přepisujícího systému rozšíříme výpočetní krok na m-kroký výpočet ($m \geq 0$), netriviální výpočet a výpočet (viz definice 4.1).

Jazyk generovaný G#RS H je definován totožně jako u CF#RS.

Poznamenejme, že zvláštní případ G#RS, kdy každé pravidlo $r\colon p\,{}_i\alpha\#\beta \to q\ \alpha\gamma\beta \in R$ splňuje podmínku, že $\alpha = \beta = \varepsilon$, pak je H bezkontextový #-přepisující systém (CF#RS).

Nechť k je kladné celé číslo. H je indexu k, pokud pro každou konfiguraci $x = qy$ systému H platí, že $s\# \Rightarrow^* qy$ implikuje $\mathrm{occur}(\#, y) \leq k$ (tedy analogicky k definici 3.2).

Definice 4.7. Pro G#RS H, $\max_L(H)$ a $\max_R(H)$ označují maximální délku levé a pravé strany pravidel. Přesněji, nechť $H = (Q, \Sigma, s, R)$ je G#RS, pak $\max_L(H) = \max(\{|\alpha| \mid p\,{}_i\alpha \to q\ \beta \in R\})$ a $\max_R(H) = \max(\{|\beta| \mid p\,{}_i\alpha \to q\ \beta \in R\})$.

Nechť k je kladné celé číslo. $\mathcal{L}_k(\mathbf{G\#RS})$ resp. $\mathcal{L}(\mathbf{G\#RS})$ označují třídu jazyků generovaných zobecněnými #-přepisujícími systémy konečného indexu k resp. bez omezení na konečný index.

Příklad 4.4. G#RS $H_3 = (\{s, p, q\}, \{a, b, c, \#\}, s, R_3)$, kde R_3 obsahuje pravidla

1: $s\,{}_1\# \to s\ a\#\#$

2: $s\,{}_2a\#\# \to p\ a\#b\#c$

3: $p\,{}_1a\# \to q\ aa\#$

4: $q\,{}_2b\#c \to p\ bb\#cc$

5: $p\,{}_1a\# \to p\ a$

6: $p\,{}_1b\#c \to p\ bc$

Například, H_3 provádí generování $aabbcc$ následovně: $s\# \Rightarrow sa\#\#$ [1] $\Rightarrow pa\#b\#c$ [2] $\Rightarrow qaa\#b\#c$ [3] $\Rightarrow paa\#bb\#cc$ [4] $\Rightarrow paabb\#cc$ [5] $\Rightarrow paabbcc$ [6].

Detailněji se podívejme například na druhé pravidlo a jeho aplikaci v tomto výpočtu. Pro pravidlo $s\,{}_2a\#\# \to p\ a\#b\#c$ jsou dva různé způsoby jak volíme levý a pravý kontext přepisované hranice. Na levé straně pravidla jsou dvě hranice a to, kterou hranici bude možné přepsat závisí na výchozí konfiguraci, protože pravidlo diktuje přepis právě druhé hranice zleva. V příkladu je tedy brán jako levý kontext $\alpha = a\#$ a pravý kontext $\beta = \varepsilon$.

Pokud si však vezmeme jiný příklad výpočetního kroku v jiném zobecněném systému, mohou být kontexty jiné. Například $sa\#a\#\# \Rightarrow pa\#a\#b\#c\#$ $[s\,{}_2a\#\# \to p\ a\#b\#c]$, kdy je levý kontext $\alpha = a$ a pravý $\beta = \#$.

Je zřejmé, že H v příkladu 4.1 a H_3 z tohoto příkladu jsou oba indexu 2 a oba systémy popisují tentýž jazyk $L(H_1) = L(H_3) = \{a^n b^n c^n \mid n \geq 1\}$, který patří do rozdílu $\mathcal{L}(\mathbf{CF\#RS}) - \mathbf{CF}$, a tedy není bezkontextový.

Hlavním výsledkem pro zobecněné #-přepisující systémy je fakt, že pod omezením na konečný index se vyjadřovací síla oproti bezkontextovým systémům nezmění, takže získáme pouze alternativní charakterizaci nám již dobře známé nekonečné hierarchie jazyků generované programovanými gramatikami konečného indexu (viz [34] a věty 3.1.2i, 3.1.7 v [12]).

Tento výsledek je zajímavý především v porovnání s obdobným zobecněním v případě gramatik Chomského hierarchie (viz věta 2.2), kde kontextové gramatiky mají mnohem větší sílu než běžné bezkontextové gramatiky.

4.1.5 Další varianty #-přepisujících systémů

V rámci dizertační práce bylo studováno také několik dalších variant #-přepisujících systémů. Většina výsledků k těmto modifikacím je v současné době rozpracovaná nebo ve fázi hypotéz. Těmto výsledkům a hypotézám se budeme částečně věnovat také v kapitolách 6 a 7.

Mějme bezkontextový #-přepisující systém $H = (Q, \Sigma, s, R)$. Některé modifikace představených systémů definujme následovně:

1. H je *deterministický* #-přepisující systém, jestliže pro každé pravidlo $p \in Q$ a pro každé celé kladné číslo i platí, že $p\,{}_i\#$ je levá strana nejvýše jednoho pravidla v H. Determinismus #-přepisujících systémů lze smysluplně definovat i několika dalšími způsoby, které však mají odlišné vlastnosti, a blíže se jimi budeme zabývat v kapitole 6.2.1;

2. Nechť $\alpha, \beta \in \chi$ jsou konfigurace. Jestliže $\alpha \Rightarrow \beta$ v H, pak H *přímo redukuje* β na α, pro odlišení značeno $\beta \vdash \alpha$, a H nazýváme *redukující #-přepisující systém*.

3. H pracuje *paralelním způsobem*, pokud simultánně (naráz) přepisuje všechny hranice v aktuální větné formě během jediného výpočetního kroku.

Po jisté úpravě lze varianty přizpůsobit i na n-pravě-lineární a zobecněné #-přepisující systémy. Nyní krátce okomentujme některé představené varianty.

Redukující varianta

Místo dosavadního přístupu generování věty shora dolů se inspirujme syntaktickými analyzátory, které umí pracovat i směrem zdola nahoru, a od generování přejděme k redukování.

Nechť $H = (Q, \Sigma, s, R)$ je #-přepisující systém. H budeme nazývat *redukující #-přepisující systém* (*accepting #-rewriting system*), pokud redukuje daný jazyk pomocí posloupnosti redukcí (místo výpočetních generativních kroků). H provádí *redukční krok* z $quxv$ do $pu\#v$ podle pravidla $r\colon p\,{}_n\# \to q\ x$, symbolicky zapsáno $quxv \vdash pu\#v\ [r]$ v H. Nechť $\vdash^*$ značí reflexivně-tranzitivní uzávěr $\vdash$.

Jazyk definovaný redukujícím systémem H, ${}_rL(H)$, je definován jako

$${}_rL(H) = \{w \mid qw \vdash^* s\#, q \in Q, w \in (\Sigma - \{\#\})^*\}.$$

Uvažujme $H = (\{s, p, q, f\}, \{a, b, c, \#\}, s, R)$ z příkladu 4.1, pak redukující varianta systému provádí přepis řetězce *aaabbbccc* následovně: $faaabbbccc \vdash faaabbb\#cc\ [5] \vdash paa\#bb\#cc [4] \vdash qaa\#bb\#c\ [3] \vdash pa\#b\#c\ [2] \vdash qa\#b\#\ [3] \vdash p\#\#\ [2] \vdash s\#\ [1]$.

Paralelní varianta

Paralelní #-přepisující systém je pětice $H = (Q, \Sigma, s, P, R)$, kde Q, Σ a s jsou definovány stejným způsobem jako u CF#RS, $P \subseteq \mathbb{N} \times \Sigma^*$ je konečná relace obsahující položky nazývané *jádra pravidel*, která budeme zapisovat ve formě ${}_n\#\,{}_s\!\!\rightarrow x$, $n \in \mathbb{N}$, $x \in \Sigma^*$ (levou a pravou stranu pravidla oddělujeme šipkou s levým dolním indexem „s“), a $R \subseteq Q \times 2^P \times Q$ je konečná relace s podmínkou, že pro každé pravidlo $(p, F, q) \in R$, $p, q \in Q$, $F \in 2^P$ platí, že pro každé dvě jádra pravidel $c, d \in F$, $c\colon {}_i\#\,{}_s\!\!\rightarrow x_c$, $d\colon {}_j\#\,{}_s\!\!\rightarrow x_d$ je $i \neq j$.

Pravidlo $t = (p_t, \{r_1, \ldots, r_m\}, q_t) \in R$, $m \geq 1$, je aplikovatelné na konfiguraci px, $p \in Q$, $x \in \Sigma^*$, když a jen když $p = p_t$, $\mathrm{occur}(\#, x) \geq i_j$, pro všechna $1 \leq j \leq m$, kde $r_j\colon {}_{i_j}\#\,{}_s\!\!\rightarrow y_j$.

Typ systému	Zkratka	Tvar pravidel
n-pravě lineární	n-RL#RS	$Q \times \mathbb{N} \times \{\#\} \times Q \times (\Sigma - \{\#\})^*(\{\#, \varepsilon\})$
Bezkontextové (základní)	#RS nebo CF#RS	$Q \times \mathbb{N} \times \{\#\} \times Q \times \Sigma^*$
Kontextové (zobecněné)	G#RS	$Q \times \mathbb{N} \times \Sigma^*\{\#\}\Sigma^* \times Q \times \Sigma^*$

Tabulka 4.1: #-přepisující systémy - přehled

H provádí výpočetní krok z pu do qv použitím pravidla $t = (p, \{r_1, \ldots, r_m\}, q)$, symbolicky $pu \,{}_p\!\Rightarrow qv\ [t]$ v H, pokud je t aplikovatelné na pu a jádra pravidel $r_1, \ldots, r_m$ jsou aplikována na u a stav p je změněn na nový stav q.

Uvažujme paralelní #-přepisující systém indexu k jako přímou analogii k CF#RS indexu k. Popis aplikovatelnosti pravidla pak musí být rozšířen o podmínku, že $\text{occur}(\#, x) - m + \sum_{l=1}^{m} \text{occur}(\#, y_l) \leq k$.

Nechť ${}_p\!\Rightarrow^*$ značí tranzitivně-reflexivní uzávěr relace ${}_p\!\Rightarrow$. *Jazyk generovaný* paralelním #-přepisujícím systémem H, ${}_pL(H)$, je definován jako

$${}_pL(H) = \{w \mid s\# \,{}_p\!\Rightarrow^* qw,\ q \in Q, w \in (\Sigma - \{\#\})^*\}.$$

Tímto jsme stručně popsali i některé další modifikace #-přepisujících systémů, které budou studovány především v budoucnu (viz kapitola 7).

Tabulka 4.1 slouží pro shrnutí různých definovaných #-přepisujících systémů.

4.2 Hluboké zásobníkové automaty

Teprve nedávno se začaly studovat duální modely k řízeným gramatikám, které souhrnně nazvěme *řízené automaty* (např. [28, 23, 52]). Jedním z dalších reprezentantů, který lze také označit jako zobecněný klasický zásobníkový automat, je následující nový formální model teorie automatů, jenž byl nedávno zaveden profesorem Medunou v [50].

I v následujícím přepisujícím systému najdeme princip kombinování automatů s gramatikami. Klasický zásobníkový automat pracuje vždy pouze s vrcholem zásobníků (viz definice 2.17), kdežto gramatika provádí přepisy obecně ve kterékoliv části větné formy. Nyní se inspirujme gramatikami a studujme modifikovaný zásobníkový automat, který umožňuje pracovat i se symboly hlouběji na zásobníku. Abychom však úplně „nezahodili“ základní princip zásobníkové struktury, jež je optimalizovaná pouze na práci s vrcholem zásobníku (operace odebrání z vrcholu a vložení na vrchol), budeme uvažovat manipulaci pouze se symboly do jisté omezené maximální hloubky (např. pouze prvními dvěma aktivními symboly od vrcholu směrem ke dnu zásobníku). Vše ostatní ponechejme stejné jako u klasického zásobníkového automatu.

Definice 4.8. *Hluboký zásobníkový automat* (*Deep Pushdown Automaton*, zkráceně DTDP) je sedmice $M = (Q, T, \Gamma, R, s, S, F)$, kde Q je konečná množina *stavů*, T je *vstupní abeceda*, Γ je *zásobníková abeceda*, $\mathbb{N}$ označuje množinu všech kladných celých čísel, $\mathbb{N}$, Q a Γ jsou po dvojicích disjunktní, $T \subseteq \Gamma$, $\# \in \Gamma - T$ a znaku $\#$ říkáme *dno zásobníku*, $R \subseteq (\mathbb{N} \times Q \times (\Gamma - (T \cup \{\#\})) \times Q \times (\Gamma - \{\#\})^+) \cup (\mathbb{N} \times Q \times \{\#\} \times Q \times (\Gamma - \{\#\})^*\{\#\})$ je konečná relace, $s \in Q$ je *počáteční stav*, $S \in \Gamma$ je *počáteční zásobníkový symbol* a $F \subseteq Q$ je množina *koncových stavů*.

Místo pětice $(m, q, A, p, v) \in R$ používáme notaci $mqA \to pv \in R$, $mqA \to pv$ nazýváme *pravidlem* a R je pak *množina pravidel automatu* M.

Definice 4.9. *Konfigurace* automatu M (viz definice 4.8) je trojice z $Q \times T^* \times (\Gamma - \{\#\})^*\{\#\}$. Nechť χ označuje množinu všech možných konfigurací automatu M a nechť $x, y \in \chi$ jsou dvě konfigurace. M provádí *porovnání* (*pop*) při přechodu od konfigurace x ke konfiguraci y, zapisujeme jako $x \ {}_p\Rightarrow\ y$, jestliže $x = (q, az, au)$, $y = (q, z, u)$, kde $a \in T$, $z \in T^*$, $u \ \in \Gamma^*$, $q \in Q$. M expanduje (*expand*) svůj zásobník z konfigurace x na y pomocí pravidla $r\colon mqA \to pv \in R$, zapisujme $x \ {}_e\Rightarrow\ y\ [r]$, když $x = (q, w, uAz)$, $y = (p, w, uvz)$, kde $A \in \Gamma - T$, $u, v, z \ \in \Gamma^*$, $q, p \in Q$, $w \in T^*$ a $\mathrm{occur}(\Gamma - T, u) = m - 1$.

Pro M provádějící přechod $x \ {}_e\Rightarrow\ y$ podle pravidla $mqA \to pv$, píšeme $x \ {}_e\Rightarrow\ y\ [mqA \to pv]$. Říkáme, že $mqA \to pv$ je *pravidlo hloubky* m a $x \ {}_e\Rightarrow\ y\ [mqA \to pv]$ je *expanze hloubky* m. M provádí *přechod* z konfigurace x do y, značme $x \Rightarrow y$, když M buď $x \ {}_e\Rightarrow\ y$, a nebo $x \ {}_p\Rightarrow\ y$. Je-li $n \in \mathbb{N}$ minimální kladné celé číslo takové, že každé pravidlo automatu M má hloubku n nebo nižší, říkáme, že M *je hloubky* n a píšeme ${}_nM$.

Standardním způsobem rozšíříme ${}_p\Rightarrow, {}_e\Rightarrow$ a $\Rightarrow$ na ${}_p\Rightarrow^m, {}_e\Rightarrow^m$ a $\Rightarrow^m$, pro $m \geq 0$ a dále definujeme ${}_p\Rightarrow^+, {}_p\Rightarrow^*, {}_e\Rightarrow^+, {}_e\Rightarrow^*, \Rightarrow^+$ a $\Rightarrow^*$.

Nechť M je hloubky n, pro nějaké $n \in \mathbb{N}$. *Jazyk přijímaný automatem* ${}_nM$ definujeme jako $L({}_nM)$, kde $L({}_nM) = \{w \in T^* |\ (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#)$ v ${}_nM$ a $f \in F\}$.

Pro všechna $k \geq 1$ definujme třídy jazyků ${}_k\mathcal{L}(\mathbf{DTDP}) = \{L({}_iM) \mid {}_iM$ je hluboký zásobníkový automat, $1 \leq i \leq k\}$.

Podívejme se na vztah mezi n-limitovaností u gramatik a hloubkou n u DTDP. Pokud budeme zapisovat komponentu konfigurace pro reprezentaci obsahu zásobníku jako řetězec zleva doprava, kde nejlevější symbol bude vrchol zásobníku a nejpravější jeho dnem, můžeme porovnat přepisování v této komponentě s větnou komponentou v gramatikách. Při omezování gramatik na n-limitovanost jsme omezovali, kolik neterminálů zleva můžeme v libovolném kroku přepsat, ale neurčovali jsme přesně kolikátý. U DTDP je omezení maximální hloubky n přesně korelující s omezováním práce gramatik pomocí n-limitovanosti. Navíc také určujeme kolikátá proměnná od vrcholu zásobníku bude přepsána (expandována). Opět jsme tedy demonstrovali úzkou provázanost světa automatů a gramatik.

Příklad 4.5. Mějme hluboký zásobníkový automat ${}_2M = (\{s, q, p, f\},\ \{a, b, c\},\ \{A, B, S, \#, a, b, c\},\ R,\ s,\ S,\ \{f\})$ s pravidly

$$\begin{aligned} R = \{\quad & 1sS \to qAB, \\ & 1qA \to paAb, \\ & 1qA \to pab, \\ & 2pB \to qBc, \\ & 2pB \to fc \qquad \}. \end{aligned}$$

Následuje krátký příklad přijetí řetězce *aabbcc* tímto automatem:

$$\begin{aligned} (s, aabbcc, S\#) \ &{}_e\Rightarrow\ (q, aabbcc, AB\#) && [1sS \to qAB] \\ &{}_e\Rightarrow\ (p, aabbcc, aAbB\#) && [1qA \to paAb] \\ &{}_p\Rightarrow\ (p, abbcc, AbB\#) \\ &{}_e\Rightarrow\ (q, abbcc, AbBc\#) && [2pB \to qBc] \\ &{}_e\Rightarrow\ (p, abbcc, abbBc\#) && [1qA \to pab] \\ &{}_p\Rightarrow\ (p, bbcc, bbBc\#) \\ &{}_p\Rightarrow\ (p, bcc, bBc\#) \\ &{}_p\Rightarrow\ (p, cc, Bc\#) \\ &{}_e\Rightarrow\ (f, cc, cc\#) && [2pB \to fc] \\ &{}_p\Rightarrow\ (f, c, c\#) \\ &{}_p\Rightarrow\ (f, \varepsilon, \#). \end{aligned}$$

Předchozí posloupnost přechodů od počáteční ke koncové konfiguraci zapíšeme stručně jako $(s, aabbcc, S\#) \Rightarrow^* (f, \varepsilon, \#)$ a prohlásíme řetězec $aabbcc$ za úspěšně přijatý hlubokým zásobníkovým automatem M. Všimněme si, že $L({}_2M) = \{a^n b^n c^n | n \geq 1\} \in {}_2\mathcal{L}(\mathbf{DTDP})$, kde $L({}_2M) \in \mathbf{CS} - \mathbf{CF}$.

Pozorování 4.1. Podobně jako u #-přepisujících systémů i u hlubokých zásobníkových automatů je omezování definováno implicitně tvarem pravidel a požadavkem na konečnost množiny pravidel. Vždy tak pro DTDP M existuje nějaké $k \geq 1$, pro které platí, že $k = \max(\{m |\ mqA \rightarrow pv \in R_M\})$.

Tento nový zobecněný automat byl poprvé ústně prezentován a diskutován na konferenci WFM'06 (příspěvek [33]), kde byla nastíněna myšlenka důkazů následujících výsledků (věta 4.1), jež byly rigorózně dokázány v [50], a též několik zajímavých modifikací určených pro další zkoumání tohoto formálního modelu.

Na tomto místě je vhodné zdůraznit, že autor tohoto textu nepracoval přímo na vytvoření hlubokého zásobníkového automatu (viz [50]), ale spolupracoval až na některých jeho modifikacích.

Následující věta 4.1 je převzata z [50] kvůli ucelenosti výkladu.

Věta 4.1 ([50])**.** *Pro každé celé číslo $k \geq 1$ platí, že ${}_k\mathcal{L}(\mathbf{DTDP}) = {}_k\mathcal{L}(\mathbf{ST})$ a ${}_k\mathcal{L}(\mathbf{DTDP}) \subset {}_{k+1}\mathcal{L}(\mathbf{DTDP})$.*

Důkaz věty 4.1. Konstrukční i rigorózní část důkazu najdeme taktéž v [50]. Důkaz je veden se silným využitím analogie mezi omezením přepisování v DTDP na hloubku n a mezi omezením přepisování n-limitovaností ve stavové gramatice. Je v něm demonstrována ekvivalence těchto dvou modelů a z toho plynoucí nekonečná hierarchie jazyků, kterou dokázal Kasai pro n-limitované stavové gramatiky ([21]).

V článku [50] byly také nastíněny otevřené problémy a otázky ohledně determinismu a povolení vymazávajících pravidel.

4.2.1 Deterministické hluboké zásobníkové automaty

Přirozenou a matematicky elegantně zapsatelnou modifikací hlubokého zásobníkového automatu je definice determinismu vzhledem k hloubce expanzí. To znamená, že pokud se rozhodneme pro expanzi v konkrétní hloubce i, budeme mít nejvýše jedinou možnost, jaké pravidlo v aktuálním stavu vybrat. Tudíž v množině pravidel nepovolujeme například současnou existenci pravidla $r_1 \colon 1pA \rightarrow qx$ a pravidla $r_2 \colon 2pC \rightarrow oy$.

Definice 4.10. M je *deterministický hluboký zásobníkový automat vzhledem k hloubce expanzí*, platí-li pro každé $q \in Q$, $\mathrm{card}(\{m \mid mqA \rightarrow pv \in R,\ A \in \Gamma - T,\ v \in \Gamma^+,\ p \in Q\}) \leq 1$, protože ze stejného stavu mají všechny možné expanze automatu M stejnou hloubku. Dále řekneme, že hluboký zásobníkový automat je *silně deterministický*, pokud pro každé pravidlo $mqA \rightarrow pv \in R$ platí, že $\mathrm{card}(\{mqA \rightarrow ow \in R |\ o \in Q,\ w \in \Gamma^+\} - \{mqA \rightarrow pv\}) = 0$.

Poznamenejme, že M z příkladu 4.5 je deterministický vůči hloubce expanzí, ale není silně deterministický.

4.2.2 Redukující hluboké zásobníkové automaty

Dále studovaná varianta hlubokého zásobníkového automatu modifikuje základní způsob jeho chování. Hluboký zásobníkový automat popsaný v [33] a [50] byl založen na principu hluboce pracujícího analyzátoru shora-dolů. Přirozenou modifikací je tedy zavedení přístupu zdola-nahoru

do konceptu hlubokého zásobníkového automatu. Takto zmodifikovaný automat pak nazveme redukující hluboký zásobníkový automat (viz [35, 36]). Zajímavostí tohoto konceptu bude také studium obou směrů zpracování vstupní věty, tedy jak klasický směr zleva doprava, tak především zprava doleva.

V rámci výsledků pak získáme další nekonečnou hierarchii tříd jazyků založenou na tom, jak hluboko v zásobníku umožníme provádění redukcí.

Motivace a princip

V teorii formálních jazyků lze zachytit již několik případů zobecňování zásobníkových automatů ([40, 47, 50]). Jednou z nejpraktičtějších modifikací jsou případy, kdy přepisující systémy, ať již plně formální nebo pro praxi účelově upravené, umožňují pracovat nejen s vrcholem zásobníku, ale také se symboly hlouběji na zásobníku.

Z pohledu klasických zásobníkových automatů získáme jak v přístupu shora-dolů, tak v případě užití metody zdola-nahoru obecně ekvivalentní modely. Pokud se však omezíme pouze na deterministické bezkontextové gramatiky, konkrétně LL(k) gramatiky pro případ práce shora-dolů a LR(k) pro zdola-nahoru, zmíněná ekvivalence již neplatí a druhý model je silnější, a tudíž pro praxi cennější.

Jednou z hlavních motivací studia redukujících hlubokých zásobníkových automatů je otázka, jak tento vztah obou přístupů (shora-dolů a zdola-nahoru) ovlivní hloubka, ve které dané operace povolíme provádět?

Uvažujme klasickou simulaci bezkontextové gramatiky pomocí klasického zásobníkového automatu, který pracuje jako obecný analyzátor zdola-nahoru (viz [48]). Během každého kroku analyzátor buď přesune symbol vstupní pásky na zásobník (*shift*), nebo redukuje část obsahu zásobníku na jediný zásobníkový symbol (*reduction*). Která z akcí se provede, závisí na symbolu na vrcholu zásobníku, aktuálním symbolu na vstupní pásce a samozřejmě na aktuálním stavu. Operace *přesun* odejme symbol ze vstupu (z pozice čtecí hlavy) a vloží jej na vrchol zásobníku a nakonec o jeden symbol posune ukazatel čtecí hlavy na vstupní pásce. Druhou operací je *redukce* podřetězce na vrcholu zásobníku, který je přepsán podle odpovídajícího pravidla na jediný zásobníkový symbol, jenž nepatří mezi vstupní symboly. Proměnné jsou tedy v tomto přepisujícím systému všechny symboly zásobníkové abecedy, protože tvoří přepisovaný, resp. redukovaný podřetězec.

Automat pak přijímá vstupní řetězec x, pokud provede posloupnost kroků takových, že vedou ke kompletnímu přečtení x ze vstupní pásky, vyprázdnění zásobníku a zároveň vstoupí do koncového stavu. Poznamenejme, že některé knihy nevyžadují úspěšné skončení výpočtu v koncovém stavu (např. věta 5.1 v [64]).

Po inspiraci předchozími odstavci představme zobecněné verze analyzátorů shora-dolů (viz hluboký zásobníkový automat v sekci 4.2) a zdola-nahoru (nyní definovaný redukující hluboký zásobníkový automat).

Zobecněný analyzátor zdola-nahoru reprezentovaný zásobníkovým automatem pracuje stejně jako ten v základní verzi (viz definice 2.17), až na to, že:

a) čte vstupní pásku zprava doleva (definujeme i modifikaci čtoucí zleva doprava) a

b) provádí redukce zásobníku až v hloubce m, kde m je hloubka zásobníkového symbolu, na který byl podřetězec zásobníku redukován.

Dále jej budeme nazývat *zprava-doleva redukující hluboký zásobníkový automat* (*Right-to-Left Reducing Deep Pushdown Automaton*).

Definice 4.11. *Redukující hluboký zásobníkový automat* (zkráceně RDPDA) je šestice, $M = (Q, T, \Gamma, R, s, S, F)$, kde Q je konečná množina stavů, T je vstupní abeceda a Γ je zásobníková abeceda, $\mathbb{N}$, Q a Γ jsou po dvojicích navzájem disjunktní (pro definici $\mathbb{N}$ viz definice 2.1), $T \subseteq \Gamma$, $\Gamma - T$ obsahuje speciální symbol *dno zásobníku* označený $\#$, $R \subseteq (Q \times \Gamma^+ \times \mathbb{N} \times Q \times (\Gamma - T))$ je konečná relace, $s \in Q$ je počáteční stav, $S \in \Gamma$ je počáteční zásobníkový symbol, $F \subseteq Q$ je množina koncových stavů. Místo $(q, v, m, p, A) \in R$, píšeme $qv \vdash mpA \in R$ a $qv \vdash mpA$ nazýváme pravidlo. Podle toho také R označujeme jako množinu pravidel automatu M.

Konfigurace automatu M je trojice z $Q \times T^* \times (\Gamma - \{\#\})^*\{\#\}$, takže vrchol zásobníku je nejlevější symbol třetí komponenty konfigurace. Nechť χ označuje množinu všech konfigurací automatu M. Nechť $x, y \in \chi$ jsou dvě konfigurace a $p, q \in Q$ jsou dva stavy. M *přesouvá* symbol svého vstupu na zásobník, tedy přepisuje konfiguraci z x na y jedním ze dvou způsobů:

a) $x \,{}_s\!\Rightarrow y$, kde $x = (q, ua, z)$, $y = (q, u, az)$, kde $a \in T$, $u \in T^*$, $z \in \Gamma^*$. M pak nazýváme konkrétněji *zprava-doleva redukující* (zkrátka ${}^{rl}\mathbf{RDPDA}$).

b) $x \,{}_s\!\Rightarrow y$, kde $x = (q, au, z)$, $y = (q, u, az)$, kde $a \in T$, $u \in T^*$, $z \in \Gamma^*$. M pak nazýváme konkrétněji *zleva-doprava redukující* (zkrátka ${}^{lr}\mathbf{RDPDA}$).

M *redukuje* obsah svého zásobníku, tedy mění konfiguraci z x na y, symbolicky $x \,{}_r\!\Rightarrow y$, jestliže $x = (q, w, uvz)$ $y = (p, w, uAz)$ a $qv \vdash mpA \in R$, kde $w \in T^*$, $A \in (\Gamma - T - \{\#\})$, $u, z \in \Gamma^*$, $v \in \Gamma^+$ a $\mathrm{occur}(\Gamma - T, u) = m - 1$.

Abychom vyjádřili, když M provádí $x \,{}_r\!\Rightarrow y$ podle pravidla $qv \vdash mpA$, píšeme $x \,{}_r\!\Rightarrow y\ [qv \vdash mpA]$. Také říkáme, že $qv \vdash mpA$ je *pravidlo hloubky* m; podobně, $x \,{}_r\!\Rightarrow y\ [qv \vdash mpA]$ je *redukce hloubky* m. M provádí krok z x do y, symbolicky zapsáno jako $x \Rightarrow y$, jestliže M provede buď $x \,{}_s\!\Rightarrow y$, nebo $x \,{}_r\!\Rightarrow y$.

Je-li $n \in \mathbb{N}$ minimální celé kladné číslo takové, že každé pravidlo automatu M je hloubky n nebo nižší, říkáme, že M je hloubky n, což symbolicky zapisujeme jako ${}_nM$.

Standardním způsobem rozšiřme ${}_s\!\Rightarrow$, ${}_r\!\Rightarrow$ a $\Rightarrow$ na ${}_s\!\Rightarrow^m$, ${}_r\!\Rightarrow^m$ a $\Rightarrow^m$, pro libovolné $m \geq 0$; pak na základě ${}_s\!\Rightarrow^m$, ${}_r\!\Rightarrow^m$ a $\Rightarrow^m$ definujme ${}_s\!\Rightarrow^+$, ${}_r\!\Rightarrow^+$, $\Rightarrow^+$, ${}_s\!\Rightarrow^*$, ${}_r\!\Rightarrow^*$, $\Rightarrow^*$.

Nechť M je hloubky n, pro nějaké $n \in \mathbb{N}$. Definujme jazyk přijímaný automatem ${}_nM$, $L({}_nM)$, jako $L({}_nM) = \{w \in T^* |\ (s, w, \#) \Rightarrow^* (f, \varepsilon, S\#)$ v ${}_nM$ s $f \in F\}$.

Pro každé $k \geq 1$, ${}_k\mathcal{L}(\mathbf{RDPDA})$, resp. ${}_k\mathcal{L}({}^{\mathbf{lr}}\mathbf{RDPDA})$ označuje třídu jazyků definovaných zprava-doleva, resp. zleva-doprava redukujícími hlubokými zásobníkovými automaty hloubky i, kde $1 \leq i \leq k$.

Následující příklad demonstruje redukci věty automatem ${}^{rl}\mathbf{RDPDA}$.

Příklad 4.6. Mějme ${}^{rl}\mathbf{RDPDA}$, ${}_2M = (\{s, p, q, t, f\}, \{a, b, c\}, \{a, b, c, A, C, S\}, R, s, S, \{f\})$ s množinou pravidel R:

1. $sab \vdash 1pA$
2. $pc \vdash 2qC$
3. $qaAb \vdash 1tA$
4. $tcC \vdash 2qC$
5. $qAC \vdash 1fS$

${}_2M$ provede při načítání vstupu *aabbcc* ve směru zprava-doleva následující posloupnost přesunů a redukcí:

$(s, aabbcc, \#) \ {}_s\Rightarrow (s, aabbc, c\#)$ ${}_s\Rightarrow (s, aabb, cc\#)$ ${}_s\Rightarrow (s, aab, bcc\#)$ ${}_s\Rightarrow (s, aa, bbcc\#)$ ${}_s\Rightarrow (s, a, abbcc\#)$ ${}_r\Rightarrow (p, a, Abcc\#)$ [1] ${}_r\Rightarrow (q, a, AbcC\#)$ [2] ${}_s\Rightarrow (q, \varepsilon, aAbcC\#)$ ${}_r\Rightarrow (t, \varepsilon, AcC\#)$ [3] ${}_r\Rightarrow (q, \varepsilon, AC\#)$ [4] ${}_r\Rightarrow (f, \varepsilon, S\#)$ [5].

Tímto jsme ukázali, že $aabbcc \in L({}_2M)$. Všimněme si také, že $L({}_2M) = \{a^n b^n c^n |\ n \leq 1\}$ a poznamenejme, že $L({}_2M) \in \mathbf{CS} - \mathbf{CF}$.

4.3 Zásobníkový automat s omezeným zásobníkem

V této sekci definujeme nový formální model modifikací klasického zásobníkového automatu (viz definice 2.17)—zásobníkový automat s omezeným obsahem zásobníku. Obsah zásobníku je omezován tzv. *omezujícím jazykem*. Tento obecný způsob omezování konfigurace přepisujícího systému se tedy vztahuje konkrétně na první komponentu konfigurace (viz definice 2.18). Během aplikace pravidla v tomto modelu je prováděna kontrola, zda obsah zásobníku tvoří větu omezujícího jazyka. V případě, že by dokončení aplikace tohoto pravidla měnilo obsah zásobníku tak, že by netvořil větu omezujícího jazyka, nelze pravidlo použít. Na klasifikaci omezujícího jazyka v Chomského hierarchii pak závisí výsledná mocnost omezovaného automatu.

Ve výsledcích ukážeme, že pokud je tento omezující jazyk regulární, model nepřekročí sílu bezkontextových jazyků (věta 5.13). Dále je demonstrováno, že již v případě lineárního omezujícího jazyka získáváme větší vyjadřovací sílu (příklad 4.7).

Definice 4.12. *Zásobníkový automat s omezeným obsahem zásobníku* (*Pushdown Automaton with Restricted Pushdown-Content*, zkráceně RCPDA) je dvojice $H = (M, \Xi)$, kde $M = (Q, T, \Gamma, R, s, S, F)$ je klasický zásobníkový automat (viz definice 2.17) a $\Xi \subseteq \Gamma^*$ je tzv. *omezující jazyk*.

Definice konfigurace RCPDA H je totožná s klasickým zásobníkovým automatem, takže je třeba pouze definovat přechodovou relaci (krok) a jazyk přijímaný tímto novým přepisujícím systémem.

Definice 4.13. Nejprve definujme množinu všech možných obsahů zásobníku automatu $H = (M, \Xi)$, $M = (Q, T, \Gamma, R, s, S, F)$, při přijímání věty w jako $K(H, w) = \{\gamma \mid w \in T^*,\ w \in L(M)$, posloupnost přechodů $(S, s, w) \Rightarrow^* (\gamma, q, u) \Rightarrow^* (\gamma_F, q_F, \varepsilon)$, $u \in \text{suffixes}(w)$, $q \in Q$, $q_F \in F$ a $\gamma, \gamma_F \in \Gamma^*\}$.

$K(H, w)$ obsahuje všechny řetězce, které se vyskytly na zásobníku během všech možných cest při přijímání řetězce w automatem M.

Definice 4.14. Jazyk přijímaný zásobníkovým automatem s omezeným obsahem zásobníku, $H = (M, \Xi)$,

$$L(H) = \{w \mid w \in L(M),\ K(H, w) \subseteq \Xi\}.$$

Třídu jazyků přijímaných RCPDA, kde řídící jazyk patří do třídy jazyků $\mathcal{L}(\mathbf{X})$ budeme značit $\mathcal{L}(\mathbf{RCPDA}, X)$.

Poznámka 4.2. Podobně jako v případě definice konečného indexu u #-přepisujících systémů, i zde je možno provést definici $L(H)$ volnějším způsobem, který neomezuje všechny cesty k větě, ale vyžaduje existenci alespoň jedné, která splňuje podmínku omezování obsahu zásobníku omezujícím jazykem Ξ. Slabší definice jazyka přijímaného RCPDA $H = (M = (Q, T, \Gamma, R, s, S, F), \Xi)$, $L(H) = \{w|$ existuje posloupnost přechodů $(S, s, w) = (\gamma_0, q_0, u_0) \Rightarrow_M (\gamma_1, q_1, u_1) \Rightarrow_M \ldots \Rightarrow_M (\gamma_n, q_n, u_n) = (\varepsilon, q_F, \varepsilon)$ taková, že $n \geq 0$, $\gamma_0, \ldots, \gamma_n \in \Xi$, $q_F \in F$, $u_i = \text{suffix}(w, |w| - i)$ pro všechna $1 \leq i \leq n\}$.

Příklad 4.7. Mějme zásobníkový automat $M = (Q, T, T \cup \{\#, \Delta\}, R, \#, s, \{f\})$, kde $Q = \{s, p, f\}$, $T = \{a, b, c\}$ a množina pravidel R obsahuje pravidla:

1. $\#s\varepsilon \to \#cs$
2. $cs\varepsilon \to ccs$
3. $csa \to cas$
4. $asa \to aas$
5. $as\varepsilon \to a\Delta s$
6. $\Delta s\varepsilon \to \varepsilon q$
7. $aqb \to \varepsilon q$
8. $cqc \to \varepsilon q$
9. $\#q\varepsilon \to \varepsilon f$.

Omezující lineární jazyk, $L(G)$, zadaný gramatikou $G = (\Sigma, T, P, S)$, $\Sigma = N \cup T$, kde $N = \{S, X, Y\}$, $T = \{a, c, \#, \Delta\}$ a množina pravidel P obsahuje:

1. $S \to \#cX$
2. $S \to \#$
3. $S \to \varepsilon$
4. $S \to \#cYa\Delta$
5. $X \to cXa$
6. $X \to cX$
7. $X \to \varepsilon$
8. $Y \to cYa$
9. $Y \to \varepsilon$.

Je zřejmé, že G je lineární gramatika, protože každé její pravidlo obsahuje na pravé straně nejvýše jeden neterminální symbol, tedy $L(G) \in \mathbf{LIN}$. $L(G) = \{\varepsilon\} \cup \{\#c^n a^m |\ m, n \geq 0, m \leq n\} \cup \{\#c^n a^n \Delta |\ n \geq 1\}$.

Definujeme-li zásobníkový automat s omezeným obsahem zásobníku $H = (M, L(G))$, zjistíme po delším studiu, že $L(H) = \{a^n b^n c^n |\ n \geq 1\}$.

Skutečně, 1. a 2. pravidlo z R nedeterministicky vygenerují potřebný počet symbolů c; 3. a 4. pravidlo provádí čtení a přesun symbolů a ze vstupu na zásobník; 5. pravidlo nedeterministicky rozhodne, že jsou přečteny všechny symboly a a provede vložení Δ na zásobník. V tuto chvíli se dostává do hry podjazyk $\{\#c^n a^n \Delta |\ n \geq 1\}$ jazyka $L(G)$, který obsahuje na konci symbol Δ, a bude tak zajištěn stejný počet symbolů a i c na zásobníku. Pak 6. pravidlo symbol Δ zase odstraní; 7. a 8. pravidlo přečte ze vstupu symboly b a pak symboly c a k tomu odstraní ze zásobníku vždy odpovídající symbol a a pak c. Není-li již co číst a zásobník je (až na dno) prázdný, poslední pravidlo přejde do koncového stavu f a zcela vymaže zásobník.

Z praktického hlediska je automat opravdu nějak omezován pouze v kroku pomocí pátého pravidla, kdy je vyžádáno splnění podmínky o stejném počtu a i c. V ostatních krocích automatu M se omezování reálně neuplatňuje. To nás může vést v praxi k takové modifikaci, kdy omezování budume aplikovat pouze na některé kroky výpočtu.

4.4 Shrnutí

Nyní si pouze tabulkově zopakujme nejpodstatnější zavedené přepisující systémy (viz tabulka 4.2).

Název (anglicky)	**Zkratka**	**Třída jazyků**
Right-Linear Grammars	RLIN	**REG**
Context-Free Grammars	CF	**CF**
Context-Sensitive Grammars	CS	**CS**
#-Rewriting Systems	CF#RS	$\mathcal{L}(\mathbf{CF\#RS})$
Generalized #-Rewriting Systems	G#RS	$\mathcal{L}(\mathbf{G\#RS})$
n-Right-Linear #-Rewriting Systems	n-RLIN#RS	$\mathcal{L}(n\text{-}\mathbf{RLIN\#RS})$
Pushdown Automata with Restricted Pushdown-Content	RCPDA	$\mathcal{L}(\mathbf{RCPDA}, X)$
Deep Pushdown Automata	DTDP	$\mathcal{L}(\mathbf{DTDP})$
Reducing Deep Pushdown Automata	RDPDA	$\mathcal{L}(\mathbf{RDPDA})$
State Grammars	ST	$\mathcal{L}(\mathbf{ST})$
Programmed Grammars	PG	$\mathcal{L}(\mathbf{P})$
Random-context Grammars	RC	$\mathcal{L}(\mathbf{RC})$

Tabulka 4.2: Přehled nejdůležitějších přepisujících systémů

Kapitola 5

Výsledky

Tato kapitola popisuje a hlavně dokazuje všechny nejdůležitější výsledky vzniklé při studiu omezování přepisujících systémů se zaměřením na jejich nové typy. Výsledky lze rozdělit do dvou sekcí: (1) vyjadřovací síla přepisujících systémů, (2) nekonečné hierarchie tříd jazyků založené na omezování přepisujících systémů.

Konvence 5.1. Všechny originální výsledky (věty a lemmata) této práce jsou dokázány prostřednictvím konstrukčních důkazů. Konstrukční důkazy by měly být správně ještě kompletovány rigorózním indukčním důkazem, který by demonstroval správnost konstrukce. Tyto důkazy jsou však v některých případech vypuštěny a v případě zájmu ponechány na laskavém čtenáři.

5.1 Vyjadřovací síla omezovaných přepisujících systémů

Nyní si představíme vyjadřovací sílu #-přepisujících systémů s různými tvary pravidel s omezením konfigurací na konečný index. To znamená, že pro nějaké $k \geq 1$ bude každá konfigurace obsahovat nejvýše k hranic.

5.1.1 Bezkontextové #-přepisující systémy

Nyní demonstrujme, že #-přepisující systémy založené na bezkontextových pravidlech konečného indexu charakterizují právě dobře známou nekonečnou hierarchii jazyků, která je definovaná například programovanými gramatikami konečného indexu (viz věty 3.1.2i a 3.1.7 v knize [12]).

Z širší perspektivy tento výsledek dokazuje, že přepisující systémy založené na kombinaci gramatik a automatů mají přirozený vztah ke klasickým tématům a výsledkům z oblasti formálních jazyků, takže na ně mohou poskytnout alternativní úhel pohledu.

Podobně jako většina důkazů ekvivalence dvou formálních modelů provedeme demonstraci obou směrů vzájemné inkluze těchto tříd jazyků, které odpovídají vyjadřovací síle porovnávaných modelů. Informatice nejbližší způsob je konstrukce algoritmu pro převod mezi těmito modely tak, aby byl zachován stejný přijímaný jazyk, resp. třída jazyků.

Základní myšlenka konstrukce má pak původ v alternativní verzi důkazu o rovnosti $\mathcal{L}_{fin}(\mathbf{RC}) = \mathcal{L}_{fin}(\mathbf{P})$ (viz [32]).

Lemma 5.1. *Pro každé $k \geq 1$, $\mathcal{L}_k(\mathbf{P}) \subseteq \mathcal{L}_k(\mathbf{CF\#RS})$.*

Konstrukční důkaz lemmatu 5.1. Nechť $k \geq 1$ je kladné celé číslo. Nechť $G = (\Sigma, T, P, S)$ je programovaná gramatika indexu k, kde $N = \Sigma - T$. Vytvoříme #-přepisující systém indexu k,

$H = (Q, T \cup \{\#\}, s, R)$, kde $\# \notin T$, $s = \langle\sigma\rangle$, σ je nový symbol, a R a Q jsou zkonstruovány podle posloupnosti následujících kroků:

1. Pro každé pravidlo $\lfloor p\colon S \to \alpha, g(p)\rfloor \in P$, $\alpha \in \Sigma^*$, přidejme $\langle\sigma\rangle_1\# \to \langle[p]\rangle\#$ do R, kde $\langle[p]\rangle$ je nový stav v Q a $g(p)$ bude značit množinu následných pravidel pravidla p.
2. Je-li $A_1A_2 \ldots A_j \ldots A_h \in N^*$, $h \in \{1, 2, \ldots, k\}$, $\lfloor p\colon A_j \to x_0B_1x_1B_2x_2 \ldots x_{n-1}B_nx_n, g(p)\rfloor \in P$, $j \in \{1, 2, \ldots, h\}$ pro $n \geq 0$, $x_0, x_t \in T^*$, $B_t \in N$, $1 \leq t \leq n$ a $n + h - 1 \leq k$, pak
   (a) pokud $g(p) = \emptyset$, tak $\langle A_1A_2 \ldots A_{j-1}[p]A_{j+1} \ldots A_h\rangle$, $\langle A_1A_2 \ldots B_1 \ldots B_n \ldots A_h\rangle$ jsou nové stavy v Q a pravidlo $\langle A_1A_2 \ldots A_{j-1}[p]A_{j+1} \ldots A_h\rangle_j\# \to \langle A_1\ A_2 \ldots B_1 \ldots B_n \ldots A_h\rangle$ $x_0\#x_1 \ldots x_{n-1}\#x_n$ přidejme do R;
   (b) pro každé $q \in g(p)$, $q\colon D_d \to \alpha$, $\alpha \in \Sigma^*$ přidejme nové stavy $\langle A_1A_2 \ldots A_{j-1}[p]\ A_{j+1} \ldots A_h\rangle$ a $\langle D_1D_2 \ldots D_{d-1}[q]D_{d+1} \ldots D_{n+h-1}\rangle$ do Q a do R přidejme následující pravidlo: $\langle A_1A_2 \ldots A_{j-1}[p]A_{j+1} \ldots A_h\rangle_j\# \to \langle D_1D_2 \ldots D_{d-1}[q]D_{d+1} \ldots D_{n+h-1}\rangle$ $x_0\#x_1 \ldots x_n$, kde až na $D_d = [q]$ platí $A_1 \ldots A_{j-1}B_1 \ldots B_nA_{j+1} \ldots A_h = D_1 \ldots D_{h+n-1}$, $B_1 \ldots B_n = D_j \ldots D_{j+n-1}$, pro nějaké $d \in \{1, 2, \ldots, n + h - 1\}$.

Hlavní myšlenka důkazu lemmatu 5.1. H simuluje derivaci programované gramatiky G. Informace nutné pro simulaci jsou zaznamenány uvnitř stavů nových pravidel v H. Každý stav v Q obsahuje řetězec neterminálů z N^*, kde jeden ze symbolů v tomto řetězci je nahrazen návěštím pravidla z P (v hranatých závorkách).

Nechť $x_0A_1x_1 \ldots x_{h-1}A_hx_h$ je větná forma derivovaná pomocí G, kde $x_i \in T^*$ pro $0 \leq i \leq h$ a $A_l \in \Sigma - T$ pro $1 \leq l \leq h$, a nechť $\lfloor p\colon A_j \to \alpha, g(p)\rfloor$ je pravidlo v P aplikovatelné v dalším kroku na neterminál A_j, $1 \leq j \leq h$. Pak, nová konfigurace systému H je tvaru $\langle A_1A_2 \ldots A_{j-1}[p]A_{j+1} \ldots A_h\rangle x_0\#x_1 \ldots x_{h-1}\#x_h$, kde jsou zapamatovány a reflektovány neterminály z odpovídající větné formy z G i návěští pravidla k následné aplikaci. Přepis pro simulaci p nahradí j-tou hranici řetězcem α a změní stav tak, aby reflektoval korespondenci neterminálů z G a vzniklých hranic na místě j-té hranice. Změna stavu také odráží nedeterministický předběžný výběr pravidla z $g(p)$ pro pokračování simulace.

Rigorózní důkaz:

Tvrzení 5.1.1. *Jestliže* $S \Rightarrow^m x_0A_1x_1A_2x_2 \ldots x_{h-1}A_hx_h$ *v* G, *pak* $\langle\sigma\rangle\# \Rightarrow^r \langle A_1A_2 \ldots A_h\rangle$ $x_0\#x_1 \ldots x_h$ $[q_1q_2 \ldots q_r]$ *v* H, *pro* $m \geq 0$. *Je-li* $g(q_r) \neq \emptyset$, *pak existuje pravidlo* $\lfloor q_{r+1}\colon A_j \to y_0B_1y_1 \ldots y_{h-1}B_ny_n, g(q_{r+1})\rfloor$, *kde* $n + h - 1 \leq k$, $q_{r+1} \in g(q_r)$, $A_j = [q_{r+1}]$ *a* $q_1, \ldots, q_r, q_{r+1} \in \mathrm{Lab}(R)$.

Důkaz tvrzení 5.1.1. Toto tvrzení je dokázáno indukcí podle m.

Základ indukce: Nechť $m = 0$. Pro $S \Rightarrow^0 S$ v G existuje $\langle\sigma\rangle\# \Rightarrow^1 \langle[p]\rangle\#$ v H, kde $\lfloor p\colon S \to \alpha, g(p)\rfloor \in P$ a $\langle\sigma\rangle_1\# \to \langle[p]\rangle\# \in R$.

Indukční hypotéza: Předpokládejme, že tvrzení 5.1.1 platí pro všechny výpočty délky m nebo méně pro libovolné $m \geq 0$.

Indukční krok: Uvažujme $S \Rightarrow^m y$ $[p_1p_2 \ldots p_m]$, kde $y = x_0A_1x_1 \ldots x_{n-1}A_hx_h$ a $p_1, \ldots, p_m, p_{m+1} \in \mathrm{Lab}(P)$ takovou, že $y \Rightarrow x$ $[p_{m+1}]$. Je-li $m = 0$, pak $p_{m+1} \in \{p \mid lhs(p) = S, p \in \mathrm{Lab}(P)\}$; jinak $p_{m+1} \in g(p_m)$. Pro $\lfloor p_{m+1}\colon A_j \to y_0B_1y_1 \ldots y_{n-1}B_ny_n, g(p_{m+1})\rfloor$ je x ve tvaru: $x = x_0A_1x_1 \ldots A_{j-1}x_{j-1}y_0B_1y_1 \ldots y_{n-1}B_ny_nx_jA_{j+1} \ldots x_{h-1}A_hx_h$, kde $x_0, \ldots, x_h \in T^*$ a $y_0, \ldots, y_n \in T^*$. Podle indukční hypotézy existuje výpočet $\langle\sigma\rangle\# \Rightarrow^r$ $\langle A_1A_2 \ldots A_{j-1}[p_{m+1}]A_{j+1} \ldots A_h\rangle x_0\#x_1 \ldots x_{h-1}\#x_h$ $[q_1q_2 \ldots q_r] \Rightarrow$
$\langle A_1A_2 \ldots A_{j-1}B_1 \ldots B_nA_{j+1} \ldots A_h\rangle x_0\# \ldots \#x_{j-1}y_0\# \ldots \#y_nx_j\# \ldots \#x_h$ $[q_{r+1}]$, $r \geq 1$, $q_i \in$

Lab(R), $1 \leq i \leq r+1$. Je-li $g(p_{m+1}) \neq \emptyset$, pak existuje pravidlo $p_{m+2} \in g(p_{m+1})$ a řetězec $D_1 D_2 \ldots D_{n+h-1}$ takový, že $A_1 A_2 \ldots A_{j-1} B_1 \ldots B_n A_{j+1} \ldots A_h = D_1 D_2 \ldots D_{n+h-1}$, kde pro nejvýše jedno $d \in \{1, 2, \ldots, n+h-1\}$ je $D_d = [q_{r+2}]$, $q_{r+2} \in g(q_{r+1})$.

Tvrzení 5.1.2. *Jestliže $S \Rightarrow^z x \ v \ G$, pak $\langle\sigma\rangle\# \Rightarrow^* \langle\rangle x \ v \ H$, kde $z \geq 0$ a $x \in T^*$.*

Důkaz tvrzení 5.1.2. Uvažujme platné tvrzení 5.1.1 pro $h = 0$. V takovém případě, jestliže $S \Rightarrow^z x_0$, pak $\langle\sigma\rangle\# \Rightarrow^* \langle\rangle x_0$. Nyní již jen položme $x_0 = x$.

Tvrzení 5.1.1 a 5.1.2 tedy formálně dokazují lemma 5.1. □

Lemma 5.2. *Pro každé $k \geq 1$, $\mathcal{L}_k(\mathbf{CF\#RS}) \subseteq \mathcal{L}_k(\mathbf{P})$.*

Konstrukční důkaz lemmatu 5.2. Nechť $k \geq 1$ je kladné celé číslo. Nechť $H = (Q, T \cup \{\#\}, s, R)$ je #-přepisující systém indexu k, kde $\Sigma = T \cup \{\#\}$ a $T \cap \{\#\} = \emptyset$. Vytvořme programovanou gramatiku indexu k, $G = (\Sigma, T, P, S)$, kde množina neterminálů $N = \Sigma - T$ a množina pravidel P jsou zkonstruovány následovně:

1. $S = \langle s, 1, 1\rangle$;
2. $N = \{\langle p, i, h\rangle \mid p \in Q,\ 1 \leq i \leq h,\ i \leq h \leq k\} \cup \{\langle q', i, h\rangle \mid q \in Q,\ 1 \leq i \leq h,\ i \leq h \leq k\}$ $\cup \{\langle q'', i, h\rangle \mid q \in Q,\ 1 \leq i \leq h,\ i \leq h \leq k\} \cup \{\langle q'', 1, 0\rangle \mid q \in Q\}$;
3. Pro každé pravidlo $r\colon p_i\# \to qy \in R$, $y = y_0\#y_1 \ldots y_{m-1}\#y_m$, $y_0, y_1, y_2 \ldots y_m \in T^*$, přidejme následující množinu do P:

   (i) $\{\langle p, j, h\rangle \to \langle q', j, h+m-1\rangle,$
   $\{r' \mid$ je-li $j+1 = i$, pak $r'\colon \langle p, i, h\rangle \to \langle q'', i, h+m-1\rangle$, jinak $r'\colon \langle p, j+1,$
   $h\rangle \to \langle q', j+1, h+m-1\rangle\ \}$
   $\mid 1 \leq j < i,\ i \leq h \leq h_{max}\}$
   $\cup$

   (ii) $\{\langle p, i, h\rangle \to \langle q'', i, h+m-1\rangle,$
   $\{r' \mid$ je-li $i = h$, pak $r'\colon \langle q'', i, h+m-1\rangle \to y_0\langle q', i, h+m-1\rangle y_1\langle q', i+1, h+$
   $m-1\rangle y_2 \ldots y_{m-1}\langle q', i+m-1, h+m-1\rangle y_m$, jinak $r'\colon \langle p, i+1, h\rangle \to$
   $\langle q', i+1+m-1, h+m-1\rangle\}$
   $\mid i \leq h \leq h_{max}\}$
   $\cup$

   (iii) $\{\langle p, j, h\rangle \to \langle q', j+m-1, h+m-1\rangle,$
   $\{r' \mid$ je-li $j = h$, pak $r'\colon \langle q'', i, h+m-1\rangle \to y_0\langle q', i, h+m-1\rangle y_1\langle q', i+1,$
   $h+m-1\rangle y_2 \ldots y_{m-1}\langle q', i+m-1, h+m-1\rangle y_m$, jinak $r'\colon \langle p, j+$
   $1, h\rangle \to \langle q', j+1+m-1, h+m-1\rangle\}$
   $\mid i < j \leq h,\ i \leq h \leq h_{max}\}$
   $\cup$

   (iv) $\{\langle q'', i, h+m-1\rangle \to y_0\langle q', i, h+m-1\rangle y_1\langle q', i+1, h+m-1\rangle y_2 \ldots y_{m-1}\langle q', i+$
   $m-1, h+m-1\rangle y_m,$
   $\{r' \mid r'\colon \langle q', 1, h+m-1\rangle \to \langle q, 1, h+m-1\rangle\}$
   $\mid i \leq h \leq h_{max}\}$
   $\cup$

   (v) $\{\langle q', j, h+m-1\rangle \to \langle q, j, h+m-1\rangle,$
   $\{r' \mid$ je-li $j < h+m-1$, pak $r'\colon \langle q', j+1, h+m-1\rangle \to \langle q, j+1, h+m-1\rangle,$
   jinak $r'\colon \langle \tilde{p}, 1, h+m-1\rangle \to \langle \tilde{q}', 1, h+m-1+\tilde{m}-1\rangle$, kde
   $\tilde{p}_{\tilde{i}}\# \to \tilde{q}\tilde{y}_0\#\tilde{y}_1 \ldots\ \tilde{y}_{\tilde{m}-1}\#\tilde{y}_{\tilde{m}} \in R$, $\tilde{y}_0, \tilde{y}_1, \ldots, \tilde{y}_{\tilde{m}} \in T^*$, je-li $\tilde{i} = 1$,

$$\text{pak } \tilde{q}' := \tilde{q}''\}$$
$$\mid 1 \leq j \leq h + m - 1,\ i \leq h \leq h_{max}\},$$

kde $h_{max} = k$ pokud $m = 0$; jinak $h_{max} = k - m + 1$.

Hlavní myšlenka důkazu lemmatu 5.2. G simuluje pomocí několika derivačních kroků jediný krok v H. Informace nutné pro přesnou simulaci uchováváme v neterminálních symbolech. Uvnitř každého neterminálu tvaru $\langle p, i, h\rangle$, který se může vyskytovat ve větné formě vygenerované gramatikou G, zaznamenáváme tyto informace

(1) p—aktuální stav systému H;

(2) i—pozici výskytu # v aktuální konfiguraci systému H;

(3) h—celkový počet všech # v aktuální konfiguraci.

Z těchto tří informací a z množiny $g(p)$ vázající se k p, najdeme, zda je p aplikovatelné v dalším kroku a pokud ano, simulujeme výpočetní krok pravidly, která jsme vytvořili ve třetím bodě předchozí konstrukce. Simulace se skládá ze čtyř částí:

(a) uvnitř všech neterminálů větné formy změňme komponentu pro celkový počet hranic, potažmo neterminálů, z hodnoty h na $h+m-1$, kde m je počet neterminálů vyskytujících se na pravé straně pravidla p, takže $h + m - 1$ je celkový počet výskytů neterminálů po aplikaci pravidla p (viz pravidla konstruovaná v (i) až (iii));

(b) v neterminálech, které následují za přepsaným neterminálem, změňme jejich aktuální pozici (druhá komponenta), aby odpovídala stavu po aplikaci pravidla p (viz konstrukční krok (iii));

(c) aplikujme p a vyberme pravidlo (respektive návěští) q z $g(p)$, které bude použito v následné simulaci dalšího kroku v H (viz (iv));

(d) dokončeme simulovaný výpočetní krok v H pravidly zavedenými konstrukčním bodem (v).

Rigorózní důkaz:

Tvrzení 5.2.1. *Jestliže* $\langle\sigma\rangle\# \Rightarrow^c \langle\vartheta\rangle y_0\#y_1 \dots y_{n-1}\#y_n$ *v* H*, pak* $S \Rightarrow^* y_0A_1y_1 \dots y_{n-1}A_ny_n$ *v* G *pro libovolné* $c \geq 0$.

Důkaz tvrzení 5.2.1. Toto tvrzení dokažme indukcí podle c.

Základ indukce: Nechť $c = 0$. Pro $\langle\sigma\rangle\# \Rightarrow^0 \langle\sigma\rangle\#$ v H existuje $S \Rightarrow^0 S$ v G.

Indukční hypotéza: Předpokládejme, že tvrzení 5.2.1 platí pro všechny výpočetní kroky délky c nebo nižší pro libovolné $c \geq 0$.

Indukční krok: Uvažujme $\langle\sigma\rangle\# \Rightarrow^c \langle\vartheta\rangle y_0\#y_1 \dots y_h\ [r_1r_2 \dots r_c]$ v H, $r_t \in \text{Lab}(R)$, $1 \leq t \leq c$ a r_{c+1}: $\langle\vartheta\rangle_i\# \to \langle\omega\rangle x_0\#x_1 \dots x_{m-1}\#x_m \in R$, $x_0, \dots, x_m \in T^*$ takovou, že $\langle\vartheta\rangle y_0\# \dots \#y_h \Rightarrow \langle\omega\rangle y_0\#y_1\# \dots \#y_{i-1}x_0\#x_1\# \dots \#x_my_i\#y_{i+1}\# \dots \#y_h\ [r_{c+1}]$. Podle tvrzení 5.2.1 existuje také derivace $D_{1*} : y_0A_1 \dots A_hy_h \Rightarrow^* y_0A_1y_1 \dots y_{i-1}x_0B_1x_1 \dots B_mx_my_iA_{i+1} \dots A_hy_h$ v G.

Nyní si na základě konstrukční části důkazu ukažme, že taková derivace existuje.

Mějme větnou formu $y_0A_1y_1 \dots A_hy_h$. Přejmenujme neterminály A_t na $\langle\vartheta, t, h\rangle$ pro $1 \leq t \leq h$ a získejme základní větnou formu $y_0\langle\vartheta, 1, h\rangle y_1 \dots y_{h-1}\langle\vartheta, h, h\rangle y_h$, na které začíná simulace derivace D_{1*}. Tato simulace musí vycházet z probíhající aplikace pravidel vzniklých ve třetím konstrukčním kroku.

(3i) $\forall j : 1 \leq j < i$ aplikujme pravidla tvaru $\langle p, j, h\rangle \rightarrow \langle q', j, h+m-1\rangle$:
$F_1 = y_0\langle\vartheta, 1, h\rangle y_1 \dots y_{h-1}\langle\vartheta, h, h\rangle y_h \Rightarrow y_0\langle\omega', 1, h+m-1\rangle y_1\langle\vartheta, 2, h\rangle y_2 \dots y_{h-1}$
$\langle\vartheta, h, h\rangle y_h \Rightarrow^{i-2} y_0\langle\omega', 1, h+m-1\rangle y_1 \dots y_{i-2}\langle\omega', i-1, h+m-1\rangle y_{i-1}\langle\vartheta, i, h\rangle$
$y_i \dots y_{h-1}\langle\vartheta, h, h\rangle y_h = F_2$

(3ii) aplikujme $\langle p, i, h\rangle \rightarrow \langle q'', i, h+m-1\rangle$:
$F_2 \Rightarrow y_0\langle\omega', 1, h+m-1\rangle y_1 \dots y_{i-1}\langle\omega'', i, h+m-1\rangle y_i \dots y_{h-1}\langle\vartheta, h, h\rangle = F_3$.
Je-li $i = h$, pak položme $F_4 = F_3$ a pokračujme s (3iv), jinak pokračujme s (3iii).

(3iii) $\forall j : i < j \leq h$ aplikujme pravidla tvaru $\langle p, j, h\rangle \rightarrow \langle q', j+m-1, h+m-1\rangle$:
$F_3 \Rightarrow y_0\langle\omega', 1, h+m-1\rangle y_1 \dots y_{i-2}\langle\omega', i-1, h+m-1\rangle y_{i-1}\langle\omega'', i, h+m-1\rangle y_i$
$\langle\omega', i+m, h+m-1\rangle\ y_{i+1}\langle\vartheta, i+2, h\rangle y_{i+2} \dots y_{h-1}\langle\vartheta, h, h\rangle y_h \Rightarrow^{h-i-1} y_0\langle\omega', 1, h+m-1\rangle$
$y_1 \dots y_{i-1}\langle\omega'', i, h+m-1\rangle y_{i+1} \dots y_{h-1}\langle\omega', h+m-1, h+m-1\rangle y_h = F_4$

(3iv) aplikujme $\langle q'', i, h+m-1\rangle \rightarrow y_0\langle q', i, h+m-1\rangle y_1 \dots y_{m-1}\langle q', i+m-1, h+m-1\rangle y_m$:
$F_4 \Rightarrow y_0\langle\omega', 1, h+m-1\rangle y_1 \dots y_{i-1}x_0\langle\omega', i, h+m-1\rangle x_1 \dots x_{m-1}\langle\omega', i+m-1, h+m-1\rangle x_m y_i \dots y_{h-1}\langle\omega', h+m-1, h+m-1\rangle y_h = F_5$

(3v) $\forall j : 1 \leq j \leq h+m-1$ aplikujme pravidla tvaru $\langle q', j, h+m-1\rangle \rightarrow \langle q, j, h+m-1\rangle$:
$F_5 \Rightarrow^{h+m-1} y_0\langle\omega, 1, h+m-1\rangle y_1 \dots y_{i-1}x_0\langle\omega, i, h+m-1\rangle x_1 \dots x_{m-1}\langle\omega, i+m-1, h+m-1\rangle x_m y_i \dots y_{h-1}\langle\omega, h+m-1, h+m-1\rangle y_h = F_6$ (výsledná forma)

Nyní zpětně přejmenujme všechny neterminály tvaru $\langle\omega, t, h+m-1\rangle$ ve výsledné formě F_6 na A_t pro $1 \leq t < i$, $\langle\omega, t, h+m-1\rangle$ na B_{t-i+1} pro $i \leq t \leq i+m-1$, a $\langle\omega, t, h+m-1\rangle$ na A_{t-m+1} pro $i+m \leq t \leq h+m-1$. Takto jsme dostali $y_0A_1y_1 \dots y_{i-1}x_0B_1x_1 \dots B_mx_my_iA_{i+1} \dots A_hy_h$.

Tvrzení 5.2.2. *Jestliže $\langle\sigma\rangle\# \Rightarrow^z \langle\rangle x$ v H, pak $S \Rightarrow^* x$ pro libovolné $z \geq 0$.*

Důkaz tvrzení 5.2.2. Toto tvrzení jednoduše plyne z tvrzení 5.2.1 pro $n = 0$.

Lemma 5.2 je tímto pomocí Tvrzení 5.2.1 a 5.2.2 formálně dokázáno.

□

Věta 5.3. *Pro každé celé číslo $k \geq 1$ platí, že $\mathcal{L}_k(\mathbf{CF\#RS}) = \mathcal{L}_k(\mathbf{P})$.*

Důkaz věty 5.3. Věta 5.3 přímo vyplývá ze dvou předchozích lemmat 5.1 a 5.2.

■

Tímto jsme dokázali ekvivalenci CF#RS a programovaných gramatik při aplikaci omezování konfigurací na stejný konečný index.

Důsledek 5.4. *Pro všechna $k \geq 1$ platí, že pro každý jazyk L definovaný CF#RS indexu k, H existuje CF#RS indexu k bez vymazávajících pravidel (viz definice 4.3), H', takový, že $L = L(H) = L(H')$.*

Důkaz důsledku 5.4. Důkaz plyne z věty 3.1.2i v [12], která mimo jiné říká, že $\mathcal{L}_k(\mathbf{P}) = \mathcal{L}_k(\mathbf{P}, CF - \varepsilon)$ a z ekvivalence dokázané ve větě 5.3.

Alternativně by bylo možno vést důkaz podobným principem jako v autorově příspěvku [29] o odstraňování některých vymazávajících pravidel z programovaných gramatik s kontrolou výskytu. Pouze místo návěští pravidla by pro uchovávání potřebných informací sloužil stav systému.

■

5.1.2 n-pravě-lineární #-přepisující systémy

Pokusíme-li se dokázat, že $\mathcal{L}(mn\text{-}\mathbf{RLIN}\#\mathbf{RS}) = \mathcal{R}^m_{[n]}$, což by slovně řečeno znamenalo, že #-přepisující systémy s $m \cdot n$ komponentami, které používají pouze pravě-lineární pravidla, mohou být odsimulovány m-paralelními n-pravě-lineárními jednoduchými maticovými gramatikami, které kombinují volně-paralelní aplikaci pravidel s řízením pomocí jednoduchých matic, zjistíme, že to z principu nelze. Je to dáno především tím, že onen paralelismus není tak restriktivní mechanismus, abychom s jeho pomocí mohli alespoň částečně odsimulovat stavové řízení v #-přepisujícím systému. V opačném směru tento převod uskutečnitelný je (viz lemma 5.5). Zapomeňme tedy na paralelní aplikace pravidel a zaměřme se na 1-paralelní n-pravě-lineární jednoduché maticové gramatiky (viz definice 2.22 a lemma 5.6). Pro tento typ gramatik pak lze pro každé $n \geq 1$ dokázat ekvivalenci

$$\mathcal{R}^1_{[n]} = \mathcal{L}(n\text{-}\mathbf{RLIN}\#\mathbf{RS}).$$

U obou následujících lemmat budou uvedeny pouze konstrukční důkazy (viz konvence 5.1).

Lemma 5.5. *Pro všechna* $m, n \geq 1$, $\mathcal{R}^m_{[n]} \subseteq \mathcal{L}(mn\text{-}\mathbf{RLIN}\#\mathbf{RS})$.

Konstrukční důkaz lemmatu 5.5. Nechť $G = (N_{11}, \ldots, N_{mn}, T, S, P)$ je m-paralelní n-pravě-lineární jednoduchá maticová gramatika a nechť $M_1, \ldots, M_m$ jsou vzájemně disjunktní *množiny maticových pravidel* (*matrix-rule sets*), kde pro každé $1 \leq i \leq m$, $M_i = \{\mu\colon [X_{i1} \to \alpha_{i1}Y_{i1}, \ldots, X_{in} \to \alpha_{in}Y_{in}] \mid \mu \in P,\ X_{ij}, Y_{ij} \in N_{ij},\ \alpha_{ij} \in T^*,\ 1 \leq j \leq n\} \cup \{\mu\colon [X_{i1} \to \alpha_{i1}, \ldots, X_{in} \to \alpha_{in}] \mid \mu \in P,\ X_{ij} \in N_{ij},\ \alpha_{ij} \in T^*,\ 1 \leq j \leq n\}$ takové, že $P - \{\sigma\colon [S \to X_{11} \ldots X_{mn}] \mid \sigma \in P\} = \bigcup_{1 \leq i \leq m} M_i$.
Z G vytvoříme ekvivalentní mn-pravě-lineární #-přepisující systém $H = (Q, \Sigma, s, R)$, $\Sigma = T \cup \{\#\}$, $T \cap \{\#\} = \emptyset$, pomocí následujících kroků:

1. $Q = \{s\} \cup \{\langle \eta, \mu, l\rangle \mid \eta \in \text{suffixes}(X_{11} \ldots X_{mn}),\ X_{ij} \in N_{ij}$ pro všechna $1 \leq i \leq m$, $1 \leq j \leq n$, $\mu \in M_k$, $1 \leq k \leq m$, $1 \leq l \leq n\}$, kde s je nový symbol pro počáteční stav;

2. $R =$

   (i) $\{s\,_1\# \to \langle X_{11} \ldots X_{mn}, \mu_1, 1\rangle\ \#$
   $\quad \mid \mu_1 \in M_1,\ X_{11} \ldots X_{mn} = \text{rhs}(\sigma),\ \sigma\colon [S \to X_{11} \ldots X_{mn}] \in P\}$
   $\cup$

   (ii) $\{\langle Y_{11} \ldots Y_{ij-1}X_{ij} \ldots X_{mn}, \mu_i, j\rangle\,_{(i-1)\cdot n+j}\# \to$
   $\quad \langle Y_{11} \ldots Y_{ij}X_{ij+1} \ldots X_{mn}, \mu_i, j+1\rangle\alpha_{ij}\#$
   $\quad \mid \mu_i\colon [X_{i1} \to \alpha_{i1}Y_{i1}, \ldots, X_{in} \to \alpha_{in}Y_{in}] \in M_i,$
   $\quad 1 \leq i \leq m,\ 1 \leq j < n\}$
   $\cup$

   (iii) $\{\langle Y_{11} \ldots Y_{in-1}X_{in} \ldots X_{mn}, \mu_i, n\rangle\,_{i\cdot n}\# \to$
   $\quad \langle Y_{11} \ldots Y_{in}X_{(i+1)1} \ldots X_{mn}, \mu_{i+1}, 1\rangle\alpha_{in}\#$
   $\quad \mid 1 \leq i < m,\ \mu_{i+1} \in M_{i+1},\ \mu_i\colon [X_{i1} \to \alpha_{i1}Y_{i1}, \ldots, X_{in} \to$
   $\quad \alpha_{in}Y_{in}] \in M_i\}$
   $\cup$

   (iv) $\{\langle Y_{11} \ldots Y_{mn-1}X_{mn}, \mu_m, n\rangle\,_{m\cdot n}\# \to \langle Y_{11}\ \ldots\ Y_{mn}, \mu_1, 1\rangle\ \alpha_{mn}\#$
   $\quad \mid \mu_1 \in M_1,\ \mu_m\colon [X_{m1} \to \alpha_{m1}Y_{m1},\ \ldots,\ X_{mn} \to \alpha_{mn}Y_{mn}] \in M_m\}$
   $\cup$

(v) $\{\langle X_{ij} \dots X_{mn}, \mu_i, j\rangle \, {}_1\# \to \langle X_{ij+1} \dots X_{mn}, \mu_i, j+1\rangle \; \alpha_{ij}$
$\mid \mu_i \colon [X_{i1} \to \alpha_{i1}, \dots, X_{in} \to \alpha_{in}] \in M_i,\ 1 \le i \le m,\ 1 \le j < n\}$

$\cup$

(vi) $\{\langle X_{in} \dots X_{mn}, \mu_i, n\rangle \, {}_1\# \to \langle X_{(i+1)1} \dots X_{mn}, \mu_{i+1}, 1\rangle \; \alpha_{in}$
$\mid 1 \le i < m,\ \mu_{i+1} \in M_{i+1},\ \mu_i \colon [X_{i1} \to \alpha_{i1}, \dots, X_{in} \to \alpha_{in}] \in M_i\}$

$\cup$

(vii) $\{\langle X_{mn}, \mu_m, n\rangle \, {}_1\# \to \langle \varepsilon, \mu_m, n\rangle \; \alpha_{mn}$
$\mid \mu_m \colon [X_{m1} \to \alpha_{m1}, \dots, X_{mn} \to \alpha_{mn}] \in M_m\}$.

Hlavní myšlenka důkazu lemmatu 5.5. H simuluje každý derivační krok v G následovně. Informace potřebné k simulaci derivace jsou uloženy v aktuálním stavu systému H tvaru $\langle \eta, \mu_i, l\rangle$. Kvůli simulaci paralelismu v G jsou pravidla z G rozdělena na množiny maticových pravidel, M_i. Každý stav z Q obsahuje řetězec neterminálů η, návěští matice μ_i a pořadové číslo l pravidla v rámci matice. Pořadové číslo vybírá pravidlo z dané matice, které bude aplikováno v následujícím kroku. Simulace posledního pravidla v M_i změní stav systému H tak, že může být provedena simulace matice z množiny M_{i+1}. Nakonec simulace posledního pravidla matice z M_m změní stav H tak, že může být simulováno první pravidlo z matice z první množiny matic M_1.

Stručně řečeno, H díky stavovému řízení zajišťuje atomičnost sekvenčně simulovaných paralelních i maticových přepisů.

Pro změnu neterminálů v řetězci η z první komponenty stavu se používají pravidla z P tvaru $X_{ij} \to \alpha_{ij} Y_{ij}$. Pravidla tvaru $X_{ij} \to \alpha_{ij}$ jsou zase používány pro odstranění neterminálů z η. Pokud již nejsou v η žádné neterminály, systém nemá možnost, jak dále pokračovat, a jeho výpočet skončí. Pokud byla simulace úspěšná, bude vygenerován terminální řetězec.

□

Lemma 5.6. *Pro všechna $n \ge 1$, $\mathcal{L}(n\text{-}\mathbf{RLIN\#RS}) \subseteq \mathcal{R}^1_{[n]}$.*

Konstrukční důkaz lemmatu 5.6. Nechť $n \ge 1$ je kladné celé číslo a $H = (Q, \Sigma, s, R)$ je n-pravě-lineární #-přepisující systém. K systému H zkonstruujme ekvivalentní 1-paralelní n-pravě-lineární jednoduchou maticovou gramatiku $G = (N_{11}, \dots, N_{1n}, T, S, P)$ podle těchto kroků:

1. $T = \Sigma - \{\#\}$.
2. $N_{1i} = \{\langle i, j, q\rangle \mid q \in Q,\ 1 \le j \le i\} \cup \{X_i\}$ pro každé $1 \le i \le n$, kde X_i je nový neterminální symbol.
3. Přidejme $S \to \langle 1, 1, s\rangle\langle 2, 2, s\rangle \dots \langle n, n, s\rangle$ do P.
4. Pro každé pravidlo $r \colon p \to q \; \alpha\# \in R$, $\alpha \in T^*$, přidejme $[\eta_1, \dots, \eta_{i-1}, \langle i, j, p\rangle \to \alpha\langle i, j, q\rangle, \eta_{i+1}, \dots, \eta_n]$ do P, kde pro každé $k \in \{1, \dots, n\} - \{i\}$ a $1 \le k' \le k$, je η_k ve tvaru $\langle k, k', p\rangle \to \langle k, k', q\rangle$ nebo $X_k \to X_k$.
5. Pro každé pravidlo $r \colon p \to q \; \alpha \in R$, $\alpha \in T^*$, přidejme $[\eta_1, \dots, \eta_{i-1}, \langle i, j, p\rangle \to \alpha X_i, \eta_{i+1}, \dots, \eta_n]$ do P, kde
pro každé $1 \le k < i$ a $1 \le k' \le k$, je η_k tvaru $\langle k, k', p\rangle \to \langle k, k', q\rangle$ nebo $X_k \to X_k$ a
pro každé $i < l \le n$ a $1 \le l' \le n$, je η_l tvaru $\langle l, l', p\rangle \to \langle l, l' - 1, q\rangle$ nebo $X_l \to X_l$.
6. Přidejme $[X_1 \to \varepsilon, X_2 \to \varepsilon, \dots, X_n \to \varepsilon]$ do P.

Hlavní myšlenka důkazu lemmatu 5.6. G simuluje každý výpočetní krok z H následujícím způsobem. Každý neterminál obsahuje tři komponenty. Kvůli zajištění disjunktnosti abeced neterminálů $N_{1i}, \ldots, N_{1n}$ bude první komponenta každého neterminálu obsahovat index (nebo případně libovolný jiný unikátní identifikátor) dané neterminální abecedy, do které onen neterminál patří. Druhá komponenta reprezentuje pozici odpovídající hranice (#) v aktuální konfiguraci systému H. A poslední, třetí komponenta obsahuje informace o stavu systému H.

Navíc využijeme pomocné neterminály $X_1, \ldots, X_n$, které neobsahují poslední dvě komponenty, tedy žádné informace o stavech, ani o hranicích. Pomocné neterminály $X_1, \ldots, X_n$ nám dovolují mít všechny matice stejné velikosti n přesně, jak to vyžaduje definice m-Pn-G (viz definice 2.22).

Pravidla z R, která jsou tvaru $p\ {}_j\# \to q\ \alpha\#$, mění informaci o stavu uvnitř všech neterminálů až na ty, které jsou tvaru X_i. Pravidla z R tvaru $p\ {}_j\# \to q\ \alpha$ provádí totéž až na přepis neterminálu $\langle i, j, p\rangle$ na X_i a přeindexování za ním následujících neterminálů. Tímto odsimulujeme odstranění #.

Když se dopracujeme do situace, že všechny neterminály jsou tvaru X_i, použijeme pravidlo $[X_1 \to \varepsilon, X_2 \to \varepsilon, \ldots, X_n \to \varepsilon]$ k odstranění všech neterminálů z větné formy, čímž získáme kýžený terminální řetězec.

□

Věta 5.7. *Pro všechna $m, n \geq 1$, $\mathcal{R}^m_{[n]} \subset \mathcal{L}(mn\text{-}\mathbf{RLIN}\#\mathbf{RS}) = \mathcal{R}^1_{[mn]}$.*

Důkaz věty 5.7. Zapakujme, že $\mathcal{R}^{mn}_{[1]} \subset \mathcal{R}^m_{[n]} \subset \mathcal{R}^1_{[mn]}$, pro libovolné $m + n > 1$ (viz věta 10 v [77]). Takže pak věta 5.7 plyne z $\mathcal{R}^m_{[n]} \subset \mathcal{R}^1_{[mn]}$, lemmatu 5.5 a lemmatu 5.6.

■

Před dalším zajímavým výsledkem ohledně #-přepisujících systémů s n pravě-lineárními komponentami je potřeba popsat jisté omezení přechodů mezi konfiguracemi. Samotná definice lehce nezvyklého, ale poměrně obecného omezení je uvedena v samotném znění věty 5.8.

Nyní se pokusme podrobněji popsat implikaci použitou ve větě 5.8.

Výpočetní krok označený ${}_ud_i$ (viz definice 4.5) budeme nyní pro lepší názornost značit podrobněji jako $\overset{{}_ud_i}{\Rightarrow}$.

Následná implikace omezuje každý úspěšný výpočet v #-přepisujícím systému.

Nechť $d\colon s\#^n = p_0w_0 \overset{{}_{u_1}d_1}{\Rightarrow} p_1w_1 \overset{{}_{u_2}d_2}{\Rightarrow} \cdots \overset{{}_{u_i}d_i}{\Rightarrow} p_iw_i \overset{{}_{u_{i+1}}d_{i+1}}{\Rightarrow} \cdots \overset{{}_{u_j}d_j}{\Rightarrow} p_jw_j \overset{{}_{u_{j+1}}d_{j+1}}{\Rightarrow} \cdots \overset{{}_{u_{|d|}}d_{|d|}}{\Rightarrow} p_{|d|}w_{|d|}$ je úspěšný výpočet, kde $1 \leq i \leq j \leq |d|$, $u = u_i$, $v = u_j$ a $w_d \in (\Sigma - \{\#\})^*$.

Jestliže $u = v$ v ${}_ud_i$ a zároveň v ${}_vd_j$, pak jsou povoleny pouze následující dvě možnosti:

(a) všechny výpočetní kroky mezi ${}_ud_i$ a ${}_vd_j$, označené ${}_zd_k$ pro všechna $i \leq k \leq j$, přepisují právě z-tou hranici a žádnou jinou, takže $z = u = v$;

(b) je povolena pouze jediná výjimka z (a) taková, že ${}_ld_h$, $i < h < j$, je #-vymazávající výpočetní krok (viz definice 4.5).

Věta 5.8. *Nechť každý úspěšný výpočet d v n-pravě-lineárním #-přepisujícím systému H, $n \geq 1$, splňuje tuto implikaci: jestliže $1 \leq i \leq j \leq |d|$ a $u = v$ v kroku ${}_ud_i$ a ${}_vd_j$, pak buď $z = u$ ve všech ${}_zd_k$ pro všechna $i \leq k \leq j$, nebo je d_h #-vymazávající pro některé $h \in \{i + 1, \ldots, j - 1\}$. Potom, $L(H) \in \mathbf{REG}$.*

Důkaz věty 5.8. Transformujme n-pravě-lineární #-přepisující systém $H = (Q, \Sigma, s, R)$ splňující předchozí implikaci, pro libovolné $n \geq 1$, na ekvivalentní pravě-lineární gramatiku $G = (\Sigma, T, P, S)$ pomocí následujících kroků:

Pro každé $1 \leq i \leq n$, $p, q \in Q$, vytvořme pomocné množiny
${}^{p}_{q}R_i = \{r \mid r \in \text{alph}(\rho),\ \rho \in R^*,\ p\gamma\ {}_i\Rightarrow^* q\delta\ [\rho],\ \text{occur}(\#, \gamma) = \text{occur}(\#, \delta)\}$ a
${}^{p}_{q}\bar{R}_i = \{p\ {}_i\# \to q\ \alpha \in R \mid \alpha \in (\Sigma - \{\#\})^*\}$. Pak, $\mathcal{Z} = \bigcup_{i \geq 1, p, q \in Q} \{{}^{p}_{q}R_i,\ {}^{p}_{q}\bar{R}_i\}$.

1. $T = \Sigma - \{\#\}$,
2. $\Sigma = N \cup T \cup \{S\}$, kde S je nový symbol a N obsahuje všechny neterminály zavedené v následující konstrukci množiny P,
3. $P = \bigcup_{1 \leq l \leq 5} P_l$, kde množiny P_1 až P_5 jsou vytvořeny pomocí následujícího postupu:

   (i) inicializace: $P_1 = \{S \to \langle \#^n, i, s\rangle \mid 1 \leq i \leq n\}$;

   (ii) přípravná fáze: $P_2 = \{\langle \nabla_1\eta_1\nabla\eta_2 \ldots \nabla_i\eta_i\nabla_{i+1}\eta_{i+1} \ldots \nabla_n\eta_n, i, p\rangle \to$
   $\langle \nabla_1\eta_1\nabla\eta_2 \ldots \nabla_i\eta_i\ {}^{p}_{q}R_{i'}\nabla_{i+1}\eta_{i+1} \ldots \nabla_n\eta_n, j, q\rangle$
   $\mid {}^{p}_{q}R_{i'} \neq \emptyset,\ 1 \leq j \leq n,\ \eta_t \in \mathcal{Z}^*,\ \nabla_t \in \{\#, \bar{\#}\}$ pro $1 \leq t \leq n,\ \nabla_i \neq \bar{\#},$
   $i' = \text{occur}(\#, \nabla_1\eta_1 \ldots \nabla_i)\}$
   $\cup$
   $\{\langle \nabla_1\eta_1 \ldots \nabla_{i-1}\eta_{i-1}\nabla_i\eta_i\nabla_{i+1} \ldots \nabla_n\eta_n, i, p\rangle \to$
   $\langle \nabla_1\eta_1 \ldots \nabla_{i-1}\eta_{i-1}\bar{\#}\eta_i\ {}^{p}_{q}\bar{R}_{i'}\nabla_{i+1}\eta_{i+1} \ldots \nabla_n\eta_n, j, q\rangle$
   $\mid {}^{p}_{q}\bar{R}_{i'} \neq \emptyset,\ 1 \leq j \leq n,\ \eta_t \in \mathcal{Z}^*,\ \nabla_t \in \{\#, \bar{\#}\}$ pro $1 \leq t \leq n,\ \nabla_i \neq \bar{\#},$
   $i' = \text{occur}(\#, \nabla_1\eta_1 \ldots \nabla_i)\}$;

   (iii) zarážka:
   $P_3 = \{\langle \gamma, i, p\rangle \to \langle \gamma, q\rangle \mid p, q \in Q,\ \# \notin \text{alph}(\gamma),\ A \to \langle \gamma, i, p\rangle \in P_2\}$;

   (iv) simulace derivačního kroku z G ($\eta_t \in \mathcal{Z}^*$ pro všechna $1 \leq t \leq n$):
   $P_4 = \{\langle \bar{\#}\ {}^{p}_{q}R_{i'}\eta_i \ldots \bar{\#}\eta_n, p'\rangle \to \alpha\langle \bar{\#}\ {}^{p}_{q}R_{i'}\eta_i \ldots \bar{\#}\eta_n, q'\rangle$
   $\mid p'\ {}_{i'}\# \to q'\ \alpha\# \in {}^{p}_{q}R_{i'},\ \alpha \in (\Sigma - \{\#\})^*,\ 1 \leq i \leq n,\ {}^{p}_{q}R_{i'} \in \mathcal{Z}\}$
   $\cup$
   $\{\langle \bar{\#}\ {}^{p}_{q}R_{i'}\eta_i \ldots \bar{\#}\eta_n, p'\rangle \to \alpha\langle \bar{\#}\eta_i \ldots \bar{\#}\eta_n, q\rangle$
   $\mid p'\ {}_{i'}\# \to q\ \alpha\# \in {}^{p}_{q}R_{i'},\ \alpha \in (\Sigma - \{\#\})^*,\ 1 \leq i \leq n,\ {}^{p}_{q}R_{i'} \in \mathcal{Z}\}$
   $\cup$
   $\{\langle \bar{\#}\ {}^{p}_{q}\bar{R}_{i'}\bar{\#}\eta_{i+1} \ldots \bar{\#}\eta_n, p\rangle \to \alpha\langle \bar{\#}\eta_{i+1}\bar{\#}\eta_n, q'\rangle$
   $\mid p\ {}_{i'}\# \to q\ \alpha \in {}^{p}_{q}\bar{R}_{i'},\ \alpha \in (\Sigma - \{\#\})^*,\ 1 \leq i \leq n,\ {}^{p}_{q}\bar{R}_{i'} \in \mathcal{Z}\}$;

   (v) finalizace:
   $P_5 = \{\langle \varepsilon, p\rangle \to \varepsilon \mid p \in Q\}$.

Převod pravě-lineární gramatiky, G, na n-pravě-lineární #-přepisující systém, H je jednoduchý a necháme tuto transformaci na čtenáři.

Hlavní myšlenka důkazu věty 5.8. Každé ${}^{p}_{q}R_i$ reprezentuje množinu pravidel, kterými lze provést výpočet stupně i vedoucí ze stavu p do stavu q v H. V každé větné formě v G je pouze jediný výskyt neterminálu a tento neterminál je složen ze tří komponent:

(1) γ—konečný řetězec, který předepisuje a řídí simulaci, $\gamma \in (\#\mathcal{Z}^*)^+$;
(2) i—pozice výskytu hranice (#), která je aktivní v aktuální konfiguraci systému H;

(3) p—právě simulovaný stav systému H.

Pro simulaci jsou během přípravné fáze nedeterministicky vygenerovány předepisující podřetězce za každou odpovídající hranicí v konfiguraci systému H. Tyto podřetězce η_t jsou tvaru $\mathcal{Z}^*$.

Ve třetím kroku jsou odstraněny druhé komponenty neterminálů z G, abychom zajistili, že žádné další pravidlo z P_2 nemůže být dále použito.

Dále jsou vygenerovány terminální symboly podle pravidel zkonstruovaných krokem (3iv) přesně podle předepisujícího řetězce γ. Každá zkompletovaná množina ${}^{p}_{q}R_i$ je postupně odstraňována z γ, dokud není γ prázdné. Nakonec je jediný zbývající neterminál ve větné formě gramatiky G přepsán na prázdný řetězec pravidlem z P_5, čímž je získán řetezec terminálů.

■

Věta 5.9. $\mathbf{REG} \subseteq \mathcal{L}(1\text{-}\mathbf{RLIN\#RS})$.

Hlavní myšlenka důkazu věty 5.9. Důkaz o převoditelnosti každé regulární gramatiky na 1-pravě-lineární #-přepisující systém je poměrně jednoduchý. Jediná hranice bude simulovat jediný neterminál používaný v regulární gramatice. Údaj o tom, který konkrétní neterminál koresponduje k oné hranici, bude uschován ve stavu v první komponentě konfigurace 1-pravě-lineárního #-přepisujícího systému (podobně jako v důkazu lemmatu 5.1).

■

Důsledek 5.10. $\mathcal{L}(1\text{-}\mathbf{RLIN\#RS}) = \mathbf{REG}$.

Důkaz důsledku 5.10. Protože 1-RLIN#RS obsahuje maximálně jednu hranici (a stejně tak jeho pravidla) a platí věta 5.9, plyne důsledek z obecnější věty 5.8 pro $n = 1$.

■

Na konec této podsekce připomeňme, že na rozdíl od předchozích variant #-přepisujících systémů, u této modifikace není třeba zvlášť uvažovat verzi s a bez konečného indexu, protože pravidla jsou takového tvaru, že nikdy není zvýšen počet hranic nad počet obsažený v počáteční konfiguraci. n-pravě-lineární #-přepisující systém je pak touto vlastností implicitně omezen na konečný index.

5.1.3 Zobecněné #-přepisující systémy

Zjistíme, že zobecnění pravidel nám v případě tohoto omezení nedopomůže k lepším vyjadřovacím schopnostem modelu, jak je tomu například v případě gramatik Chomského hierarchie (bezkontextové vs kontextové gramatiky).

V důkazech tohoto výsledku jsou uvedeny pouze konstrukční části a nastínění rigorózního důkazu v souladu s konvencí 5.1.

Věta 5.11. *Pro všechna* $k \geq 1$, $\mathcal{L}_k(\mathbf{CF\#RS}) = \mathcal{L}_k(\mathbf{G\#RS})$.

Protože bezkontextový #-přepisující systém indexu k je (plyne z definice) jenom speciální případ zobecněného systému, stačí nám dokázat, že $\mathcal{L}_k(\mathbf{G\#RS}) \subseteq \mathcal{L}_k(\mathbf{CF\#RS})$.

Důkaz věty 5.11. Nechť $k \geq 1$ je celé kladné číslo. Nechť $H = (Q, T \cup \{\#\}, s, R)$ je zobecněný #-přepisující systém indexu k, kde $\Sigma = T \cup \{\#\}$, $\# \notin T$. Nechť $\mu = \max(\{\max_L(H), \max_R(H)\})$ a nechť \$ je nový symbol, $\$ \notin Q \cup \Sigma$. Zkonstruujme ekvivalentní bezkontextový #-přepisující systém indexu k, $H' = (Q', T \cup \{\#\}, s', R')$, kde komponenty Q', s' a R' jsou vytvořeny následujícím způsobem:

1. $s' = \langle s, \# \rangle$;

2. $Q' = \{\langle p, y_0\#y_1\#\dots y_{h-1}\#y_h\rangle$
$\quad \mid p \in Q,\ 1 \le h \le k,$
$\quad\ y_j = y'_j \nabla_j y''_j,\ y'_j, y''_j \in T^*,\ \nabla_j \in \{\varepsilon, \$\},$
$\quad\ |y'_j| \le 2\mu,\ |y''_j| \le 2\mu,\ 0 \le j \le h\}$;

3. $R' = \{\langle p, x_0\#\dots\#x'_j\nabla_j x''_j\#\dots\#x_h\rangle\ {}_1\# \to \langle p, x_0\#\dots\#y'_j\$y''_j\#\dots\#x_h\rangle\ \#$
$\quad \mid \langle p, x_0\#\dots\#x'_j\nabla_j x''_j\#\dots\#x_h\rangle, \langle p, x_0\#\dots\#y'_j\$y''_j\#\dots\#x_h\rangle \in Q',$
$\quad\ \nabla_j \in \{\varepsilon, \$\},\ y'_j \in \text{prefixes}(x'_j),\ y''_j \in \text{suffixes}(x''_j),$
$\quad\ x_j \in T^*,\ 1 \le h \le k,\ 0 \le j \le h,$ kde $x'_0 = y'_0 = x''_h = y''_h = \varepsilon\}$;

4. Pro každé pravidlo $r\colon p\ {}_i\alpha\#\beta \to q\ \alpha\gamma\beta \in R$, přidejme následující množinu do R':
$\{\langle p, \eta'\alpha\#\beta\eta''\rangle\ {}_i\# \to \langle q, \eta'\alpha\gamma\beta\eta''\rangle\ \gamma$
$\quad \mid \text{occur}(\#, \eta'\alpha) = i - 1,$
$\quad\ \langle p, \eta'\alpha\#\beta\eta''\rangle, \langle q, \eta'\alpha\gamma\beta\eta''\rangle \in Q'\}$.

Hlavní myšlenka důkazu věty 5.11. H' pomocí několika výpočetních kroků simuluje jediný krok v H. Každý stav v konfiguraci systému H' je tvaru $\langle p, \eta\rangle$ a obsahuje dvě komponenty:

(1) p—aktuální stav v simulovaném H;

(2) η—kontextový řetězec aktuální konfigurace z H, který reprezentuje kontext každé hranice délky maximálně μ na obě strany hranice (tj. levý i pravý kontext).

Samotná simulace je prováděna ve dvou krocích podle pravidel, která byla vytvořena během třetího a čtvrtého konstrukčního kroku předchozí konstrukce. Detailněji probíhá následovně:

(a) Každý podřetězec terminálů mezi dvěma hranicemi může být nedeterministicky zkrácen pravidly z třetího konstrukčního kroku. Pozice, kde byl kontextový řetězec zkrácen, je označena speciálním symbolem \$, abychom byly schopni tuto skutečnost rozpoznat v následné kontextové kontrole. Tento krok nedeterministického zkracování je nutný proto, abychom v konečném kontextovém řetězci vytvořili dostatek místa pro aplikaci pravidla a poznačení si kontextu tohoto pravidla pro další simulaci.

(b) Každým pravidlem zobecněného systému H je přepsána vždy jen jediná hranice, s ohledem na její bezprostřední levý a pravý kontext. Pravidla bezkontextového tvaru konstruovaná ve čtvrtém kroku jsou aplikována, aby v H' simulovaly kontextové přepisování ze systému H. Toho docílí prostřednictvím kontextové kontroly, která však nemůže být zajištěna přepisovacím jádrem pravidla, neboť se jedná o bezkontextové jádro. Místo toho je využita možnost tuto kontrolu přenést na stavové řízení a kontrolu relevantního podřetězce ve druhé komponentě každého stavu tvaru $\langle p, \eta\rangle$ v H'.

Simulace je dokončena v momentě, kdy získáme terminální řetězec.

Nástin rigorózního důkazu:

Nechť π je homomorfismus z konfigurace bezkontextového #-přepisujícího systému H' na odpovídající konfiguraci zobecněného #-přepisujícího systému H definovaný jako $h(\langle p, \eta\rangle z) = pz$.

Tvrzení 5.11.1. *Pokud* $s\# \Rightarrow^m pz_0\#z_1\#\dots\#z_h$ *v* H, *pak* $\langle s, \#\rangle\# \Rightarrow^r \langle p, x_0\#x_1\#\dots\ \#x_h\rangle$ $z_0\#z_1\dots\#z_h\ [r'_1r'_2\dots r'_r]$ *v* H', *kde* $x_i = x'_i\nabla_i x''_i$, $\nabla_i \in \{\varepsilon, \$\}$, $x'_i \in \text{prefixes}(z_i)$, $x''_i \in \text{suffixes}(z_i)$, $|x_i| \le 4\mu + 1$, $x_i, z_i \in T^*$, $0 \le i \le h$, $1 \le h \le k$, *pro* $m \ge 0$.

Důkaz tvrzení 5.11.1. Toto tvrzení dokažme indukcí podle m.

Indukční základ: Nechť $m = 0$. Pro $s\# \Rightarrow^0 s\#$ v H existuje $s'\# \Rightarrow^0 s'\#$ v H', kde $s' = \langle s, \# \rangle$.

Indukční hypotéza: Předpokládejme, že tvrzení 5.11.1 platí pro všechny výpočty délky m nebo kratší pro nějaké $m \geq 0$.

Indukční krok: Uvažujme $s\# \Rightarrow^{m+1} qz'$, kde $z' \in \Sigma^*$. Vyjádřeme $s\# \Rightarrow^{m+1} qz'$ jako $s\# \Rightarrow^m pz\ [r_1 r_2 \dots r_m]$, kde $z = z_0\#z_1\# \dots \alpha\#\beta \dots \#z_h$, $\#$ mezi α a β je i-tá hranice, $r_1, \dots, r_m, r_{m+1} \in \mathrm{Lab}(R)$ a $pz \Rightarrow qz'\ [r_{m+1}]$. Pro pravidlo $r_{m+1}\colon p\,{}_i\alpha\#\beta \to q\ \alpha\gamma\beta$ je z' tvaru: $z' = z_0\#z_1\# \dots \alpha\gamma\beta \dots \#z_h$, kde $z_0, \dots, z_h \in T^*$.

Na základě indukční hypotézy existuje $\langle s, \# \rangle\# \Rightarrow^r \langle p, x_0\#x_1\# \dots \alpha\#\beta \dots \#x_h \rangle\ z_0\# \dots \alpha\#\beta \dots z_{h-1}\#z_h\ [r'_1 r'_2 \dots r'_r] \Rightarrow^* \langle p, y_0\#y_1\# \dots \alpha\#\beta \dots \#y_h \rangle\ z_0\# \dots \alpha\#\beta \dots z_{h-1}\#z_h\ [\rho] \Rightarrow \langle q, y_0\#y_1\# \dots \alpha\gamma\beta \dots \#y_h \rangle\ z_0\# \dots \alpha\gamma\beta \dots z_{h-1}\ \#z_h\ [r'_{r+1}]$, kde ρ je posloupnost pravidel zkonstruovaná v kroku 3, $|\rho| \geq 0$, $r \geq 1$, $r'_j \in \mathrm{Lab}(R')$, $1 \leq j \leq r+1$, $x_j, y_j \in T^*\{\varepsilon, \$\}T^*$, $z_j \in T^*$, $0 \leq j \leq h$, $\mathrm{occur}(\#, y_0\#y_1\# \dots \alpha) = \mathrm{occur}(\#, z_0\#z_1\# \dots \alpha) = i-1$, $\alpha = \bar{x}_{i-\bar{\alpha}}\#x_{i-\bar{\alpha}+1} \dots x_{i-2}\#x_{i-1}$, $\bar{\alpha} = \mathrm{occur}(\#, \alpha)+1$, $\bar{x}_{i-\bar{\alpha}} \in \mathrm{suffixes}(x_{i-\bar{\alpha}})$, $\beta = x_i\#x_{i+1} \dots x_{i+\bar{\beta}-1}\ \#\bar{x}_{i+\bar{\beta}}$, $\bar{\beta} = \mathrm{occur}(\#, \beta)$, $\bar{x}_{i+\bar{\beta}} \in \mathrm{prefixes}(x_{i+\bar{\beta}})$, a pravidlo $r'_{r+1}\colon \langle p, y_0\#y_1\# \dots \alpha\#\beta \dots\ \#y_h \rangle\, \#\, \# \to \langle q, y_0\#y_1\# \dots \alpha\gamma\beta \dots \#x_h \rangle\ \gamma \in R'$ představené v kroku 4.

Tedy, $\pi(\langle p, y_0\#y_1\# \dots \alpha\#\beta \dots \#y_h \rangle\ z_0\# \dots \alpha\#\beta \dots z_{h-1}\#z_h) = pz$ a $\pi(\langle q, y_0\#y_1\# \dots \alpha\gamma\beta \dots \#y_h \rangle\ z_0\# \dots \alpha\gamma\beta \dots z_{h-1}\#z_h) = qz'$, takže tvrzení platí.

Tvrzení 5.11.2. *Jestliže $s\# \Rightarrow^z pw$ v H, tak $\langle s, \# \rangle\# \Rightarrow^* \langle p, \eta \rangle w$ v H' pro některé $z \geq 0$, $w \in T^*$.*

Důkaz tvrzení 5.11.2. Uvažujme tvrzení 5.11.1 pro $h = 0$. V tom případě pokud $s\# \Rightarrow^z pz_0$ v H, pak $\langle s, \# \rangle\# \Rightarrow^* \langle p, x_0 \rangle z_0$ v H', kde $z_0 = w$.

Tvrzení 5.11.1 a 5.11.2 nastiňují formální důkaz věty 5.11 a její konstrukční části.

■

Důsledek 5.12. *Pro všechna $k \geq 1$ platí, že pro každý jazyk L definovaný G#RS indexu k, H, existuje G#RS indexu k bez vymazávajících pravidel (viz definice 4.3), H', takový, že $L = L(H) = L(H')$.*

Důkaz důsledku 5.12. Triviálně plyne z věty 5.11 a důsledku 5.4. ■

Neomezené #-přepisující systémy

Prozatím jsme diskutovali zobecněné jazyk-definující zařízení, které reprezentuje kombinaci konečného automatu, kontextové gramatiky a několika dalších podmínek při aplikaci pravidel. Nyní se podívejme na možnost ještě více zobecnit aplikovaná pravidla po vzoru neomezených gramatik (viz definice 2.11), které při přepisování v podstatě stírají rozdíl mezi terminálními a neterminálními symboly. Pravý a levý kontext na levé straně pravidel zůstane zachován s tím rozdílem, že ho nyní budeme moci při přepisování libovolně modifikovat. To znamená, že již nebudeme modifikovat pouze hranici mezi levým a pravým kontextem, ale budeme skutečně přepisovat celý podřetězec z levé strany pravidla na řetězec na pravé straně pravidla.

Definice 5.1. Mějme G#RS $H = (Q, \Sigma, s, R)$, kde jsou ale pravidla z R tvaru:

$$p\,{}_i\alpha \to q\ \beta,$$

kde $p, q \in Q$, $i \in \mathbb{N}$, $\alpha, \beta \in \Sigma^*$, $\mathrm{occur}(\#, \alpha) \geq 1$, pak H nazýváme *neomezený #-přepisující systém* (*unrestricted #-rewriting system*, zkráceně U#RS). Výpočetní krok definujeme takto:

Nechť $pu\alpha'\#\alpha''v$, $qu\beta v$ jsou dvě konfigurace, $p, q \in Q$, $u, v \in \Sigma^*$, $i \in \mathbb{N}$ a $\text{occur}(\#, u\alpha') = i - 1$. Pak neomezený #-přepisující systém provádí výpočetní krok z $pu\alpha'\#\alpha''v$ do $qu\beta v$ pomocí *neomezeného pravidla* $r\colon p\ {}_i\alpha'\#\alpha'' \to q\ \beta$.

Nyní si demonstrujme funkčnost neomezeného #-přepisujícího systému na příkladě.

Příklad 5.1. V [12] věta 3.1.7 dokazuje, že pro všechna $k \geq 2$, $L_k \in \mathcal{L}_k(\mathbf{CF\#RS})$ a $L_k \notin \mathcal{L}_{k-1}(\mathbf{CF\#RS})$, kde $L_k = \{b(a^ib)^{2k} \mid i \geq 1\}$.

Uvažujme $U\#RS$ $H = (\{s, s', p, q, r, r', f\}, \{a, b, \#\}, s, R)$, kde R obsahuje

1: $s\ {}_1\# \to s'\ \#\#a\#\#$
2: $s'\ {}_2\# \to s'\ \#a$
3: $s'\ {}_1\# \to p\ \#$
4: $p\ {}_2\#a \to q\ a\#$
5: $q\ {}_3\# \to p\ \#a$
6: $p\ {}_2\#\# \to r\ \#\#$
7: $r\ {}_1\# \to r'\ b$
8: $r'\ {}_3\# \to p\ \#\#$
9: $p\ {}_1\#\# \to f\ b$
10: $f\ {}_1\#\# \to f\ b$.

H například umí provést výpočet k větě $baabaabaab = b(a^2b)^3$ následovně:

$s\# \Rightarrow s'\#\#a\#\#$ [1] $\Rightarrow s'\#\#aa\#\#$ [2] $\Rightarrow p\#\#aa\#\#$ [3]
$\Rightarrow q\#a\#a\#\#$ [4] $\Rightarrow p\#a\#a\#a\#$[5]
$\Rightarrow q\#aa\#\#a\#$ [4] $\Rightarrow p\#aa\#\#aa\#$ [5]
$\Rightarrow r\#aa\#\#aa\#$ [6] $\Rightarrow r'baa\#\#aa\#$ [7] $\Rightarrow pbaa\#\#aa\#\#$ [8]
$\Rightarrow qbaa\#a\#a\#\#$ [4] $\Rightarrow pbaa\#a\#a\#a\#$ [5]
$\Rightarrow qbaa\#aa\#\#a\#$ [4] $\Rightarrow pbaa\#aa\#\#aa\#$ [5]
$\Rightarrow rbaa\#aa\#\#aa\#$ [6] $\Rightarrow r'baabaa\#\#aa\#$ [7] $\Rightarrow pbaabaa\#\#aa\#\#$ [8]
$\Rightarrow fbaabaabaa\#\#$ [9] $\Rightarrow fbaabaabaab$ [10].

Je zřejmé, že podle zavedených definic pro předchozí typy #-přepisujících systémů je H indexu 4 a $L(H) = \{b(a^ib)^j \mid i, j \geq 1\} \in \mathcal{L}_4(\mathbf{U\#RS})$. Tento příklad tedy ukazuje, že pro každé $k \geq 1$, existuje jazyk $L = L_n$, $n > k$, takový, že $L \notin \mathcal{L}_k(\mathbf{CF\#RS})$ a L je generován neomezeným #-přepisujícím systémem indexu 4.

Pozorování 5.1. Všimněme si, že pak věta analogická k větě 5.21 pro neomezené systémy, na rozdíl od věty 5.21, neplatí. Dále můžeme pozorovat, že omezení na konečný index (v případě, že omezujeme pouze počet hranic, jak to bylo u předchozích typů běžné) je zde nepodstatné a spíše bychom se museli zamýšlet nad omezením pracovního prostoru (viz definice 3.3), abychom dosáhli nějakého faktického omezení vyjadřovací síly.

Hypotéza 5.1. $\mathcal{L}(\mathbf{U\#RS}) = \mathbf{RE}$.

Hypotéza se zdá býti pravdivá již pro index 1, protože neomezená pravidla nám umožňují # volně přesouvat ve větné komponentě například pomocí pravidel tvaru $p\ {}_1\#a \to p\ a\#$ a $p\ {}_1a\# \to p\ \#a$. Lze tak simulovat neomezené gramatiky (viz definice 2.11) a to pravděpodobně dokonce pouze s využitím omezeného počtu stavů. Zbývá pouze vyřešit drobný technický problém s kontrolou, zda již máme větu, která neobsahuje žádnou #.

Navíc díky pozorování 5.1 věříme, že neomezené #-přepisující systémy netvoří na základě konečného indexu nekonečnou hierarchii, protože předpokládáme, že každý U#RS indexu k bude možno převést na ekvivalentní U#RS indexu 1. Tato hypotéza je předmětem dalšího studia.

Otevřený problém 5.2. Mějme libovolné #-přepisující systémy (CF#RS, n-RLIN#RS, G#RS) bez omezování jejich konfigurací na konečný index. Jaké třídy jazyků tyto systémy definují a jaké jsou vztahy mezi nimi navzájem?

Během celé kapitoly jsme se soustředili na #-přepisující systémy konečného indexu. Je ale také vhodné si připomenout, že v případě neomezování #-přepisujícího systému (s libovolnými tvary pravidel) na konečný index, nevíme téměř nic konkrétního o vyjadřovací síle tohoto modelu. Tato pasáž čeká na budoucí detailnější výzkum.

5.1.4 Zásobníkové automaty s omezovaným obsahem zásobníku

Nejprve si ukážeme, že na rozdíl od regulárním jazykem řízených gramatik, v případě zásobníkových automatů s omezeným zásobníkem pomocí regulárního jazyka lepší vyjadřovací schopnosti nezískáme (věta 5.13). V druhé části podsekce dokážeme, že stačí lineární jazyk pro omezování konfigurace RCPDA a dokážeme přijímat i nebezkontextové jazyky (věta 5.14).

Pro každý RCPDA $H = (M, L)$ s omezujícím jazykem $L \in \mathbf{REG}$ (viz definice 4.12) lze zkonstruovat ekvivalentní zásobníkový automat M (viz definice 2.17).

Věta 5.13. $\mathcal{L}(\mathbf{RCPDA}, \mathbf{REG}) = \mathbf{CF}$.

Důkaz věty 5.13. Nechť $K = (Q_K, \Sigma_K, R_K, s_K, F_K)$ je konečný automat takový, že $L_K = L(K)$, a nechť $H = (M, L_K)$ je zásobníkový automat s omezeným zásobníkem, kde $M = (Q_M, \Gamma_M, \Sigma_M, R_M, s_M, S_M, F_M)$ je zásobníkový automat. Nyní zkonstruujeme klasický zásobníkový automat $N = (Q, \Gamma, \Sigma, R, s, S, F)$ takový, že $L(H) = L(N)$:

1. $Q = Q_M \cup \{s\}$, kde $s \notin Q_M$ je nový stav;
2. $\Gamma = \Gamma_M \cup Q_K \cup \{S\}$, kde $S \notin \Gamma_M \cup Q_K$ je nový zásobníkový symbol;
3. $\Sigma = \Sigma_M$;
4. $F = F_M$;
5. $R = \{Ss \to s_K S_M s_M\} \cup$
   $\{p_1 B_1 p_2 B_2 \dots p_m B_m p_{m+1} o_M a \to q_1 A_1 q_2 A_2 \dots q_n A_n q_{n+1} r_M \mid$
   $p_1, p_2, \dots, p_m \in Q_K,\ p_{m+1} \in F_K,\ m \geq 0,$
   $B_1, B_2, \dots, B_m \in \Gamma_M,\ A_1, A_2, \dots, A_m \in \Gamma_M \cap \Sigma_K,$
   $q_1 A_1 \to q_2,\ q_2 A_2 \to q_3,\ \dots,\ q_n A_n \to q_{n+1} \in R_K,\ q_{n+1} \in F_K,\ n \geq 0,$
   $B_1 B_2 \dots B_m o_M a \to A_1 A_2 \dots A_n r_M \in R_M,\ o_M, r_M \in Q_M,\ a \in \Sigma_M\}$.

Rigorózní důkaz demonstrující skutečnou rovnost definovaných jazyků definovaných oběma modely je ponechán na laskavém čtenáři. Je možno jej vést matematickou indukcí nad sekvencí přechodů od počáteční konfigurace ke koncové konfiguraci v obou přepisujících systémech.

Hlavní myšlenka důkazu věty 5.13. Převod klasického zásobníkového automatu na automat s omezeným obsahem zásobníku je triviální, protože z definice je každý zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$ zároveň automatem s omezeným obsahem zásobníku s omezujícím jazykem $\Xi = \Gamma^*$, který je regulární.

Důkaz druhého směru inkluze má konstrukční povahu. Kromě prvního pravidla, které zastupuje inicializační fázi, prokládají všechna ostatní pravidla zásobník pomocnými symboly, jež reprezentují stavy z Q_K konečného automatu K.

Obsah zásobníku v M je tedy tvaru $(Q_K \Gamma_M)^*$. Pamatování si všech stavů, kterými je nutné projít při přijímání obsahu zásobníku konečným automatem K zajišťujeme prokládáním zásobníkových symbolů z Γ_M symboly z Q_K, což reflektují zkonstruovaná pravidla tvaru $(p_1 B_1 p_2 B_2 \dots p_m B_m p_{m+1} o_M a \rightarrow q_1 A_1 q_2 A_2 \dots q_n A_n q_{n+1} r_M)$. Tím zajistíme kontrolu, že multipodřetězce $B_1 B_2 \dots B_m$ a $A_1 A_2 \dots A_n$ tvoří příponu věty omezujícího jazyka, která odpovídá aktuálnímu obsahu zásobníku.

Pro platnost omezování konfigurace omezujícím jazykem definovaným omezujícím automatem K vyžadujeme, aby poslední symbol na zásobníku automatu M byl vždy nějaké $q_{n+1} \in F_K$.

■

Věta 5.14. $\mathcal{L}(\mathbf{RCPDA}, \mathbf{REG}) \subset \mathcal{L}(\mathbf{RCPDA}, \mathbf{LIN})$.

Důkaz věty 5.14. V příkladu 4.7 je popsána instance zásobníkového automatu s lineárním omezujícím jazykem, který přijímá jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$. Pomocí pumping lemma pro bezkontextové jazyky (strana 512 v [48]) lze dokázat, že $L \notin \mathbf{CF}$. Protože každý omezující regulární jazyk je zároveň lineární a platí věta 5.13, dostáváme ostrou inkluzi.

■

Důsledek 5.15. $\mathbf{CF} = \mathcal{L}(\mathbf{RCPDA}, \mathbf{REG}) \subset \mathcal{L}(\mathbf{RCPDA}, \mathbf{LIN}) \subseteq \mathbf{RE}$

Důkaz důsledku 5.15. Důsledek plyne z vět 5.13 a 5.14. ■

V budoucnu by mohlo být zajímavé studovat možnosti omezování konfigurací gramatik. Pravděpodobně bychom došli na úzký vztah k operacím průnik a substituce nad jazyky.

5.1.5 Redukující hluboké zásobníkové automaty

Nyní si budeme demonstrovat mocnost a z ní plynoucí nekonečnou hierarchii tříd jazyků definovaných redukujícími hlubokými zásobníkovými automaty založenými na hloubce n, které zpracovávají vstupní větu zprava-doleva ([35, 36]). To jest, pro všechna $n \geq 1$, ${}_n\mathcal{L}(\mathbf{RDPDA}) \subset {}_{n+1}\mathcal{L}(\mathbf{RDPDA}) \subset \mathbf{CS}$. Tato nekonečná hierarchie využívá existující hierarchie stavových gramatik v závislosti na n-limitovanosti, kterou dokázal Kasai v roce 1970 ([21] nebo viz věta 2.5). Proto v následujících lemmatech a větě nejprve dokážeme ekvivalenci ${}_n\mathcal{L}(\mathbf{ST})$ a ${}_n\mathcal{L}(\mathbf{RDPDA})$ pro každé $n \geq 1$.

Lemma 5.16. *Pro všechna* $n \geq 1$, ${}_n\mathcal{L}(\mathbf{ST}) \subseteq {}_n\mathcal{L}(\mathbf{RDPDA})$.

Konstrukční důkaz lemmatu 5.16. Nechť $G = (V, W, T, P, S)$ je n-limitovaná stavová gramatika a $n \geq 1$. Položme $N = V - T$. Definujme homomorfismus f nad $(\{\#\} \cup V)^*$ jako $f(A) = A$ pro všechna $A \in \{\#\} \cup N$ a $f(a) = \varepsilon$ pro všechna $a \in T$. Představme ${}^{rl}\mathbf{RDPDA}$ hloubky n,

$$ {}_nM = (Q, T, \{\#\} \cup V, R, s, S, \{\$\}), $$

kde $Q = \{s, \$\} \cup \{\langle p, u\rangle \mid p \in W,\ u \in \mathrm{prefix}(N^*\{\#\}^n, n),\ |u| \leq n\}$ a R je zkonstruováno pomocí následujících kroků:

1. pro všechna $(p, A) \rightarrow (q, x) \in P$, $p, q \in W$, $A \in N$, $x \in T^+$,
   přidejme $sx \vdash 1\langle p, A\#^{n-1}\rangle A$ do R;

2. je-li $(p, A) \rightarrow (q, x) \in P$, $\langle q, \mathrm{prefix}(uf(x)v, n)\rangle \in Q$, $p, q \in W$, $A \in N$,
   $x \in V^+$, $u \in N^*$, $v \in N^*\{\#\}^*$, $|uAv| = n$, $p \notin {}_G states(u)$, pak
   přidejme $\langle q, \mathrm{prefix}(uf(x)v, n)\rangle x \vdash |uA|\langle p, uAv\rangle A$ do R;

3. pro všechna $(p, S) \to (q, x) \in P$, $p, q \in W$, $x \in V^+$,
   přidejme $\langle q, \text{prefix}(f(x)\#^n, n)\rangle x \vdash 1\$S$ do R;

4. je-li $A \in N$, $p \in W$, $u \in N^*$, $v \in \{\#\}^*$, $|uv| \leq n-1$, $p \notin {}_G states(u)$, pak
   přidejme $\langle p, uv\rangle A \vdash |uA|\langle p, u\#v\rangle A$
   a $\langle p, uv\rangle A \vdash |uA|\langle uAv\rangle A$ do R.

Hlavní myšlenka důkazu lemmatu 5.16. Každý n-limitovaný derivační krok v G simulujeme pomocí opačného/reverzního redukčního kroku v ${}_nM$. To znamená, že pokud je některý neterminál (i-tý zleva) přepsán řetězcem v G, tak přesně stejný řetězec je redukováno/nahrazen na zásobníku automatu ${}_nM$ přesně tím samým nevstupním symbolem v hloubce i, $1 \geq i \geq n$. Stavy automatu ${}_nM$ se skládají ze dvou komponent:

a) původního stavu gramatiky G a

b) řetězce délky n, který zaznamenává prvních n neterminálů v aktuální simulované větné formě (případně do stanovené délky doplněno symboly #)

První krok vytváří inicializační pravidla, protože na rozdíl od stavové gramatiky ^{rl}**RDPDA** specifikuje koncové stavy. Druhým krokem zkonstruujeme pravidla pro údržbu konzistence druhé komponenty stavů s obsahem zásobníku. Pravidla třetího konstrukčního kroku simulují první derivační krok v G (z počáteční konfigurace (p, S)) pomocí vyprázdnění zásobníku automatu ${}_nM$. Čtvrtý krok doplňuje obsah druhé komponenty stavů ${}_nM$ tak, aby reflektovala obsah zásobníku, respektive řetězec n nejvrchnějších symbolů z N.

Když G úspěšně dokončí generování řetězce terminálů, ${}_nM$ skončí v koncovém stavu \$, s přečteným vstupem a s prázdným zásobníkem.

Rigorózní důkaz: Dále následuje formální důkaz, že pro každou n-limitovanou stavovou gramatiku, G, a všechna $n \geq 1$, vytvoří předchozí konstrukce ekvivalentní zprava-doleva hluboký redukující zásobníkový automat hloubky n, ${}_nM$, takový, že $L(G, n) = L({}_nM)$.

Tvrzení 5.16.1. *Nechť* $(p, S)_n \Rightarrow^m (q, dy)$ *v* G, *kde* $d \in T^*$, $y \in (NT^*)^*$, $p, g \in W$, $m \geq 0$. *Pak* $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, d, y\#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ *v* ${}_nM$.

Důkaz tvrzení 5.16.1 (indukcí podle $m = 0, 1, \ldots$)**.**
Indukční základ: Nechť $i = 0$, takže $(p, S)_n \Rightarrow^0 (p, S)$ v G, $d = \varepsilon$ a $y = S$. Konkrétně $(\langle p, S\#^{n-1}\rangle, \varepsilon, S\#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$, takže indukční základ platí.

Indukční hypotéza: Předpokládejme, že tvrzení 5.16.1 platí pro všechna $0 \leq m \leq k$, kde $k \in \mathbb{N}_0$ je nezáporné celé číslo.

Indukční krok: Nechť $(p, S)_n \Rightarrow^{k+1} (q, dy)$ v G, kde $d \in T^*$, $y \in (NT^*)^*$, $p, q \in W$. Protože $k+1 \geq 1$, tak lze vyjádřit $(p, S)_n \Rightarrow^{k+1} (q, dy)$ jako $(p, S)_n \Rightarrow^k (h, buAo)_n \Rightarrow (q, buxo)$ $[(h, A) \to (q, x)]$, kde $b \in T^*$, $u \in (NT^*)^*$, $A \in N$, $h, q \in W$, $(h, A) \to (q, x) \in P$, $\text{maxsuffix}(buxo, (NT^*)^*) = y$ a $\text{maxprefix}(buxo, T^*) = d$. Podle indukční hypotézy, $(\langle h, \text{prefix}(f(uAo\#^n), n)\rangle, w, uAo\#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$, kde $w = \text{maxprefix}(buAo, T^*)$. Jelikož $(p, A) \to (q, x) \in P$, konstrukční krok 2 zavádí pravidlo $\langle q, \text{prefix}(f(uxo\#^n), n)\rangle x \vdash |uA|\langle h, \text{prefix}(f(uAo\#^n), n)\rangle A$. Použitím tohoto pravidla ${}_nM$ simuluje $(h, buAo)_n \Rightarrow (q, buxo)$ jako $(\langle q, \text{prefix}(f(uxo\#^n), n)\rangle, \varepsilon, uxo\#) \Rightarrow (\langle h, \text{prefix}(f(uAo\#^n), n)\rangle, \varepsilon, uAo\#)$ v ${}_nM$. Pokud $uxo \in (NT^*)^*$, tak $uxo = y$ a indukční krok

je dokončen. Nyní předpokládejme $uxo \neq y$, takže $uxo = ty$ a $d = wt$ pro nějaké $t \in T^+$. Všimněme si, že pak $\text{prefix}(f(uxo\#^n), n) = \text{prefix}(f(y\#^n), n)$. Následně ${}_nM$ přidá t pomocí $|t|$ kroků, které budou číst ze vstupu a zapisovat na zásobník, takže $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, t, y\#) \Rightarrow^{|t|} (\langle q, \text{prefix}(f(uxo\#^n), n)\rangle, w, ty\#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$, což také kompletuje tento indukční krok.

Podle předchozího tvrzení pro $y = \varepsilon$, je-li $(p, S)_n \Rightarrow^* (q, d)$ v G, kde $d \in T^*$, $p, q \in W$, pak $(\langle q, \#^n\rangle, d, \#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$. A jelikož R obsahuje pravidla z 1 a 3, máme také $(s, d, \#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#) \Rightarrow (\$, \varepsilon, S\#)$ v ${}_nM$. Takže $d \in L(G)$ implikuje $d \in L({}_nM)$, což znamená, že $L(G, n) \subseteq L({}_nM)$.

Tvrzení 5.16.2. *Nechť* $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, c, by\#) \Rightarrow^m (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ *v* ${}_nM$*, kde* $c, b \in T^*$*,* $y \in (NT^*)^*$*,* $p, q \in W$*,* $m \geq 0$*. Potom* $(p, S)_n \Rightarrow^* (q, cby)$ *v* G*.*

Důkaz tvrzení 5.16.2 (indukcí podle $m = 0, 1, \ldots$)**.**
Indukční základ: Nechť $m = 0$. Pak $c = b = \varepsilon$, $y = S$ a $(\langle q, \text{prefix}(f(S\#^n), n)\rangle, \varepsilon, S\#) \Rightarrow^0 (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$. Protože $(p, S)_n \Rightarrow^0 (p, S)$ v G, tak indukční základ platí.

Indukční hypotéza: Předpokládejme, že tvrzení 5.16.2 platí pro všechna $0 \leq m \leq k$, kde $k \in \mathbb{N}_0$ je nezáporné celé číslo.

Indukční krok: Nechť $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, c, by\#) \Rightarrow^{k+1} (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$, kde $c, b \in T^*$, $y \in (NT^*)^*$, $p, q \in W$ v ${}_nM$. Díky $k+1 \geq 1$ můžeme vyjádřit $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, c, by\#) \Rightarrow^{k+1} (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ jako $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, c, by\#) \Rightarrow \alpha \Rightarrow^k (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$, kde α je konfigurace automatu ${}_nM$, jejíž tvar závisí na posledním kroku:

(A) Předpokládejme, že $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, c, by\#)_s \Rightarrow \alpha$ v ${}_nM$. Podrobněji, nechť je $\alpha = (\langle q, \text{prefix}(f(y\#^n), n)\rangle, \text{prefix}(c, |c| - 1), aby\#)$ s $a \in T$ takové, že $c = \text{prefix}(c, |c| - 1)a$. Potom, $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, c, by\#)_s \Rightarrow (\langle q, \text{prefix}(f(y\#^n), n)\rangle, \text{prefix}(c, |c| - 1), aby\#) \Rightarrow^k (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$. Protože $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, \text{prefix}(c, |c| - 1), aby\#) \Rightarrow^k (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$, tak máme $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, \text{prefix}(c, |c| - 1), by\#) \Rightarrow^k (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$. Podle indukce $(p, S)_n \Rightarrow^* (q, \text{prefix}(c, |c| - 1)aby)$ v G. Z $c = \text{prefix}(c, |c| - 1)a$ pak dostáváme $(p, S)_n \Rightarrow^* (q, cby)$ v G.

(B) Na druhou stranu, předpokládejme, že $(\langle q, \text{prefix}(f(y\#^n), n)\rangle, \varepsilon, by\#)_r \Rightarrow \alpha$ v ${}_nM$. Podrobněji, uvažujme $\alpha = (\langle o, \text{prefix}(f(uAv\#^n), n)\rangle, \varepsilon, uAv\#)$ a $(\langle q, \text{prefix}(f(uxv\#^n), n)\rangle, \varepsilon, uxv\#)_r \Rightarrow (\langle o, \text{prefix}(f(uAv\#^n), n)\rangle, \varepsilon, uAv\#)$ použitím pravidla r_1: $\langle q, \text{prefix}(f(uxv\#^n), n)\rangle x \vdash |f(uA)|\langle o, \text{prefix}(f(uAv\#^n), n)\rangle A \in R$ zavedeným v konstrukčním kroku 2, kde $A \in N$, $u \in (NT^*)^*$, $v \in (N \cup T)^*$, $o \in W$, $|f(uA)| \leq n$, $by\# = uxv\#$. Podle indukční hypotézy, $(\langle o, \text{prefix}(f(uAv\#^n), n)\rangle, c, uAv\#) \Rightarrow^k (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ implikuje $(p, S)_n \Rightarrow^* (o, cuAv)$ v G. Protože $r_1 \in R$, $(o, A) \rightarrow (q, x) \in P$ a $A \notin {}_Gstates(f(u))$. Takže, $(p, S)_n \Rightarrow^* (o, cuAv)_n \Rightarrow (q, cuxv)$ v G. A protože $by\# = uxv\#$, tak platí i $(p, S)_n \Rightarrow^* (q, cby)$ v G.

Uvažujme předchozí tvrzení 5.16.1 s $b = y = \varepsilon$, abychom mohli vidět, že $(\langle q, \text{prefix}(f(\#^n), n)\rangle, c, \#) \Rightarrow^* (\langle p, S\#^{n-1}\rangle, \varepsilon, S\#)$ v ${}_nM$ implikuje $(p, S)_n \Rightarrow^* (q, c)$ v G. Nechť $c \in L({}_nM)$. Pak, $(s, c, \#) \Rightarrow^* (\$, \varepsilon, S\#)$ v ${}_nM$. Při zkoumání způsobu konstrukce ${}_nM$ lze vidět, že $(s, c, \#) \Rightarrow^* (\$, \varepsilon, S\#)$ začíná použitím přechodů pro přesun vstupního symbolu na zásobník a pak následuje pravidlo vytvořené v kroku 1, takže $(s, c, \#)_s \Rightarrow^* (s, \alpha, \beta\#) \Rightarrow (\langle p, A\#^{n-1}\rangle, \alpha, uAv\#)$. Dále si všimněme, že posloupnost přechodů končí $(s, c, \#) \Rightarrow^* (\$, \varepsilon, S\#)$ při použití pravidla

z kroku 3. Tím pádem, lze $(s, c, \#) \Rightarrow^* (\$, \varepsilon, S\#)$ vyjádřit jako $(s, c, \#)_s \Rightarrow^* (s, \alpha, \beta\#) \Rightarrow (\langle p, A\#^{n-1}\rangle, \alpha, uAv\#) \Rightarrow^* (\langle q, \text{prefix}(f(x)\#^n), n)\rangle, \varepsilon, x\#) \Rightarrow (\$, \varepsilon, S\#)$ v ${}_nM$. Takže, $c \in L({}_nM)$ implikuje $c \in L(G, n)$, a tedy $L({}_nM) \subseteq L(G, n)$.

Z $L({}_nM) \subseteq L(G, n)$ a $L(G, n) \subseteq L({}_nM)$ plyne $L(G, n) = L({}_nM)$ a tedy lemma 5.16 platí.

□

Lemma 5.17. *Pro všechna $n \geq 1$, ${}_n\mathcal{L}(\mathbf{RDPDA}) \subseteq {}_n\mathcal{L}(\mathbf{ST})$.*

Konstrukční důkaz lemmatu 5.17. Nechť $n \geq 1$ a ${}_nM = (Q, T, V, R, s, S, F)$ je zprava-doleva redukující hluboký zásobníkový automat hloubky n. Nechť Z a \$ jsou dva nové symboly, které se nevyskytují v žádné komponentě automatu ${}_nM$. Položme $N = V - T$. Zaveďme množiny $C = \{\langle q, i, \triangleright\rangle |\ q \in Q,\ 1 \leq i \leq n-1\}$, $D = \{\langle q, i, \triangleleft\rangle |\ q \in Q,\ 0 \leq i \leq n-1\}$. Dále mějme abecedu W s $\text{card}(V) = \text{card}(W)$ a pro všechna $1 \leq i \leq n$, abecedy U_i takové, že pro všechny platí $\text{card}(U_i) = \text{card}(N)$. Bez ztráty na obecnosti můžeme předpokládat, že V, Q a všechny nově zavedené množiny a abecedy jsou vzájemně disjunktní. Dále položme $U = \bigcup_{i=1}^{n} U_i$. Zaveďme bijekci h z V do W. Pro každé $1 \leq i \leq n$ mějme další bijekci ${}_ig$ z N do U_i. Nyní definujme ekvivalentní stavovou gramatiku

$$G = (V \cup W \cup U \cup \{Z\}, Q \cup C \cup D \cup \{\$, z\}, T, P, Z),$$

kde P je vytvořena podle následující posloupnosti kroků:

1. přidejme $(z, Z) \to (\langle z, 1, \triangleright\rangle, h(S))$ do P;
2. pro každé $q \in Q$, $A \in N$, $1 \leq i \leq n-1$,
   přidejme $(\langle q, i, \triangleright\rangle, A) \to (\langle q, i+1, \triangleright\rangle, {}_ig(A))$ a $(\langle q, i, \triangleleft\rangle, {}_ig(A)) \to (\langle q, i-1, \triangleleft\rangle, A)$ do P;
3. je-li $qxY \vdash ipA \in R$, pro některé $p, q \in Q$, $A \in N$, $x \in V^*$, $Y \in V$, $i = 1, \ldots, n$, pak přidejme
   $(\langle p, i, \triangleright\rangle, A) \to (\langle q, i-1, \triangleleft\rangle, xY)$ a
   $(\langle p, i, \triangleright\rangle, h(A)) \to (\langle q, i-1, \triangleleft\rangle, xh(Y))$ do P;
4. pro každé $q \in Q$, $A \in N$, přidejme $(\langle q, 0, \triangleleft\rangle, A) \to (\langle q, 1, \triangleright\rangle, A)$ a
   $(\langle q, 0, \triangleleft\rangle, h(Y)) \to (\langle q, 1, \triangleright\rangle, h(Y))$ do P;
5. pro každé $q \in F$, $a \in T$, přidejme $(\langle q, 0, \triangleleft\rangle, h(a)) \to (\$, a)$ do P.

Hlavní myšlenka důkazu lemmatu 5.17. G simuluje s obráceným efektem aplikace pravidel $qx \vdash ipA \in R$. G načítá (zleva-doprava) větnou formu a počítá výskyty neterminálních symbolů, dokud nedosáhne i-tého výskytu neterminálu. Jestliže je tento symbol A, provádí G přepis A na řetězec x, což koresponduje redukování x na A v automatu ${}_nM$. G dokončí simulaci redukce řetězce x automatem ${}_nM$ tak, že si poznačí vždy poslední symbol pomocí bijekce h (zajistí, že není ze vstupní abecedy) a v posledním kroku tento symbol přepíše na vstupní symbol, a umožní tak dokončení generování x v G. Bijekce h je tedy jakousi kompenzací neexistence koncového stavu ve stavové gramatice G.

Formální důkaz, že $L(G, n) = L({}_nM)$ je vynechán. □

Věta 5.18. *Pro všechna $k \geq 1$, ${}_k\mathcal{L}(\mathbf{RDPDA}) = {}_k\mathcal{L}(\mathbf{ST})$.*

Důkaz věty 5.18. Věta 5.18 plyne z lemmat 5.16 a 5.17.

■

Nyní popišme některé otevřené problémy z probírané oblasti $RDPDA$.

Nechť ${}_n\mathcal{L}({}^{\mathbf{lr}}\mathbf{RDPDA})$ je třída jazyků akceptovaných redukujícími hlubokými zásobníkovými automaty hloubky i, kde $1 \leq i \leq n$, které zpracovávají vstupní řetězec zleva doprava.

Zopakujme část definice 2.6 o reverzaci řetězce a o reverzním jazyku. Reverzace řetězce w, značíme $\mathrm{rev}(w)$, je w zapsáno v opačném pořadí. Reverzace jazyka L, $\mathrm{rev}(L)$, je potom definována jako $\mathrm{rev}(L) = \{\mathrm{rev}(w) |\ w \in L\}$.

Otevřený problém 5.3. Uvažujme otázku—Je třída ${}_n\mathcal{L}(\mathbf{RDPDA})$ pro libovolné $n \geq 1$ uzavřena vůči operaci reverzace? Odpověď je podstatná pro další otevřený problém—Je podstatné pro vyjadřovací sílu $RDPDA$, zdali čte vstupní pásku zprava doleva nebo zleva doprava?

Modifikace $RDPDA$

Následující modifikace jsou prezentovány pouze neformálně a je k nim diskutována jejich předpokládaná vyjadřovací síla (ovšem bez důkazů). Všechny nastíněné varianty mají odpovídající obraz v hlubokých zásobníkových automatech v předchozí podsekci.

$RDPDA$ s vymazávajícími pravidly. Dosud jsme vymazávající pravidla nepřipouštěli. V případě, že povolíme zásobníku jak redukovat svůj obsah, tak také rozšiřovat prostřednictvím vkládání nevstupních symbolů přes redukci prázdného symbolu, získáme nový typ $RDPDA$ neznámých vyjadřovacích schopností. Je také patrné, že zavedením vymazávajících pravidel do $RDPDA$ bychom znatelně zvýšili nedeterminismus, kdy nevíme, kde se má redukovaný prázdný řetězec přesně nacházet, pouze hloubka i ohraničuje výskyt $(i-1)$-ním a i-tým nevstupním symbolem. Povolení vymazávajících pravidel patří rozhodně mezi obecné modifikace formálních modelů, kterými je možné se zabývat.

Symbolově-závislý $RDPDA$. Nyní požadujme, aby nebyl kontrolován pouze počet nevstupních symbolů před vzniknuvším, ale počet výskytů právě toho samého symbolu.

Definice symbolově-závislého $RDPDA$ je identická se základní verzí $RDPDA$ (viz definice 4.11), až na definici redukčního kroku mezi dvěma konfiguracemi:

$$(q, w, uvz) \,{}_r\!\Rightarrow (p, w, uAz) \ [qv \vdash mpA], \text{ kde } \mathrm{occur}(A, u) = m - 1 \text{ (místo původní podmínky } \mathrm{occur}(\Gamma - \Sigma, u) = m - 1).$$

Příklad 5.2. Jazyk $\{a^n b^n c^n |\ n \geq 1\}$ může být popsán symbolově-závislým $RDPDA$ s těmito pravidly:

$$sab \vdash 1pA,\ pc \vdash 1qC,\ qaAb \vdash 1tA,\ tcC \vdash 1qC,\ qAC \vdash 1fS.$$

V tomto modifikovaném přepisujícím systému lze zavést nad n-limitovaností ještě obecnější typ omezování, a to (n, M)-limitovanost, kdy omezujeme přepis na aplikaci na jeden z prvních n výskytů některého ze symbolů z množiny M.

Výsledky k těmto modifikacím budou předmětem budoucího výzkumu.

5.2 Nekonečné hierarchie tříd jazyků

Z výsledků předchozí sekce můžeme odvodit několik různých i stejných nekonečných hierarchií tříd jazyků. Obecnějším závěrem této kapitoly je fakt, že většina přepisujících systémů s omezováním jejich dynamické složitosti vede na tvorbu hierarchií tříd jazyků právě v závislosti na omezovaném parametru (např. konečný index, hloubka, n-limitovanost).

5.2.1 Založené na konečném indexu

Konečný index je velmi restriktivní omezení konfigurace přepisujícího systému. Většina řízených přepisujících systémů s tímto omezením spadají do jediné třídy, která se nachází v prostoru mezi regulárními a kontextovými jazyky (např. viz rovnice 3.1 na straně 37).

Věta 5.19. *Pro každé celé číslo $k \geq 1$ platí, že $\mathcal{L}_k(\mathbf{CF\#RS}) \subset \mathcal{L}_{k+1}(\mathbf{CF\#RS})$.*

Důkaz věty 5.19. V roce 1980 dokázal Gheorge Păun nekonečnou hierarchii tříd jazyků definovaných programovanými gramatikami indexu k (viz [58] nebo strana 160, věta 3.1.7 v monografii [12]) pomocí jazyků $\{b(a^ib)^{2k} \mid i \geq 1\}$, $k \geq 1$.

Věta 5.19 plyne z věty 5.3 a z této nekonečné hierarchie. Tento teoretický výsledek, včetně představení #-přepisujících systémů, je publikován v [37].

■

Věta 5.20. *Pro všechna $n \geq 1$, $\mathcal{L}(n\text{-}\mathbf{RLIN\#RS}) \subset \mathcal{L}((n+1)\text{-}\mathbf{RLIN\#RS})$.*

Důkaz věty 5.20. Připomeňme, že $\mathcal{R}^m_{[n]} \subset \mathcal{R}^m_{[n+1]}$, pro všechna $m, n \geq 1$ (viz věta 8 v [77]) a tedy i pro $m = 1$. Věta tedy plyne z platnosti věty 5.7, která dokazuje $\mathcal{L}(n\text{-}\mathbf{RLIN\#RS}) = \mathcal{R}^1_{[n]}$.

■

Nyní ustanovíme nekonečnou hierarchii tříd jazyků definovanou zobecněnými #-přepisujícími systémy na základě omezení na konečný index.

Věta 5.21. *Pro všechna $k \geq 1$, $\mathcal{L}_k(\mathbf{G\#RS}) \subset \mathcal{L}_{k+1}(\mathbf{G\#RS})$.*

Důkaz věty 5.21. $\mathcal{L}_k(\mathbf{G\#RS}) = \mathcal{L}_k(\mathbf{CF\#RS})$ plyne z věty 5.11. Připomeňme, že $\mathcal{L}_k(\mathbf{P}) = \mathcal{L}_k(\mathbf{CF\#RS})$ (viz [34]) a $\mathcal{L}_k(\mathbf{P}) \subset \mathcal{L}_{k+1}(\mathbf{P})$ pro každé $k \geq 1$ (viz věty 3.1.2i a 3.1.7 v [12]). Pak platí také věta 5.21.

■

5.2.2 Založené na n-limitovanosti

V roce 1970 zavedl Kasai tzv. stavové gramatiky (viz definice 2.21 nebo původní článek [21]) a dokázal, že v případě omezení počtu neterminálů, se kterými lze při derivaci věty pracovat (tzv. n-limitovanost, viz definice 3.1), vzniká nekonečná hierarchie. Demonstrací ekvivalence modelu n-limitovaných stavových gramatik s nově zavedenými zprava-doleva redukujícími hlubokými zásobníkovými automaty dostáváme opět nekonečnou hierarchii i pro tento nový model. V tomto případě je oproti stavovým gramatikám ona n-limitovanost vyjádřena maximální hloubkou redukcí, která je dána množinou pravidel redukujícího hlubokého zásobníku.

Věta 5.22. *Pro všechna $k \geq 1$, ${}_k\mathcal{L}(\mathbf{RDPDA}) \subset {}_{k+1}\mathcal{L}(\mathbf{RDPDA})$.*

Důkaz věty 5.22. Věta 5.18 říká, že pro všechna $k \geq 1$, ${}_k\mathcal{L}(\mathbf{RDPDA}) = {}_k\mathcal{L}(\mathbf{ST})$. Takže tato věta 5.22 je důsledkem věty 5.18 a věty 2.5 z [21].

■

V této kapitole byly dokázány hlavní autorovy výsledky, jejichž zaměření bylo na snižování vyjadřovací síly omezovaných přepisujících systémů. Některé typy omezování naopak zvyšovaly vyjadřovací schopnosti. Ve většině případů jsme navíc získali nekonečné hierarchie tříd jazyků závislé na použitém přepisujícím systému, na typu omezování a na konkrétní omezující konstantě (viz sekce 5.2).

Část III

Relace k praxi

Kapitola 6

Využití omezování konfigurací

V této kapitole se podíváme na potenciál praktického využití nově představených přepisujících systémů a případně i jejich omezování. Již tradičně se zaměříme na #-přepisující systémy.

Nejprve se zamyslíme nad možnostmi a zajímavými úlohami, které mají předpoklady jakožto aplikační oblasti pro tyto nové systémy. Dále budeme studovat vlastnosti esenciální pro praxi jako jsou determinismus a kanonické přepisování. Nakonec se pokusíme shrnout problémy při snaze o využití těchto systémů v syntaktické analýze.

6.1 Hledání aplikací

Hledání nevykonstruovaných aplikací je u nových řízených přepisujících systémů (bez přímé motivace v praktické úloze přímo při definování tohoto systému) obvykle velmi obtížné a často se od něj upouští. Vzhledem k omezovací povaze předchozích přístupů asi nelze hledat využití v příliš náročných problémech jako je popis přirozeného jazyka či zcela obecný popis kontextových vlastností programovacích jazyků.

Hlavní oblasti praktického využívání formálních jazyků bychom mohli shrnout do několika následujících bodů:

- překladače (popis a zpracování programovacích jazyků tvoří jeden ze základních pilířů informatiky)
- lingvistika (studuje vztah přirozených a formálních jazyků, snaží se o formální popis a zpracování přirozených jazyků)
- biologicky motivované systémy (např. v oblasti mikrobiologie, bioinformatiky, genetiky atd.)
- modelování (formální jazyk je formálním modelem, a proto bezesporu slouží i pro obecné modelování či verifikaci; viz např. [7])

Všeobecně praktická úloha u přepisujících systémů je řešení problému členství dané věty do jazyka, který je tímto systémem definován. Tuto úlohu nejlépe řeší různé přepisující systémy akceptačního typu (např. deterministické automaty různých typů).

Nyní zmiňme některé zajímavé konkrétnější náměty na praktické využití omezování konfigurací přepisujících systémů:

- studovat vztah k-limitovanosti a syntaktických analyzátorů; například případ hlubokých zásobníkových automatů hloubky k a LL(k) syntaktické analýzy shora dolů nebo analogicky redukujících hlubokých zásobníkových automatů hloubky k a LR(k) syntaktické analýzy (viz [70]);
- využití omezování počtu aktivních symbolů při syntaktické analýze (v případě nedeterministických výpočtů se díky omezení na konečný index nebo n-limitovanost omezí prohledávaný stavový prostor)
- zavedení srovnávací míry složitosti modelů prostřednictvím počtu pravidel (normalizovaných) pro získání stejné věty v rámci dvou různých ekvivalentních přepisujících systémů (tj. modelů definujících stejný jazyk); tento námět má vztah ke studiu dynamické i popisné složitosti omezených přepisujících systémů
- využití řízení i omezování jako prostředku pro zlepšení rozhodovacích schopností přepisujícího systému, který přepis aplikovat v momentě rozhodovacího procesu výběru pravidla (např. řízené zásobníkové automaty nebo zde představené zásobníkové automaty s omezeným obsahem zásobníku či deterministické #-přepisující systémy typu 2).

6.2 Vhodné modifikace formálních modelů

Jedním ze signifikantních modifikací formálních modelů klíčových pro jejich praktické využití je determinismus. Počítače již od svého počátku pracují podle zadaného přesného předpisu, algoritmu. Tento algoritmus však musí být vždy striktně deterministický, abychom byli schopni výpočet zopakovat se stejným výsledkem (nemluvme o problémech jako generování náhodného čísla a podobně, o kterých tato práce nepojednává).

V případě Turingových strojů a odpovídajících modelů (např. neomezených gramatik) nesnižuje determinismus vyjadřovací sílu, ale pouze efektivnost výpočtu (deterministická simulace nedeterminismu vyžaduje vyšší prostorovou a v podstatě i časovou složitost). U nižších modelů je to však jinak. Výhodou determinismu u jednodušších modelů bývá podstatně nižší a pro praxi přijatelná časová složitost (např. lineární).

Determinismus je tedy klíčový především pro efektivní práci daného formálního modelu (např. přijímání věty deterministickým zásobníkovým automatem).

Definice determinismu u konkrétního řízeného formálního modelu je jen velmi zřídka matematicky jednoduše popsatelná a pokud ano, tak většinou dostáváme v praxi nevyužitelné výsledky (příliš nízká efektivita nebo přílišné omezení vyjadřovací síly modelu).

Tuto situaci nyní demonstrujme na dvou typech determinismu #-přepisujících systémů.

6.2.1 Deterministické #-přepisující systémy

Tyto systémy jsou svou povahou generativní, a to implikuje zhoršenou schopnost definice determinismu (analogicky např. bezkontextové gramatiky versus zásobníkové automaty).

V této podsekci se zaměříme na bezkontextové #-přepisující systémy, ale všechny typy determinismu lze analogicky definovat i pro pravě-lineární a zobecněné tvary pravidel.

Typ 1

První způsob definice determinismu, který nás může napadanout, je omezit pravidla systému tak, aby v aktuálním stavu pro konkrétní přepisovanou hranici nebylo možno najít více jak jedno pravidlo, a tak zajistit jeho přímočařejší výběr (místo obecného nedeterministického).
Definice 6.1. Speciální případ bezkontextového #-přepisujícího systému $H = (Q, \Sigma, R, s)$, kdy pro každé $p \in Q$ a každé kladné celé číslo $i \in \mathbb{N}$ bude $p_i\#$ levá strana nejvýše jednoho pravidla z H, tj. pro každou dvojici $p \in Q$ a $i \in \mathbb{N}$ platí $\mathrm{card}(\{r \mid r\colon p\,{}_i\# \to q\ x \in R, q \in Q, x \in \Sigma^*\}) \leq 1$, budeme nazývat *deterministický bezkontextový #-přepisující systém typu 1* (zkráceně det_1CF#RS).

Obecně deterministický #-přepisující systém typu X značme zkráceně det_XCF#RS a třídu jazyků definovanou těmito systémy označujme $\mathcal{L}(\mathbf{det_X CF\#RS})$.

Například systém H z příkladu 4.1 na straně 44 je evidentně nedeterministický, protože jeho druhé a čtvrté pravidlo má totožnou levou stranu.

Při hlubším zamyšlení si uvědomíme, že pro aplikaci takto deterministického systému v generování věty máme stále problém určit, kterou hranici je nejvhodnější v daném kroku přepsat. Tento problém determinismu z definice 6.1 přímo ovlivňuje také následující výsledky.

Věta 6.1. $\mathcal{L}(\mathbf{CF\#RS}) = \mathcal{L}(\mathbf{det_1 CF\#RS})$.

Důkaz věty 6.1. Protože det_1CF#RS je pouze speciální případ CF#RS, potřebujeme pouze dokázat, že $\mathcal{L}(\mathbf{CF\#RS}) \subseteq \mathcal{L}(\mathbf{det_1 CF\#RS})$.
Nechť $H = (Q_H, \Sigma, s_H, R_H)$ je CF#RS, kde $T = \Sigma - \{\#\}$. Sestrojme det_1CF#RS, $D = (Q_D, \Sigma, s_H, R_D)$, kde R_D a Q_D jsou zkonstruovány podle těchto kroků:

1. Pro každé $p \in Q_H$ a $i \in \mathbb{N}$ položme ${}_p^i R_H = \{r\colon p\,{}_i\# \to q\ x \mid r \in R_H, q \in Q_H, x \in \Sigma\}$;
2. Položme $Q_D = Q_H$ a pomocné $R' = \bigcup\{{}_p^i R_H \mid \mathrm{card}({}_p^i R_H) \leq 2\}$;
3. Pro každou množinu ${}_p^i R_H$ s $\mathrm{card}({}_p^i R_H) \geq 3$ proveďme následující algoritmus:
   $o := p$;
   `while` $(\mathrm{card}({}_p^i R_H) \geq 3)$ `do:`
   - odstraňme r z množiny ${}_p^i R_H$
   - přidejme $\langle {}_p^i R_H \rangle$ do Q_D
   - přidejme $o\,{}_i\# \to \mathrm{rhs}(r)$ a $o\,{}_i\# \to \langle {}_p^i R_H \rangle\ \#$ do R';
   - $o := \langle {}_p^i R_H \rangle$;
4. Položme $R_D = \bigcup\{{}_p^i R' \mid \mathrm{card}({}_p^i R') = 1\}$.
5. Nechť $<$ je ostré uspořádání na $\mathrm{Lab}(R')$. Pro každou dvojici pravidel $r_i\colon p\,{}_i\# \to q_1\ x_1$ a $r_j\colon p\,{}_i\# \to q_2\ x_2$ z R' takovou, že $r_i < r_j$, přidejme do R_D následující pravidla:
   - $p\,{}_i\# \to \langle r_i, r_j \rangle\ \#\#$
   - $\langle r_i, r_j \rangle\,{}_i\# \to \langle r_i \rangle\ \#$
   - $\langle r_i, r_j \rangle\,{}_{i+1}\# \to \langle r_j \rangle\ \#$
   - $\langle r_i \rangle\,{}_i\# \to \langle r_i' \rangle\ x_1$
   - $\langle r_j \rangle\,{}_i\# \to \langle r_j' \rangle\ x_2$
   - $\langle r_i' \rangle\,{}_{i+1}\# \to q_1\ \varepsilon$

- $\langle r'_j \rangle\ {}_{i+1}\# \rightarrow q_2\ \varepsilon$

a nově vzniklé stavy $\langle r_i, r_j \rangle, \langle r_i \rangle, \langle r_j \rangle, \langle r'_i \rangle, \langle r'_j \rangle$ přidejme do Q_D.

Hlavní myšlenka důkazu věty 6.1.

Krok 1. Podmnožiny ${}_p^i R$ označují množiny pravidel se stejnou levou stranou, takže kardinalita těchto množin určuje počet pravidel se stejnou levou stranou (tzv. stupeň nedeterminismu).

Kroky 2-3. Množina pravidel, R_H, je transformována do nové množiny pravidel R' tak, aby obsahovala pro každou levou stranu nejvýše dvě pravidla se stejnou levou stranou.

Krok 4. Ve čtvrtém kroku je nová množina pravidel, R_D, inicializována všemi již deterministickými pravidly z R'. Zbývá se již vypořádat pouze s nedeterministickými pravidly, která mají od každé kombinace levé strany pravidel právě dva exempláře.

Krok 5. Každá dvojice pravidel se stejnou levou stranou je simulována posloupností sedmi nových pravidel v R_D: (1) vygenerování pomocné $(i+1)$-ní hranice, (2 a 3) provedení výběru jednoho z oněch dvou nedeterministických pravidel k simulaci, (4 a 5) přepis i-té hranice, (6 a 7) změna stavu na cílový stav a vymazání pomocné hranice.

■

Příklad 6.1. Ukažme si na příkladě konstrukci z důkazu věty 6.1, která transformuje CF#RS H z příkladu 4.1 na $\det_1$CF#RS D. Protože stupeň nedeterminismus je menší než tři, můžeme přeskočit kroky 1 až 3. Pak nakopírujeme všechna deterministická pravidla do R_D podle popisu v kroku 4, takže $R_D = \{1\colon s\ {}_1\# \rightarrow p\ \#\#, 3\colon q\ {}_2\# \rightarrow p\ \#c, 5\colon f\ {}_1\# \rightarrow f\ c\}$ a $Q_D = \{s, p, q, f\}$. Nakonec pomocí kroku 5 vygenerujeme posloupnost pravidel simulující dvojici nedeterministických pravidel: $2\colon p\ {}_1\# \rightarrow q\ a\#b$ a $4\colon p\ {}_1\# \rightarrow f\ ab$.

Protože $r_i = 2$, $r_j = 4$, $p = p$, $i = 1$, $q_1 = q$, $q_2 = f$, $x_1 = a\#b$, $x_2 = ab$, přidejme následující pravidla do R_D:

$p\ {}_1\# \rightarrow \langle 2, 4 \rangle\ \#\#$
$\langle 2, 4 \rangle\ {}_1\# \rightarrow \langle 2 \rangle\ \#$
$\langle 2, 4 \rangle\ {}_2\# \rightarrow \langle 4 \rangle\ \#$
$\langle 2 \rangle\ {}_1\# \rightarrow \langle 2' \rangle\ a\#b$
$\langle 4 \rangle\ {}_1\# \rightarrow \langle 4' \rangle\ ab$
$\langle 2' \rangle\ {}_2\# \rightarrow q\ \varepsilon$
$\langle 4' \rangle\ {}_2\# \rightarrow f\ \varepsilon$

Na konci $Q_D = \{s, p, q, f, \langle 2, 4 \rangle, \langle 2 \rangle, \langle 4 \rangle, \langle 2' \rangle, \langle 4' \rangle\}$.

Důsledek 6.2. *Pro všechna $k \geq 1$, $\mathcal{L}_k(\mathbf{CF\#RS}) \subseteq \mathcal{L}_{k+1}(\mathbf{det_1 CF\#RS})$.*

Důkaz důsledku 6.2. Konstrukční důkaz věty 6.1 potřebuje zvýšit počet hranic v konfiguraci vždy maximálně o jedna, takže proto tento důsledek platí. ■

Věta 6.3. *Pro $k = 1$ a libovolný jazyk $L \in \mathcal{L}_k(\mathbf{det_1 CF\#RS})$ platí, že* $\mathrm{card}(L) \leq 1$.

Důkaz věty 6.3. Předpokládejme, že $k = 1$ a $\text{card}(L) > 1$. Determinismus nám zaručuje, že každé pravidlo má unikátní levou stranu, takže stavy a jejich programování konečně-stavovým řízením nám nedovolují větvení nebo možnost výběru z množiny pravidel. Tím pádem nemáme možnost výběru ani v samotném výpočtu. Nemáme tedy možnost v $\det_1$CF#RS, jak vygenerovat dvě odlišné konfigurace ze stejné počáteční konfigurace, $s\#$ (nezáleží ani na tom, zda to zkoušíme jedním nebo více výpočetními kroky). Tímto jsme dospěli ke sporu, takže předpoklad, že $\text{card}(L) > 1$ pro $k = 1$ neplatí. ■

Hypotéza 6.1. Hypotézou je existence konstrukčního algoritmu, který při determinizaci libovolného CF#RS indexu k nezvýší počet hranic. Symbolicky zapsáno platí pro každé $k \geq 2$, že $\mathcal{L}_k(\mathbf{CF\#RS}) = \mathcal{L}_k(\mathbf{det_1CF\#RS})$.

Částečnou odpověď na tuto hypotézu se pokusme dát nyní.

Máme totiž možnost negenerovat pomocnou hranici jako $(i + 1)$-ní, ale například pomocí funkce modulo (vrací zbytek po celočíselném dělení) vybrat libovolnou jinou hranici, která zrovna není určena k přepisu. Například máme-li přepisovat druhou hranici, můžeme v transformačním algoritmu využít jako pomocnou hranici třetí nebo první, pro $k = 3$, a nemusíme zavádět čtvrtou pomocnou hranici a zvyšovat tak uměle konečný index systému. Tento princip lze však uplatnit pouze pro $k \geq 2$. Dostali bychom však ještě podrobnější výsledek než ve větě 6.1, že $\mathcal{L}_k(\mathbf{CF\#RS}) = \mathcal{L}_k(\mathbf{det_1CF\#RS})$ pro všechna $k \geq 2$.

Ve skutečnosti způsob dosažení determinismu (podle definice 6.1) z nedeterministického CF#RS je pouhým nahrazením jednoho nedeterminismu jiným, skrytějším. Původní otázku: „Kterou hranici nahradíme a které pro to použijeme pravidlo?“ jsme transformovali na novou specifičtější otázku: „Kterou hranici v rozhodujícím okamžiku přepíšeme?“ Nyní již nemusíme řešit, které pravidlo použijeme, pokud jsme se již rozhodli pro konkretní hranici k přepsání.

V dalších odstavcích se proto zamyslíme nad další možností definice determinismu #-přepisujících systémů, která by mohla být blíže praktické použitelnosti (například v syntaktické analýze).

Typ 2

U generujících formálních modelů je všeobecně problém intuitivně stanovit podmínky determinismu. Většinou nedostačují triviální požadavky na tvar pravidel, ale je třeba jít s analýzou pravidel více do hloubky. Příkladem jsou i jednoduché bezkontextové gramatiky, které definují deterministické bezkontextové gramatiky na základě splňování sady podmínek pro každé pravidlo. Stručně řečeno, na základě několika málo symbolů ze vstupu jsme schopni se jednoznačně rozhodnout, které pravidlo použít, a v případě, že věta opravdu patří do popisovaného jazyka, tak nikdy nebudeme potřebovat v některém dalším kroku navracení (*backtracking*, tj. zahození části již provedeného výpočtu a provedení nového výpočtu jinou cestou).

Problém definice použitelného determinismu u řízených formálních modelů se ještě prohlubuje s novými možnosti, jak k determinismu přistupovat a do jaké míry jej vyžadovat (např. pouze na úrovni řízení, pouze na úrovni přepisu, obojí, atd.).

V definici determinismu typu 2 je volnější determinismus při přepisu první hranice (analogické s LL-podmínkami v LL(1) gramatikách, viz [70]) a přísnější determinismus při přepisu ostatních hranic, abychom se vyvarovali nutnosti pracovat (číst) ve vstupu na několik místech. S výhodou by tento přístup mohl být praktikován také na hluboké zásobníkové automaty.

Koncept 6.1. Mějme bezkontextový #-přepisující systém $H = (Q, \Sigma, R, s)$, který nazvěme deterministickým typu 2, pokud platí následující.

V každém stavu existuje nejvýše jedno pravidlo manipulující s jinou než první hranicí. Navíc v tomto stavu již nedovolujeme existenci dalšího pravidla, které by přepisovalo první hranici. V případě pravidel pracujících s první hranicí aplikujme omezení analogické k LL(1) gramatikám. To znamená, že pouze při přepisech první hranice je při výběru pravidla prováděno jednoznačné rozhodnutí na základě dvojice informací: (1) aktuálního stavu a (2) čteného symbolu věty. V ostatních přepisech je pravidlo jednoznačně určeno pouze aktuálním stavem. Tento obecný postup demonstrujme na příkladu 6.2.

Příklad 6.2. Pro samotnou definici konceptu determinismu typu 2 nepotřebujeme popisovat způsob práce CF#RS v přijímacím režimu, ale pro ukázku na příkladě si jej popišme.

#-přepisující systém pracuje ve své konfiguraci generativním způsobem, ovšem při rozhodování při přepisu podle některých pravidel má možnost přečíst jeden symbol ze vstupu (značme in) a posunout čtecí hlavu na další symbol. Nepožadujeme přečtení celého vstupního řetězce, ale pouze jeho předpony nutné pro správné řízení přepisů první hranice. Pravidla pro přepis první hranice, která pro rozhodování čtou symbol ze vstupní věty, obsahují podmínku tvaru (in = seznam akceptovaných symbolů tzv. *podmínková množina*). U pravidla, které neobsahuje podmínku, nezáleží na vstupním symbolu a pravidlo žádný symbol nečte. Pravidla se stejným výchozím stavem musí mít všechny podmínkové množiny navzájem disjunktní, aby nedošlo k nerozhodnosti.

Mějme následující pravidla deterministického CF#RS typu 2:

1: $s\,{}_1\# \rightarrow p\ \#\#$
2: $p\,{}_1\# \rightarrow q\ a\#b\ (in = \{a\})$
3: $q\,{}_2\# \rightarrow p\ c\#$
4: $p\,{}_1\# \rightarrow f\ \varepsilon\ (in = \{b\})$
5: $f\,{}_1\# \rightarrow f\ \varepsilon$

Zpracování věty $aabbcc$ pak vypadá následovně: $s\# \Rightarrow p\#\#\ [1] \Rightarrow qa\#b\#\ [2\ (in = a)] \Rightarrow pa\#bc\#\ [3] \Rightarrow qaa\#bbc\#\ [2\ (in = a)] \Rightarrow paa\#bbcc\#\ [3] \Rightarrow faabbcc\#\ [4\ (in = b)] \Rightarrow faabbcc\ [5]$.

Hlavní problém, který jsme se snažili definicí determinismu typu 2 odstranit, je problém při zpracovávání věty #-přepisujícím systémem při přepisech na různých místech konfigurace, kdy ale nevíme, kterému podřetězci reálné věty daná část konfigurace odpovídá. Touto korespondencí si můžeme být jisti pouze na začátku věty, která v tomto typu determinismu hraje kromě stavu hlavní řídící roli, aniž bychom „zahodili“ možnost pracovat s některými nebezkontextovými jazyky (viz příklad 6.2).

Závěrem tedy je, že matematicky elegantně popsatelný determinismus daný model buď téměř neomezí, takže není skutečně deterministický pro použití (viz typ 1), a nebo je takovýto determinismus omezující až přespříliš (typ 2).

6.2.2 Kanonické #-přepisující systémy

Kanonické derivace v bezkontextových gramatikách mají zásadní význam v syntaktické analýze a teorii překladačů. Z toho plyne také motivace ke studiu kanonických (především nejlevějších) přepisů v nově zavedených formálních modelech, jako například #-přepisujících systémech. Kanoničnost znamená, že se při přepisování snažíme postupovat více systematicky a deterministicky v tom, že přepisujeme výhradně nejlevější aktivní symboly. Tento pojem koreluje s omezováním na n-limitovanost. Pojem kanoničnost budeme chápat obecně jako přepisování několika nejlevějších aktivních symbolů. V případě nejlevějšího (nebo zkráceně levého) přepisu budeme uvažovat přepisování skutečně toho nejlevějšího aktivního symbolu (nepočítaje komponentu konfigurace pro aktuální stav, viz poznámka 6.1).

Různými typy kanonických derivací v řízených gramatikách, které jsou jedním z inspiračních zdrojů pro #-přepisující systémy, se zabývá relativně aktuální článek prof. Fernaua ([15]) a dalších vědců ([10, 12, 42, 59]).

Definujme si kanoničnost výpočetních kroků v #-přepisujících systémech dvěma způsoby:

a) **přísně kanonický** přepis jedné z k nejlevějších hranic bez ohledu na aktuální stav (přímo odpovídá omezování posloupnosti aplikovaných pravidel na k-limitovanost, viz definice 3.1), kde $k \geq 1$;

b) **stavově-řízené kanonické** přepisování pracuje s k-nejlevějšími hranicemi, ovšem s ohledem na aktuální stav, tedy vybereme k nejlevějších výskytů aktivních symbolů, které lze v aktuálním stavu nějakým pravidlem přepsat.

Ve zbytku podsekce, nechť $H = (Q, \Sigma, R, s)$ je #-přepisující systém, $n = \max(\{i \mid p\,{}_i\# \to q\ x \in R\})$ je maximální index přepsatelné hranice pomocí pravidel z R, $k \in \mathbb{N}$.

Definice 6.2. *Přísně k-kanonický výpočetní krok* je výpočetní krok stupně j, kde $j \leq k$. Přísně k-kanonický výpočet je složen pouze z k-kanonických kroků a přísně k-kanonický jazyk $L(H) = \{w \mid s\# \Rightarrow^* qw$ pomocí k-kanonického výpočtu v $H\}$. Třídu těchto jazyků značme $\mathcal{L}(\mathbf{CF\#RS}, k$-přísně kanonické).

Definice 6.3. Pro každý stav $p \in Q$ definujme množinu $K(H,p) = \{i \mid p\,{}_i\# \to q\ x \in R\}$, což je množina indexů všech hranic přepsatelných systémem H ve stavu p. Výpočetní krok $px \Rightarrow qy$ nazveme *stavově-řízený k-kanonický*, pokud $\max(K(H,p)) - \min(K(H,p)) < k$. Analogicky k předchozí definici zaveďme stavově-řízený k-kanonický výpočet a jazyk. Třídu jazyků označujme $\mathcal{L}(\mathbf{CF\#RS}$, stavově-řízené k-kanonické).

Všimněme si, že k má smysl volit pouze v intervalu $1 \leq k \leq n$, protože každý CF#RS je ze své definice implicitně přísně n-kanonický.

Demonstrujme si jádro algoritmu pro převod každého #-přepisujícího systému na stavově-řízený 1-kanonický #-přepisující systém, tedy $\mathcal{L}(\mathbf{CF\#RS}) \subseteq \mathcal{L}(\mathbf{CF\#RS}$, stavově-řízené 1-kanonické).

Algoritmus 6.1 (nástin)**.** Každé pravidlo $r\colon p\,{}_i\# \to q\ x \in R$, kde $i > \min(K(H,p)) = m$, nahraďme novou dvojicí pravidel:

$$p\,{}_m\# \to \langle r \rangle\ \#$$

$$\langle r \rangle\,{}_i\# \to q\ x$$

kde $\langle r \rangle$ je nový stav.

Z tohoto algoritmu 6.1 vyplývá, že druhý typ kanonických výpočtů nebude pro praxi příliš vhodný, a zaměříme se proto dále na přísně k-kanonické přepisování. Pro další snížení nedeterminismu nás v praxi nejvíce zajímá varianta s $k = 1$, tzv. *nejlevější* (nebo zkráceně levý) výpočetní krok. Při tomto omezení množiny pravidel nemají smysl jiná pravidla, než která přepisují právě první hranici.

Mezi hypotézy, ke kterým bude vhodné se v budoucnu vrátit, patří: Každý jazyk definovaný nejlevějším (tj. přísně 1-kanonickým) #-přepisujícím systémem je bezkontextový.

Důkaz by využil taktéž nejlevějších přepisů v bezkontextové gramatice, která by pomocí prvního výskytu neterminálu zachycovala informaci o aktuálním stavu původního #-přepisujícího systému a pomocí dalších výskytů pozice dalších hranic v původní konfiguraci systému.

Otevřený problém 6.2. $\mathcal{L}(\mathbf{CF\#RS}$, 1-přísně kanonické$) \subset \mathbf{CF}$?

Důkaz tohoto otevřeného problému, kromě demonstrace předchozí hypotézy, vyžaduje i důkaz opačného směru, tj. že existuje bezkontextový jazyk, který nelze tímto nejlevějším #-přepisujícím systémem popsat.

Poznámka 6.1. Při detailnějším pohledu na k-kanonické přepisování konfigurací #-přepisujících systémů je třeba poznamenat, že i když se přepisu účastní také aktuální stav, my jej do konstanty k nezapočítáváme. Například pro $k = 1$ jsou ve skutečnosti aplikovány přepisy na prvním a druhém aktivním symbolu celé konfigurace: (1) změna aktuálního stavu v první komponentě a (2) přepis první hranice v druhé komponentě konfigurace.

Závěrem je vhodné podotknout, že z praktického hlediska by mohlo být zajímavé studovat kombinaci k-kanonického přepisování a některého typu determinismu.

6.2.3 Poznámka o determinismu hlubokých zásobníkových automatů

Velmi podobnou ideu transformace jako u deterministického #-přepisujícího systému typu 1 lze aplikovat na hluboké zásobníkové automaty deterministické vzhledem k hloubce expanzí (viz definice 4.10 či důsledek 1 v [50]).

Nyní tedy prezentujme jádro transformačního algoritmu, které netvoří úplný důkaz, ale nastiňuje, jak libovolný hluboký zásobníkový automat ${}_nM$ přímo (bez mezipřevodu) převést na deterministický vzhledem k hloubce expanzí.

Algoritmus 6.2 (nástin)**.** Máme-li pro libovolné $p \in Q$ dvě pravidla $r' \colon ipA \to qu \in R$ a $r'' \colon jpB \to ov \in R$, kde $i \neq j$ (tj. pravidla r' a r'' porušují determinismus vzhledem k hloubce expanzí), pak tato dvě pravidla nahraďme pravidly

$$r' \colon ipA \to qu$$

$$\rho \colon ipA \to \langle r'' \rangle A$$

$$\rho' \colon j\langle r'' \rangle B \to ov$$

kde ρ, ρ' jsou nová unikátní návěští pravidel a $\langle r'' \rangle$ je nový stav. Tento výpočet iterativně opakujme, dokud nedostaneme R obsahující pouze deterministická pravidla vzhledem k hloubce expanzí.

Otevřenou otázkou stále zůstává, zda by i silný determinismus hlubokých zásobníkových automatů bylo možno zajistit podobným, avšak rozšířeným trikem.

6.3 Syntaktická analýza

Syntaktická analýza provádí jako nejčastější úlohu rozhodnutí, zda zadaná věta patří nebo nepatří do jazyka definovaného daným přepisujícím systémem ([1, 69]).

Obecný problém řízených přepisujících systémů se silou menší jak kontextové jazyky je přepis na více místech konfigurace, který není ani náhodný ani nejlevější, ale většinou je řízen konečněstavovým řízením, které však může být také nedeterministické. V případě jak generativních, tak akceptačních systémů je tedy nutné mít možnost predikovat, které aktivní symboly odpovídají jakým podřetězcům výsledné věty, což je většinou zásadním problémem. Nejpřímočařejší, ale dosti omezující řešení může být požadavek na disjunktnost množiny počátečních předpon takovýchto podřetězců, čímž zajistíme jejich pohodlnou identifikaci.

6.3.1 Programovací jazyky

Nejčastější využití syntaktické analýzy a případně následného překladu je v oblasti programovacích jazyků ([19, 38, 75, 76, 79]). Programovací jazyky jsou většinou specifikovány pomocí bezkontextové gramatiky, jejíž pravidla obsahují dodatečné tzv. kontextové podmínky, které jsou do výsledného analyzátoru nebo překladače zapracovávány víceméně manuálně.

#-přepisující systémy bohužel neumí popsat všechny bezkontextové jazyky, například jazyk pro výraz (viz příklad 4.2 pro předem neomezený počet různých typů závorek), což má zásadní vliv na jejich využitelnost v oblasti popisu programovacích jazyků. Bohužel si ani neporadí s některými palčivými kontextovými problémy, které se vyskytují v programovacích jazycích.

Například problém *deklarace-definice proměnné*, kdy ve větě programovacího jazyka (programu) vyžaduji, aby použití nějaké proměnné předcházela deklarace této proměnné (ne nutně těsně před použitím proměnné). Záležitost navíc komplikují strukturované programy, kde lze zanořovat bloky a k nim odpovídající lokální proměnné, které mohou přes stejný identifikátor překrývat jiné již deklarované proměnné atd. Tento problém lze zjednodušeně vyjádřit formálním jazykem $\{(wc)^n |\ n \geq 1, w \in (\Sigma - \{c\})^*\}$ (viz [12]), kde w představuje identifikátor proměnné; první výskyt w odpovídá deklaraci a další výskyty w symbolizují definici nebo použití w. c slouží jako oddělovač, který se nevyskytuje v podřetězci w. Přestože #-přepisující systém si s tímto problémem nedokáže poradit úplně, pořád ho zvládá o trochu lépe než bezkontextová gramatika. #-přepisující systém dokáže tento jazyk generovat, pokud dopředu omezíme konstantu n.

Závěr této kapitoly, která se zabývala především vlastnostmi bezkontextových #-přepisujících systémů v relaci s praktickým využitím, je následující:

Představili jsme si několik typů determinismů a nastínili jejich vyjadřovací sílu; obdobně v případě kanonických přepisů. Většina praktických omezení však vede k dalšímu značnému snižování síly #-přepisujících systémů, a tak je jejich využitelnost pro popis náročnějších problémů v oblasti mezi bezkontextovými a kontextovými třídami jazyků diskutabilní.

Část IV

Závěr

Kapitola 7

Závěr

V závěrečné kapitole provedeme tradiční shrnutí výsledků a detailněji se zamyslíme nad budoucím zkoumáním. Krátce zmíníme i některé konkrétní otevřené problémy a hypotézy, jejichž řešení by bylo vhodné hledat v nejbližší budoucnosti.

7.1 Shrnutí

V práci bylo provedeno zobecnění automatů a gramatik pod jednotným pojmem přepisující systém. Přepisující systém mění svůj vnitřní stav přepisováním některých svých částí (podřetězců, symbolů) podle definovaného chování. Pro tento vnitřní stav používáme pojem konfigurace, jenž byl převzat z teorie automatů a zobecněn na přepisující systémy. Konfigurace je konečná posloupnost komponent. Komponenta je řetězec symbolů úplné abecedy. Přepisující kroky mezi jednotlivými konfiguracemi provádíme pomocí přepisujících pravidel. Díky sjednocení pojmů z oblasti gramatik a automatů jsme mohli pohodlněji studovat systémy vzniklé jejich kombinováním (kapitoly 1 a 2).

Dále byla zavedena intuitivní klasifikace omezování a řízení přepisujících systémů za účelem studia jejich dynamické složitosti, což je pohled na složitost trochu z jiného úhlu, než v případě popisné nebo časové a prostorové složitosti. Dynamická složitost sdružuje metriky blízké teorii formálních jazyků podobně jako popisná složitost. Na rozdíl od popisné složitosti, která studuje efektivitu popisu jazyka pomocí instance přepisujícího systému (např. kardinalitu množiny pravidel), dynamická složitost se zaměřuje na efektivitu samotného přepisování při zpracovávání věty jazyka (např. maximální dostačující počet výskytů neterminálů v konfiguraci) (kapitola 3).

Ve 4. a 5. kapitole byly zavedeny nové přepisující systémy, na kterých bylo demonstrováno omezování konfigurací a omezování posloupností aplikovaných pravidel. Jednalo se především o tyto modely (v závorkách jsou uvedeny označení definovaných tříd jazyků):

- bezkontextové #-přepisující systémy ($\mathcal{L}(\mathbf{CF\#RS})$), n-pravě-lineární #-přepisující systémy ($\mathcal{L}(n\text{-}\mathbf{RLIN\#RS})$) a zobecněné #-přepisující systémy ($\mathcal{L}(\mathbf{G\#RS})$)
- hluboké ($\mathcal{L}(\mathbf{DTDP})$) a redukující hluboké zásobníkové automaty ($\mathcal{L}(\mathbf{RDPDA})$)
- zásobníkové automaty s omezeným obsahem zásobníku ($\mathcal{L}(\mathbf{RCPDA}, X)$)

Tyto formální modely jsme omezovali především pomocí následujících typů omezování, které byly klasifikovány v kapitole 3 (v závorkách je styl značení tříd jazyků):

- konečný index k ($\mathcal{L}_k(\mathbf{X})$)

- k-limitovanost ($_k\mathcal{L}(\mathbf{X})$)
- hloubka k ($_k\mathcal{L}(\mathbf{X})$)
- omezování konfigurací obecným omezujícím jazykem

Vztah dynamické složitosti a omezování většinou dává vzniknout nekonečným hierarchiím tříd jazyků, což bylo ověřeno hned na několika různých typech přepisujících systémů (především těch řízených nebo omezovaných).

Nejdůležitější výsledky této práce (kapitole 5) charakterizující nekonečné hierarchie tříd jazyků založené na omezování různých typů přepisujících systémů na základě jistého parametru $k \geq 1$ jsou

$$_k\mathcal{L}(\mathbf{ST}) = {}_k\mathcal{L}(\mathbf{DTDP}) = {}_k\mathcal{L}(\mathbf{RDPDA})$$

$$\mathcal{L}_k(\mathbf{P}) = \mathcal{L}_k(\mathbf{RC}) = \mathcal{L}_k(\mathbf{CF\#RS}) = \mathcal{L}_k(\mathbf{det_1CF\#RS}) = \mathcal{L}_k(\mathbf{G\#RS})$$

$$\mathcal{L}(k\text{-}\mathbf{RLIN\#RS}) = \mathcal{R}^1_{[k]}$$

Studium omezování, potažmo řízení formálních modelů je součástí teorie formálních jazyků již několik desítek let, ale pořád lze v této oblasti nalézat teoretické, ale i praktické výzvy (např. [24, 25, 26, 65]). Poslední část (kapitola 6) je zaměřena na vlastnosti blízké praktickému využití představených #-přepisujících systémů, které tvoří jádro celého textu. Studovali jsme jak možné oblasti využití, tak vlastnosti jako:

- různé definice determinismu (2 typy)
- dva typy kanonického přepisování a jejich vazbu na limitovanost

Ke skutečné praktické využitelnosti však zbývá nejen propracovanější modifikace některých systémů a algoritmů na jejich zpracování, ale především dobrá aplikační oblast, která je v případě #-přepisujících systémů zatím nejasná. Na druhou stranu například obecnější hluboké zásobníkové automaty mohou mnoho představených konceptů převzít a jejich aplikovatelnost, například v syntaktické analýze některých nebezkontextových jazyků, má mnohem jasnější obrysy.

7.2 Přínos

Nyní stručně shrňme přínosy této práce:

1. studium kombinovaných přepisujících systémů a zavedení sjednocujících pojmů, které usnadňují a zobecňují jejich studium (snaha o unifikovaný přístup);
2. klasifikace omezování formálních modelů a především omezování konfigurací přepisujících systémů, včetně několika nových výsledků s tímto přístupem spjatých (nekonečné hierarchie tříd jazyků);
3. představení nového řízeného generativního přepisujícího systému — #-přepisujícího systému, který má několik zajímavých, až netradičních vlastností a tvoří časopisecky opublikované jádro této práce.

7.3 Budoucí vývoj

Jak už to ve světě vědy bývá, tak zdaleka ne všechny otázky bývají uspokojivě zodpovězeny, a proto i tato práce není výjimkou a v úplném závěru diskutuje řadu zajímavých otázek a oblastí k budoucímu výzkumu.

7.3.1 Rozpracované hypotézy a otevřené problémy

Kromě studia modifikací přepisujících systémů a otevřených problémů zmíněných v samotném textu se autor plánuje věnovat také některým z následujících námětů, hypotéz či otevřených problémů.

#-přepisující systémy

V celé práci jsme studovali přepisující systémy s různými omezeními. Přirozenou otázkou však je, jaká bude síla těchto modelů v případech, kdy dané omezení implicitně neplyne ze samotné definice systému (např. omezení na konečný index u všech typů #-přepisujících systémů). Tuto otázku jsme si již položili v otevřeném problému 5.2 a jejímu studiu bude nutné věnovat v budoucnosti více pozornosti.

Otevřený problém 7.1. ${}_n\mathcal{L}(\mathbf{ST}) \subseteq \mathcal{L}(\mathbf{G\#RS}) \subseteq \mathbf{CS}$?

Otevřený problém 7.2. $\mathbf{CF} \subseteq \mathcal{L}(\mathbf{CF\#RS})$?

Jen v několika bodech uveďme náměty na modifikace, rozšíření a další zkoumání #-přepisujících systémů:

- varianty nastíněné v podsekci 4.1.5: redukující a paralelní #-přepisující systémy;
- zobecnění #-přepisujících systémů směrem ke gramatikám, kdy kromě jediného symbolu hranice (#) opět zavedeme konečnou množinu neterminálních symbolů;
- další nekonečné hierarchie tříd jazyků definovaných bezkontextovými #-přepisujícími systémy v závislosti na kardinalitě množiny stavů, což je hierarchie založená nikoli na dynamické, ale spíše popisné složitosti.

Hluboké zásobníkové automaty

Prvním, čeho si pozorný čtenář všimne, je podobnost mezi #-přepisujícími systémy (bez omezení na konečný index) a hlubokými zásobníkovými automaty. Oba systémy implicitně omezují zleva počet proměnných, které mohou přepisovat, oba obsahují konečně-stavové řízení. Na tomto místě vyvstává tedy otázka porovnání jejich mocnosti.

Pro případné využití představených přepisujících systémů by bylo nutné hlouběji rozpracovat problém determinismu. Správným směrem by mohla být inspirace u LL(k) gramatik a determinismu typu 2 u #-přepisujících systémů. Oba přístupy by bylo velmi zajímavé aplikovat na hluboké zásobníkové automaty.

Další možná modifikace tkví ve zobecnění n-limitovanosti na n-limitovanost závislou na konkrétním symbolu A, tzv. (n, A)-limitovanost (viz modifikace symbolově-závislý RDPDA v sekci 5.1.5).

Zásobníkové automaty s omezeným obsahem zásobníku

RCPDA tvoří komplementární pojem k řízeným zásobníkovým automatům ([23]). Místo omezování posloupnosti aplikovaných pravidel pomocí omezujícího jazyka aplikuje RCPDA omezování na konfigurace, respektive obsah zásobníku. Lze pro RCPDA dokázat podobný výsledek jako pro automaty řízené lineárními jazyky?

Otevřený problém 7.3. $\mathcal{L}(\mathbf{RCPDA}, \mathbf{LIN}) = \mathbf{RE}$?

Nastiňme dvě možnosti, jak se pokusit tuto rovnost dokázat: (1) pokusit se RCPDA omezovaný lineárním jazykem převést na řízený zásobníkový automat řízený lineárním jazykem ([23]), který právě třídu všech rekurzivně spočetných jazyků (**RE**) charakterizuje nebo (2) využít frontové gramatiky ([22]).

Literatura

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*, 2nd ed. Addison Wesley, 2006.

[2] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling, Volume I: Parsing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[3] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling, Volume II: Compiling*. Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[4] A. V. Aho and J. D. Ullman. *Principles of Compiler Design*. Addison-Wesley, Reading, Massachusetts, 1977.

[5] R. Bidlo and P. Blatný. How to generate recursively enumerable languages using only context-free productions and eight nonterminals. In *Proceedings of 11th Conference and Competition Student EEICT 2005, Volume 3*. Faculty of Electrical Engineering and Communication BUT, 2005, pp. 536–541.

[6] H. Borodihn and H. Fernau. Accepting grammars and systems: an overview. In *Proceedings of Development in Language Theory Conf.*, vol. 53. Magdeburg, 1995, pp. 199–208.

[7] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. of 16th International Conference on Computer Aided Verification—CAV'04*, vol. 3114 of LNCS. Springer-Verlag, 2004, pp. 197–202.

[8] J. G. Brookshear. *Theory of Computation*. Benjamin/Cummings, Redwood City, California, 1989.

[9] M. Chytil. *Automaty a gramatiky*. SNTL, Praha, 1984.

[10] A. Cremers, H. A. Maurer, and O. Mayer. A note on leftmost restricted random context grammars. *Information Processing Letters*, 2:31–33, 1973.

[11] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and Gh. Păun. *Grammar systems. A grammatical approach to distribution and cooperation. Topics in Computer Mathematics 8.* Gordon and Breach Science Publishers, Yverdon, 1994.

[12] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*. Akademie-Verlag, Berlin, 1989.

[13] M. Češka. *Gramatiky a jazyky*. VUT FEI Brno, 1992.

[14] H. Fernau. Graph-controlled grammars as language acceptors. *Journal Automata, Languages and Combinatorics*, 2(2):79–91, 1997.

[15] H. Fernau. Regulated grammars under leftmost derivation. *Grammars*, pp. 37–62, 2000.

[16] H. J. Hoogeboom. *Coordinated Pair Systems*. Universiteit Leiden, 1987.

[17] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[18] O. H. Ibarra. Simple matrix languages. *Information and Control*, 17:359–394, 1970.

[19] P. M. Lewis II, D. J. Rosenkrantz, and R. E. Stearns. *Compiler Design Theory*. Addison-Wesley, Reading, Massachusetts, 1976.

[20] L. Kari. *On insertion and deletion in formal languages*. Turku, Finland, 1991.

[21] T. Kasai. A hierarchy between context-free and context-sensitive languages. *Journal of Computer and System Sciences*, 4:492–508, 1970.

[22] H. C. M. Kleijn and G. Rozenberg. On the generative power of regular pattern grammars. *Acta Informatica*, 20:391–411, 1983.

[23] D. Kolář and A. Meduna. Regulated pushdown automata. *Acta Cybernetica*, 4:653–664, 2000.

[24] D. Kolář. *Pushdown Automata: Another Extensions and Transformations*. FIT VUT, Brno, CZ, 2005.

[25] D. Kolář and A. Meduna. Regulated automata: From theory towards applications. In *Proceeding of 8th International Conference on Information Systems Implementation and Modelling ISIM'05*. MARQ, Ostrava, CZ, 2005, pp. 33–48.

[26] M. Kot. *Řízené gramatiky [Diplomová práce]*. VŠB – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky, 2002.

[27] Z. Křivka. Dvoucestné k-lineární n-komponentní gramatické systémy. In *Proceedings of the 10th Conference and Competition STUDENT EEICT 2004 Volume 1*. Faculty of Electrical Engineering and Communication BUT, 2004, pp. 203–205.

[28] Z. Křivka. Zásobníkové automaty s omezeným obsahem zásobníku. In *ACM STUDENT CZ*, 2004, pp. 8.

[29] Z. Křivka. Recursive erasing in programmed grammars. In *Pre-Proceedings MEMICS 2005*, 2005, pp. 139–144.

[30] Z. Křivka. String-partitioning systems. In *Proceedings of 11th Conference and Competition STUDENT EEICT 2005 Volume 3*. Faculty of Electrical Engineering and Communication BUT, 2005, pp. 556–560.

[31] Z. Křivka. String-partitioning systems. In *Proceedings of International Interdisciplinary HONEYWELL EMI 2005*. Faculty of Electrical Engineering and Communication BUT, 2005, pp. 217–221.

[32] Z. Křivka and A. Meduna. Random context and programmed grammars of finite index have the same generative power. In *Proceedings of 8th International Conference ISIM'05 Information Systems Implementation and Modelling*, 1st edition, 2005, pp. 67–72.

[33] Z. Křivka and A. Meduna. General top-down parsers based on deep pushdown expansions. In *Proceedings of 1st International Workshop on Formal Models (WFM'06)*, 2006, pp. 11–18.

[34] Z. Křivka, A. Meduna, and R. Schönecker. Generation of languages by rewriting systems that resemble automata. *International Journal of Foundations of Computer Science*, 17(5):1223–1229, 2006.

[35] Z. Křivka, A. Meduna, and R. Schönecker. Reducing deep pushdown automata and infinite hierarchy. In *MEMICS 2006 Second Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, 2006, pp. 214–221.

[36] Z. Křivka and R. Schönecker. Reducing deep pushdown automata. In *Proceedings of the 12th Conference and Competition STUDENT EEICT 2006 Volume 4*, 2006, pp. 365–369.

[37] Z. Křivka and R. Schönecker. String-partitioning systems and an infinite hierarchy. In *Proceedings of 1st International Workshop on Formal Models (WFM'06)*, 2006, pp. 53–60.

[38] R. C. Linger, H. D. Mills, and B. I. Witt. *Structured Programming: Theory and Practice*. Addison-Wesley, Reading, Massachusetts, 1979.

[39] P. Linz. *An Introduction to Formal Languages and Automata*. D.C. Heath and Co., Mass, Lexigton, 1990.

[40] L. Lorenc and A. Meduna. Self-reproducing pushdown transducers. *Kybernetika*, 41(4):531–537, 2005.

[41] T. Masopust and A. Meduna. Descriptional complexity of grammars regulated by context conditions. In *LATA 2007 Pre-proceedings*. Tarragona, ES, 2007, pp. 403–411.

[42] H. A. Maurer. Simple matrix languages with a leftmost restriction. *Information and Control*, 23:128—-139, 1973.

[43] A. Meduna. Syntactic complexity of context-free grammars over word monoids. *Acta Informatica*, 33:457–462, 1996.

[44] A. Meduna. Four-nonterminal scattered context grammars characterize the family of recursively enumerable languages. *International Journal of Computer Mathematics*, 63:67–83, 1997.

[45] A. Meduna. On the number of nonterminals in matrix grammars with leftmost derivations. *LNCS*, 1217:27–38, 1997.

[46] A. Meduna. Descriptional complexity of multi-continuous grammars. *Acta Cybernetica*, 13:375–384, 1998.

[47] A. Meduna. Prefix pushdown automata and their simplification. *International Journal of Computer Mathematics*, 71(1):1–20, 1999.

[48] A. Meduna. *Automata and Languages: Theory and Applications*. Springer, London, GB, 2000.

[49] A. Meduna. *Moderní Teoretická Informatika [materiály k přednáškám]*. FIT VUT, Brno, CZ, 2004.

[50] A. Meduna. Deep pushdown automata. *Acta Informatica*, 2006(98):114–124, 2006.

[51] A. Meduna and H. Fernau. A simultaneous reduction of several measures of descriptional complexity in scattered context grammars. *Information Processing Letters*, 86:235–240, 2003.

[52] A. Meduna and D. Kolář. One-turn regulated pushdown automata and their reduction. *Fundamenta Informaticae*, 16:399–405, 2002.

[53] A. Meduna and M. Švec. *Grammars with Context Conditions and Their Applications*. John Wiley & Sons, Hoboken, New Jersey, USA, 2005.

[54] A. Meduna and M. Vítek. New language operations in formal language theory. *Schedae Informaticae*, 2004(13):123–150, 2004.

[55] E. Moriya, D. Hofbauer, M. Huber, and F. Otto. On state-alternating context-free grammars. *Theoretical Computer Science*, 337:183–216, 2005.

[56] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Mass, 1994.

[57] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[58] Gh. Păun. An infinite hierarchy of matrix languages. *Stud. Cerc. Mat.*, 32:697–707, 1980.

[59] Gh. Păun. On leftmost derivation restriction in regulated rewriting. *Rev. Roumaine Math. Pures Appl.*, 30:751—758, 1985.

[60] R. D. Rosebrugh and D. Wood. A characterization theorem for n-parallel right linear languages. *Journal of Computer and System Sciences*, 7:579–582, 1973.

[61] R. D. Rosebrugh and D. Wood. Restricted parallelism and right linear grammars. *Utilitas Mathematica*, 7:151–186, 1975.

[62] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16:107–131, 1969.

[63] G. Rozenberg and P. Doucet. On 0l-languages. *Information and Control*, 19:302–318, 1971.

[64] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages*, vols. 1–3. Springer, Berlin, 1997.

[65] L. Rychnovský. Parsing of context-sensitive languages. In *Information Systems and Formal Models (Proceedings of 2nd International Workshop on Formal Models (WFM'07))*. Silesian University, 2007.

[66] A. Salomaa. *Theory of Automata*. Pergamon Press, London, 1969.

[67] A. Salomaa. *Formal Languages*. Academic Press, New York, 1973.

[68] A. Salomaa. *Computation and Automata*. Cambridge University Press, Cambridge, England, 1985.

[69] S. Sippu and E. Soisalon-Soininen. *Parsing Theory*. Springer-Verlag, New York, 1987.

[70] S. Sippu and E. Soisalon-Soininen. *Parsing Theory II: LR(k) and LL(k) Parsing (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 1990.

[71] J. Šlapal. *Metody diskrétní matematiky*. VUT FSI Brno, Technická 2, Brno, 2001.

[72] S. H. von Solms. Modelling the growth of simple biological organisms using formal language theory. *Manuscript*.

[73] T. A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*, 2nd ed. Addison-Wesley, Reading, Massachusetts, USA, 1996.

[74] A. P. J. van der Walt. Random context grammars. In *Proceedings of the Symposium on Formal Languages*. Oberwolfach, 1970.

[75] P. Wegner. Programming language semantics. In R. Rustin, ed., *Formal Semantics of Programming Languages*. Prentice-Hall, Englewood Cliffs, New Jersey, 1972, pp. 149–248.

[76] M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company, Inc., 1996.

[77] D. Wood. m-parallel n-right linear simple matrix languages. *Utilitas Mathematica*, 8:3–28, 1975.

[78] D. Wood. Iterated a-ngsm maps and γ systems. *Information and Control*, 32(1), 1976.

[79] D. Wood. *Paradigms and Programming with PASCAL*. Computer Science Press, Rockville, Maryland, 1984.

Seznam symbolů a zkratek

$\square$	konec důkazu lemmatu
$\blacksquare$	konec důkazu věty nebo důsledku
$\emptyset$	prázdná množina
$\cup$	sjednocení množin
$\cap$	průnik množin
$-$	operace odčítání; záporné znaménko
$\overline{X}$	doplněk množiny X vzhledem k danému univerzu
2^X	potenční množina množiny X
$\times$	kartézský součin
$\in$	náležející; je prvkem množiny
$\notin$	nenáležející; není prvkem množiny
$\subset$	je vlastní podmnožinou
$\subseteq$	je podmnožinou
$\Rightarrow$	symbol relace přímého přepisu; výpočetního kroku; derivace
$\Rightarrow^*$	reflexivní a tranzitivní uzávěr relace $\Rightarrow$; přepisuje
$\Rightarrow^+$	tranzitivní uzávěr relace $\Rightarrow$; netriviálně přepisuje
$\Rightarrow^n$	$n-$násobný součin (n-tá mocnina) relace $\Rightarrow$, $n \geq 0$; přepisuje v n krocích
$\rightarrow$	symbol přepsání; přepiš
$\vdash$	symbol přepsání; redukuj
$\#$	symbol hranice; dno zásobníku
$=$	symbol rovnosti; je rovno
$\neq$	symbol nerovnosti; není rovno; různé od
$\leq$	menší nebo rovno; relace uspořádání
$<$	menší; relace ostrého uspořádání
$\geq$	větší nebo rovno
$\cdot$	násobení (možno vynechat)
$\mid$	oddělovač v množinovém konstruktu; svislá závorka
$\lvert x \rvert$	délka posloupnosti/řetězce x
$\lfloor, \rfloor$	závorky ohraničující pravidlo (pro lepší oddělení od textu)
:	oddělovač pro návěští a označovaný objekt
$\langle, \rangle$	úhlové závorky; ohraničení víceznakových symbolů nebo stavů
*	symbol volného monoidu, není-li v kontextu uvedeno jinak; iterace
$^+$	symbol volné semigrupy, není-li v kontextu uvedeno jinak; pozitivní iterace
$\triangleright, \triangleleft$	pomocné symboly; směr průchodu řetězce
$\triangle$, \$	zástupné a pomocné symboly
$\mathbb{N}$	množina kladných celých čísel
$\mathbb{N}_0$	množina nezáporných celých čísel (včetně nuly)

α	řecké písmeno *alfa*
β	řecké písmeno *béta*
Γ, γ	řecké písmeno *gama*
Δ, δ	řecké písmeno *delta*
ε	řecké písmeno *epsilon*; symbol prázdného řetězce
η	řecké písmeno *éta*
μ	řecké písmeno *mý*
Ξ, ξ	řecké písmeno *ksí*
π	řecké písmeno *pí*
ρ	řecké písmeno *ró*
Σ, σ	řecké písmeno *sigma*; suma
τ	řecké písmeno *tau*
χ	řecké písmeno *chí*
Ω, ω	řecké písmeno *omega*

Značení tříd jazyků

CF	třída bezkontextových jazyků
CS	třída kontextových jazyků
$\mathcal{L}(\mathbf{CF\#RS})$	třída jazyků generovaných bezkontextovými #-přepisujícími systémy
$\mathcal{L}(\mathbf{det_T CF\#RS})$	třída jazyků definovaných deterministickými bezkontextovými #-přepisujícími systémy typu T
$\mathcal{L}(\mathbf{DTDP})$	třída jazyků přijímaných hlubokými zásobníkovými automaty
$\mathcal{L}(\mathbf{FA})$	třída jazyků (dále jen tř. j.) přijímaných konečnými automaty
$\mathcal{L}(\mathbf{G\#RS})$	třída jazyků generovaných zobecněnými #-přepisujícími systémy
$\mathcal{L}(n\text{-}\mathbf{RLIN\#RS})$	třída jazyků generovaných n-pravě-lineárními #-přepisujícími systémy
$\mathcal{L}(\mathbf{P})$	třída jazyků generovaných programovanými gramatikami
$\mathcal{L}(\mathbf{PDA})$	třída jazyků přijímaných zásobníkovými automaty (dále jen zás. aut.)
$\mathcal{L}(\mathbf{RC})$	třída jazyků generovaných gramatikami s rozptýleným kontextem
$\mathcal{L}(\mathbf{ST})$	třída jazyků generovaných stavovými gramatikami
$\mathcal{L}(\mathbf{RCPDA}, X)$	tř. j. přijímaných zás. aut. s omezeným obsahem zásobníku jazykem z X
$\mathcal{L}(\mathbf{RDPDA})$	tř. j. definovaných zprava-doleva redukujícími hlubokými zás. aut.
$\mathcal{L}(^{\mathbf{lr}}\mathbf{RDPDA})$	tř. j. definovaných zleva-doprava redukujícími hlubokými zás. aut.
${}_n\mathcal{L}(\mathbf{X})$	třída jazyků definovaných n-limitovaným modelem X; třída jazyků definovaných modelem X hloubky nejvýše n
$\mathcal{L}_k(\mathbf{X})$	třída jazyků definovaných modelem X indexu k
$\mathcal{L}(\mathbf{X}, ac)$	třída jazyků definovaných modelem X s kontrolou výskytu
$\mathcal{L}(\mathbf{X}, CF - \varepsilon)$	třída jazyků definovaných modelem X bez vymazávajících pravidel
LIN	třída lineárních jazyků
$\mathcal{R}^m_{[n]}$	třída jazyků generovaná m-paralelními n-pravě-lineárními jednoduchými maticovými gramatikami
RE	třída rekurzivně spočetných jazyků; třída jazyků generovaných neomezenými gramatikami
REG	třída regulárních jazyků