SLEZSKÁ UNIVERZITA
FILOZOFICKO-PŘÍRODOVĚDECKÁ FAKULTA V OPAVĚ

Šárka Vavrečková

Skripta do předmětů

Teorie jazyků a automatů

I Základy teoretické informatiky

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě

Opava
30. dubna 2018

Anotace: Tato skripta jsou určena pro studenty předmětů *Teorie jazyků a automatů I* (obory Informatika a výpočetní technika, Informatika dvouoborové) a *Základy teoretické informatiky I* (obor Aplikovaná informatika). Cílem je seznámit studenty s teoretickou informatikou, jejími prostředky a metodami.

Teorie jazyků a automatů I, Základy teoretické informatiky I

RNDr. Šárka Vavrečková, Ph.D.

Dostupné na: http://vavreckova.zam.slu.cz/formal1.html

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě
Bezručovo nám. 13, Opava

Sázeno v systému LaTeX

Předmluva

Co najdeme v těchto skriptech

Tato skripta jsou určena pro studenty informatických oborů na Ústavu informatiky Slezské univerzity v Opavě. Obsahují látku vyučovanou v předmětech *Teorie jazyků a automatů I* a *Základy teoretické informatiky I*, ve kterých se zabýváme především prostředky a metodami teoretické informatiky.

Pro většinu studentů je toto téma zcela nové, proto předem upozorňuji: budete potřebovat základy matematiky, především z teorie množin, a především schopnost logického úsudku. Pokud tyto dvě oblasti ovládáte, určitě zvládnete i zde probíraná témata.

Mohlo by se zdát, že se budeme zabývat jen nezáživnou teorií, která nemá s praxí nic společného. Jenže to není pravda – vše, co budeme probírat, je inspirováno praxí či přímo přírodou, a vše je také v praxi použitelné, jen to nemusí být na první pohled patrné. Například na bakalářském stupni studia je předmět, ve kterém si uplatnění teoretické informatiky studenti vyzkoušejí na vlastní kůži – *Překladače*.

Některé oblasti jsou „navíc“ (jsou označeny ikonami fialové barvy), ty nejsou probírány a ani se neobjeví na zkoušce – jejich úkolem je motivovat k dalšímu samostatnému studiu nebo pomáhat v budoucnu při získávání dalších informací. Pokud je fialová ikona před názvem kapitoly (sekce), platí pro vše, co se v dané kapitole či sekci nachází.

Značení

Ve skriptech se používají následující barevné ikony:

-  Nové *pojmy*, vysvětlení významu některých postupů a značení, používané symboly, postupy, nástroje, apod. jsou značeny modrým symbolem, který vidíme také zde vpravo.
-  Některé části textu jsou označeny fialovou ikonou, což znamená, že jde o *nepovinné úseky*, které nejsou probírány (většinou; studenti si je mohou podle zájmu vyžádat nebo sami prostudovat). Jejich účelem je dobrovolné rozšíření znalostí studentů o pokročilá témata, na která obvykle při výuce nezbývá moc času.

-  Žlutou ikonou jsou označeny odkazy, na kterých lze získat *další informace* o tématu. Nejčastěji u této ikony najdeme webové odkazy na stránky, kde se dané tématice jejich autoři věnují podrobněji.
- Červená  je ikona pro *upozornění* a poznámky.

Každý semestr se probere odlišné množství látky, proto se při rozlišování „povinného“ a „nepovinného“ řiďte především seznamem otázek, který taktéž najdete na mém webu.

Opticky jsou odlišeny také definice, věty, řešené příklady a neřešené úlohy. Definice, věty a příklady jsou číslovány, čísla slouží k jednoduchému odkazování v textu.

Definice 0.1

Definice stanovují význam určitého *(definovaného)* pojmu. *Jejich plné znění je důležité – definici sice nemusíme nutně umět slovo od slova zpaměti, nicméně nesmíme na žádnou část definice zapomenout, musíme znát všechny části, a to formálně správně. S definicí obvykle souvisí určité* formální značení*, které je závazné.*

Věta 0.1

Věta (na rozdíl od definice) stanoví určitý vztah*, obvykle jde o vztah založený na implikaci nebo ekvivalenci. Každá věta by správně měla být dokázána, zde však budeme uvádět důkazy jen u některých vět, u jiných jen myšlenku důkazu.*

Příklad 0.1

Takto vypadá prostředí s příkladem, například nějakého postupu. Příklady jsou obvykle komentovány, aby byl jasný postup jejich řešení.

Úkol

Otázky a úkoly, náměty na vyzkoušení, které se doporučuje při procvičování učiva provádět, jsou uzavřeny v tomto prostředí. Pokud je v prostředí více úkolů, jsou číslovány.

Obsah

Kapitola 1

Teoretická informatika

Tato kapitola je především motivační – dovíme se zde, jaký má teoretická informatika význam, jak vznikla a také jaké jsou možnosti jejího praktického použití.

1.1 Vznik a vývoj teoretické informatiky

Základy teoretické informatiky byly položeny již v několika předchozích stoletích. Dá se říct, že vycházíme z těchto tří kořenů:

1. matematika,
2. jazykověda,
3. biologie.

1.1.1 Matematika

Počátek 20. století se vyznačoval především rozvojem logiky a logického usuzování. Logika existovala a byla používána již v antice, ovšem ve 20. století pozorujeme snahu aplikovat logiku do prakticky všech existujících vědních oborů. Postupně se přišlo na to, že ne vše je vypočitatelné (dokazatelné, matematicky odvoditelné), tedy vyvstala otázka: *Co lze vypočítat, matematicky odvodit, dokázat?*

Na tuto otázku dokázal odpovědět *Alan Turing (1912–1954)*, který *za svých studií* na King's College v Cambridgi publikoval roku 1937 článek *On Computable Numbers*, ve kterém představil koncept univerzálního matematického modelu, který při správném „naprogramování“ dokáže vypočítat právě to (vše), co je vypočítatelné. Pro tento model se později vžil název Turingův stroj.

Turingův stroj se skládá z *řídicí jednotky* (obdoba procesoru), která má k dispozici tzv. *stav* (obdobu vnitřní proměnné či registru procesoru), z *pásky* (obdoba operační paměti, do které je před začátkem výpočtu zapsán vstupní řetězec) a *čtecí a zápisové hlavy*, která pracuje s páskou a může se na ní libovolně pohybovat (vždy o jeden krok). Schéma je na obrázku 1.1.

Obrázek 1.1: Turingův stroj

Vlastnosti Turingova stroje:

- řídicí jednotka se vždy nachází v některém z předem určených stavů,
- čtecí a zápisová hlava ukazuje vždy na jedno pole pásky,
- v každém kroku se stroj řídí podle stavu jednotky a podle obsahu pole, na které ukazuje hlava,
- podle těchto dvou údajů (stav a obsah pole) se rozhodne o akci, která spočívá
  - ve změně stavu jednotky (například ze stavu „načítání“ do stavu „načteno“ nebo ze stavu „pracuji“ do stavu „vypínám“),
  - přepsání obsahu pole na pásce něčím jiným (nebo může pole zůstat beze změny), a
  - posunu na pásce vpravo, vlevo, anebo může hlava zůstat na stejném poli.

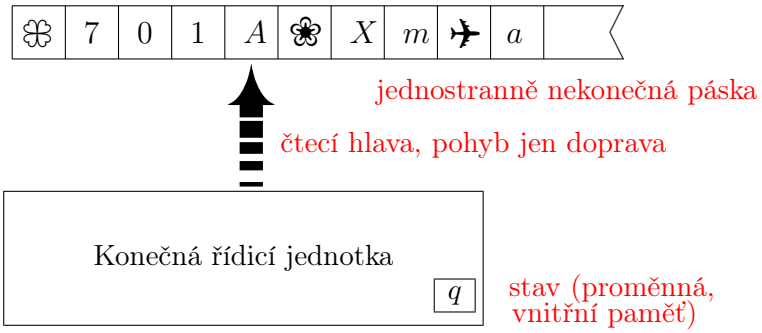
Výsledkem činnosti stroje obvykle bývá obsah pásky, důležitou informací může být stav, ve kterém stroj skončil výpočet (je množina koncových stavů).

Alan Turing byl především matematik, ale záběr jeho znalostí, zkušeností, článků a praktických dovedností je velmi široký. Například za druhé světové války se podílel na rozlušení kódu Enigmy, čímž přispěl k úspěchům ve válce proti fašistickému Německu, také se zabýval velkými poválečnými počítači a umělou inteligencí (Turingův test). To, že Turingův stroj není jen suchá teorie, dokázali studenti ve Francii, když ho sestrojili z kostiček Lega (video najdete na odkazu [5] v seznamu literatury).

Další modely: Turingův stroj byl pro některé jednodušší výpočty zbytečně složitý, proto vznikly jednodušší modely – konečný automat a zásobníkový automat. Tyto automaty již nejsou „univerzální“, nedokážou vypočíst vše vypočitatelné, ale jejich používání je mnohem jednodušší, dokážou pracovat efektivněji a pro stanovené úkoly zcela dostačují (kromě jiného se dají snadno naprogramovat).

Nejdřív si představíme *konečný automat*. Ze základních modelů, kterými se budeme zabývat, je nejjednodušší. Umí pouze zpracovávat svůj vstup, nemění ho (neumí zapisovat), výstupem je konkrétní stav. Nicméně není problém tento koncept rozšířit a umožnit automatu i konkrétní činnost závislou na stavu, ve kterém se nachází. Schéma vidíme na obrázku 1.2. Je založen na velmi jednoduchém principu:

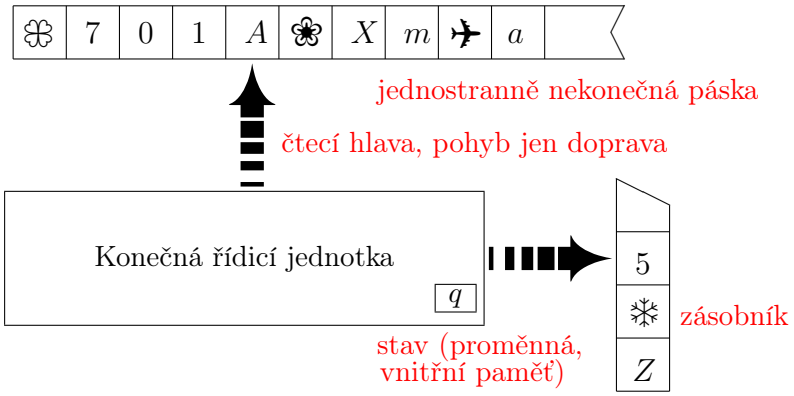
- řídicí jednotka se vždy nachází v některém z předem určených stavů,
- čtecí hlava ukazuje vždy na některé pole pásky,



Obrázek 1.2: Konečný automat

- v každém kroku se stroj řídí podle stavu jednotky a podle obsahu pole, na které ukazuje hlava,
- podle těchto dvou údajů se rozhodne o akci, která spočívá
  - ve změně stavu jednotky,
  - v posunu na pásce o pole vpravo.

Na rozdíl od Turingova stroje se čtecí hlava posouvá v každém kroku o jedno pole doprava a nemá možnost zapisovat. Výsledkem činnosti automatu je pouze informace o tom, ve kterém stavu stroj skončil, a zda přečetl celý obsah pásky (nepřečetl a „zasekl se“ – to nastane, když pro daný obsah pole na pásce a momentální stav není definována žádná akce).



Obrázek 1.3: Zásobníkový automat

Zásobníkový automat toho již umí víc (říkáme, že je „silnější“), svými schopnostmi je někde mezi konečným automatem a Turingovým strojem. Můžeme si ho představit jako konečný automat, kterému jsme přidali další pásku – zásobník (představme si dodatečnou paměť na pomocné výpočty). Vlastnosti zásobníkového automatu:

- řídicí jednotka se vždy nachází v některém z předem určených stavů,
- čtecí hlava je vždy na některém poli pásky,
- automat má k dispozici zásobník, kde přidává shora a také shora vybírá,
- v každém kroku automat vyjme jeden prvek ze zásobníku, a řídí se podle tohoto vyjmutého symbolu, stavu jednotky a také se může (nemusí) řídit podle obsahu pole, na které ukazuje hlava,

- podle těchto tří (nebo dvou) údajů se rozhodne o akci, která spočívá
  - ve změně stavu jednotky,
  - posunu na pásce o jedno pole vpravo, pokud v tomto kroku četl symbol ze vstupní pásky (tj. když čte ze vstupu, zároveň se posune dál),
  - může uložit do zásobníku jakýkoliv počet prvků (i žádný), a to postupně po jednom, ten, který vložil jako poslední, bude hned v dalším kroku vyjmut (tj. je na vrcholu zásobníku).

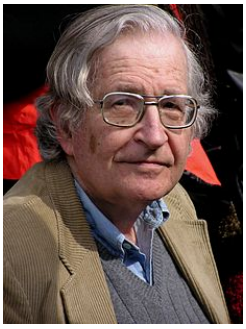
Na rozdíl od konečného automatu čtecí hlava nemusí pracovat v každém kroku (buď čte a zároveň se posune, nebo nepracuje vůbec), nemá možnost zapisovat a pohybuje se jen směrem doprava. Výsledkem činnosti je informace o stavu, ve kterém stroj skončil, zda přečetl celý obsah pásky, a případně může být důležité, zda do konce výpočtu stačil vyprázdnit celý zásobník.

Poznámka:

Se všemi těmito koncepty se seznámíme během dvou semestrů výuky tohoto předmětu.

1.1.2 Jazykověda

Formální gramatika řeší, jak se slova utvářejí (u matematických kořenů byl řešen problém rozpoznávání již utvořených slov či sekvencí signálů).



Noam Chomsky (1928–) je známý americký lingvista, který ve svém výzkumu zkoumal syntaxi jazyka a schopnost lidí mluvit.

„Všechny jazyky jsou utvořeny přibližně stejným způsobem, mají stejný základ.“

Jeho představou bylo, že syntaxi jakéhokoliv jazyka lze popsat pomocí sady pravidel (která například stanovují místo podmětu, přísudku a dalších větných členů ve větě), tuto sadu pravidel nazval formální gramatikou. Každé dítě má (podle něj) vrozenou znalost jakési univerzální gramatiky společné všem jazykům, která mu pomáhá naučit se používat svůj konkrétní jazyk.

Formální gramatika je soubor obecných pravidel, generujeme větu na základě její struktury (syntaxe). Ukázku jednoduché formální gramatiky vidíme v následujícím příkladu.

Příklad 1.1

Je dána tato formální gramatika:

$S \to \langle \text{begin} \rangle\, T\, \langle \text{end} \rangle$

$T \to P; \mid P; T$

$P \to \langle \text{vypis} \rangle\, \langle str \rangle \mid \langle \text{vypocti} \rangle\, V$

$V \to V + A \mid V - A \mid A$

$A \to A * B \mid A/B \mid B$

$B \to (V) \mid \langle cislo \rangle$

Podle pravidel této gramatiky můžeme provést následující odvození (všimněte si, že symbol na levé straně prvního pravidla jsme použili jako *startovací symbol*, tedy na začátku celého odvozování):

$S \Rightarrow \langle\text{begin}\rangle\, T\, \langle\text{end}\rangle \Rightarrow \langle\text{begin}\rangle\, P; T\, \langle\text{end}\rangle \Rightarrow \langle\text{begin}\rangle\, P; P; \langle\text{end}\rangle \Rightarrow$
$\Rightarrow \langle\text{begin}\rangle\, \langle\text{vypis}\rangle\, \text{"}\mathit{vysledek}: \text{"}; P; \langle\text{end}\rangle \Rightarrow$
$\Rightarrow \langle\text{begin}\rangle\, \langle\text{vypis}\rangle\, \text{"}\mathit{vysledek}: \text{"}; \langle\text{vypocti}\rangle\, V; \langle\text{end}\rangle \Rightarrow \ldots$
$\Rightarrow \langle\text{begin}\rangle\, \langle\text{vypis}\rangle\, \text{"}\mathit{vysledek}: \text{"}; \langle\text{vypocti}\rangle\, 25 + (6 * 92) - 57; \langle\text{end}\rangle$

Po přečtení výsledného řetězce už můžeme odhadnout, jak asi vypadá jazyk generovaný touto gramatikou. Na začátku máme klíčové slovo *begin*, na konci klíčové slovo *end*. Mezi nimi má být posloupnost příkazů oddělených středníkem, jsou definovány dva zjednodušené příkazy – klíčové slovo *vypis* následované posloupností písmen, která zřejmě mají být vypsána, nebo klíčové slovo *vypocti* následované matematickým výrazem, který má být vypočten. Je to jen velmi zjednodušené, určitě by každého napadlo, co by se ještě „dalo přidat“, aby byl takový programovací jazyk použitelný.

V příkladu vidíme velká písmena (například S) a dále určité sekvence, které můžeme považovat za slova (například ⟨begin⟩). Každé pravidlo má levou a pravou stranu, kde na levé straně máme velké písmeno, na pravé straně je řetězec. Pravidlo předepisuje určitou činnost – pokud ve zpracovávaném řetězci objevíme velké písmeno, které je na levé straně některého pravidla (některých pravidel), nahradíme je pravou stranou některého pravidla určeného pro zpracování tohoto velkého písmene (tj. ve směru šipky).

Základní princip je tedy následující: Věta se skládá ze slov, při skládání částí potřebujeme také pomocné symboly („obecné termíny“), za které můžeme dosadit posloupnost skládající se ze symbolů a pomocných symbolů – rekurze.

Pomocné symboly (tedy jakési proměnné) nazýváme *neterminální symboly*, skutečné symboly jazyka pak *terminální symboly* (protože jsou na konci generování).

1.1.3 Biologie

Aristid Lindenmayer (1925–1989) byl maďarský biolog, který se kromě jiného zabýval popisem růstu různých rostlin. Zjistil, že když upraví formální gramatiku a pozmění její chování, dokáže například simulovat růst a vývoj rostliny nebo dělení buněk. Model, který pro tento účel vytvořil, nazýváme *Lindenmayerovy systémy* (L-systémy).

Jeho žáci pak přišli na možnost grafické interpretace L-systémů, čímž vznikají různé typy fraktálů. *Fraktál* (ze slova *fractus*, rozbitý) je geometrický obrazec, který je soběpodobný (tj. část tohoto útvaru bývá podobná útvaru celému, například větvička má podobnou strukturu jako celý strom), přičemž coby velmi složitý obrazec je generován velmi jednoduchými pravidly. L-systémy jsou pouze jednou z několika možností pro generování fraktálů.

U fraktálů generovaných L-Systémy jde o to, jak poměrně složitý nákres vygenerovat co nejjednodušším řetězcem „obyčejných“ symbolů, které mají význam určité instrukce (vykresli čáru,

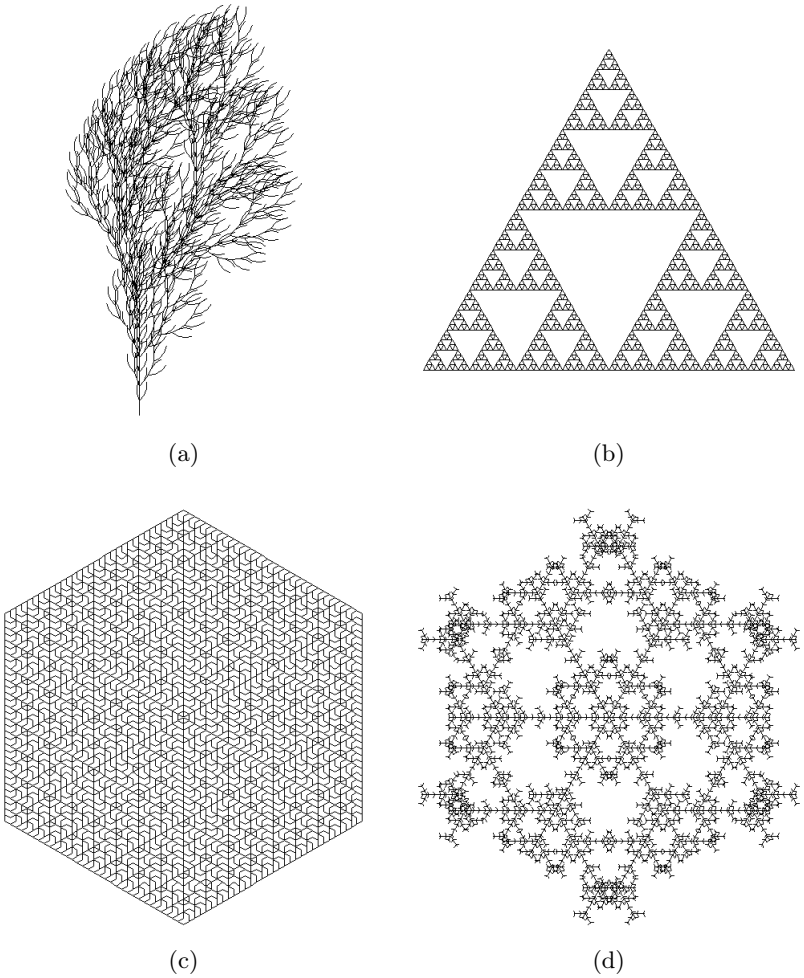
popojdi bez vykreslení, otoč se doprava, doleva, ulož písmeno do zásobníku, vyjmi písmeno ze zásobníku, apod.).

Příklad 1.2

Pravidlo: $b \to bb$

Výpočet: $b \Rightarrow bb \Rightarrow b^4 \Rightarrow b^8 \Rightarrow \ldots$

Uplatňuje se rekurze, tedy totéž pravidlo (nebo tatáž pravidla) se používá pořád dokola tak dlouho, dokud nevznikne dostatečně dlouhá a „složitá“ posloupnost instrukcí.



(a) (b) (c) (d)

Obrázek 1.4: Několik fraktálů vygenerovaných pomocí L-Systémů

Příklad 1.3

Například obrázek 1.4c na straně 6 vznikl z řetězce `F+F+F+F+F+F` tak, že jsme rekurzívně všechny symboly `F` (i ty nově přidávané) zpracovali pravidlem $\texttt{F} \to \texttt{F[+F+F]F}$.

Odvození by tedy vypadalo takto (uvedeme pouze první krok):

`F+F+F+F+F+F` $\Rightarrow$ `F[+F+F]F+F[+F+F]F+F[+F+F]F+F[+F+F]F+F[+F+F]F+F[+F+F]F` $\Rightarrow \ldots$

V každém kroku se řetězec značně prodlouží, protože se pokaždé zpracuje každý symbol, pro který máme pravidlo – to znamená, že délka řetězce roste exponenciálně.

Interpretovat (tj. v tomto případě vykreslit jako obrázek) můžeme výsledek kteréhokoliv kroku odvození. Čím dál odvozujeme, tím složitější obrázek bude. Symbol „`F`“ se interpretuje jako nakreslení čáry v daném směru, symbol „`+`“ je příkaz k pootočení o určitý úhel (může být třeba 60^o). Hranaté závorky určují práci se zásobníkem: levá závorka je příkaz „ulož do zásobníku momentální pozici na plátně a úhel natočení“, pravá závorka znamená „vyjmi ze zásobníku dříve uloženou pozici na plátně a úhel natočení“ – princip zásobníku spočívá v tom, že to, co jsme tam uložili jako poslední, to jako první vyjmeme.

L-systémy mají různé varianty. Existují také tzv. stochastické L-systémy umožňující zanášet do generování princip náhody, také lze použít různé barvy. Na obrázku 1.5 vlevo vidíme záhon rostlin generovaných stochastickým L-systémem, vpravo pak sadu mušlí z jediného L-systému, variace jsou vytvořeny pozměňováním podmínek generování.

Obrázek 1.5: Rostliny a mušle generované stochastickým L-systémem[1]

Příklad 1.4

Obrázek na titulní stránce těchto skript byl vytvořen následovně:

- axiom (počáteční řetězec) je `F-F-F-F`
- jediné pravidlo je `F` → `FF-F+F-F-FF`
- po první iteraci získáme řetězec `F-F-F-F` ⇒ `FF-F+F-F-FF-FF-F+F-F-FF-FF-F+F-F-FF-FF-F+F-F-FF`
- počet iterací je 3, tedy interpretovaný řetězec bychom zapsali na víc než polovinu této stránky
- úhel pro natočení je zvolen na 162^o

Interpretace takto vygenerovaného řetězce by proběhla následovně: symbol `F` znamená „jdi rovně a přitom nakresli čáru“, symbol `+` znamená „natoč se o zadaný úhel (zde 162^o) doleva“, symbol `-` je natočení opačným směrem.

Poté byl obrázek barevně upraven v programu GIMP.

[1] Zdroj: http://www.ccs.neu.edu/course/csg140/abstracts/abstracts2006.html

1.2 Možnosti použití teoretické informatiky

V Úvodu jsme naznačili, že teoretická informatika má také své praktické využití. Nyní si shrňme několik oblastí, kde se teoretická informatika využívá:

- stavové programování, programování překladačů, obecně vyhodnocování řetězců s pevně danou syntaxí (strukturou) včetně například zpracovávání HTML kódu webovým prohlížečem,
- modelování matematických výpočtů, vizualizace,
- analýza přirozeného jazyka (kontrola pravopisu, překlady, atd.),
- L-systémy: biologie, fraktály (malířství, filmy, apod.), fyzika (teorie chaosu, modelování turbulence, studium tornád, atd.), filmový průmysl (modelování obrovského množství objektů, které jsou podobné, třebaže ne zcela stejné – les různých stromů, dav lidí, roj hmyzu, členitý povrch, říční síť, ledové květy na okně, apod.),
- porovnávání složitosti různých algoritmů dělajících totéž (hledáme efektivnější algoritmus pro danou činnost),
- DNA-výpočty,
- fyzika – kvantové počítače,

atd.

Další informace:

- Generátor L-systémů na NolandC: http://nolandc.com/sandbox/fractals/
- Další generátor: http://www.kevs3d.co.uk/dev/lsystems/ – jsou zde i předpřipraveny příklady (na stránce klepněte na některý obrázek vpravo)

Kapitola 2

Jazyky a regulární výrazy

2.1 Možnosti využití regulárních výrazů

Regulární výrazy se v různých podobách využívají v praxi, zejména při vyhledávání (na internetu nebo třeba hledání souboru v počítači), anebo tehdy, když chceme něco provést s množinou objektů (souborů, textu v souborech, v databázích, apod.) a potřebujeme tuto množinu nějak specifikovat.

Příklad 2.1

Ukážeme si několik možností využití regulárních výrazů v praxi.

Vyhledávání na internetu (například Google):

`faktoriál pascal OR C++`
– chceme program pro výpočet faktoriálu v Pascalu nebo C++

`mravenec -Ferda`
– chceme stránky o mravencích, ale ne s Ferdou mravencem

Vyhledávání na počítači (Windows, DOS, unixové systémy včetně Linuxu):

`*.txt`
– všechny soubory s příponou .TXT

`?psa*.*`
– první písmeno jakékoliv, následuje řetězec „psa“, pak cokoliv, přípona také jakákoliv; znamená třeba `opsané.doc`, `upsanec.exe`, `xpsa.xls`, atd.

Vyhledávání na počítači (unixové systémy: `grep`*, Windows:* `findstr`*):*

`[a-z]*[0-9]`
– všechny řetězce obsahující pouze malá písmena (jakýkoliv počet, to zajistí hvězdička) a končící číslicí; hranaté závorky určují množinu povolených možností pro znak, který má být na daném místě

`a[^0-9]*.?`
– vše, co začíná malým „a“, přímo za ním může být jakýkoliv řetězec, který nezačíná číslicí (stříška na začátku množiny dané hranatými závorkami znamená negaci), pak je tečka a za ní ještě jeden znak (otazník představuje jeden jakýkoliv znak)

Vyhledávání v databázích (většinou používáme SQL):

```
SELECT Jmeno, Email FROM Uzivatel WHERE Email LIKE '%@gmail.com'
```

– chceme vypsat jméno a e-mail těch uživatelů (z tabulky `Uzivatel`), kteří mají adresu na Gmailu; v SQL se jako zástupný symbol pro „jakkoliv dlouhý řetězec jakýchkoliv znaků“ používá symbol procenta.

Regulární výrazy využíváme coby běžní uživatelé i tehdy, když si to ani neuvědomujeme, nicméně jako informatici bychom si to určitě uvědomovat měli. Podporu regulárních výrazů dnes máme vestavěnou v mnoha programovacích jazycích (včetně C#, C++, Delphi, Java), některé jazyky mají práci s regulárními výrazy doslova v „popisu práce“ (Perl, je pro Windows i unixové systémy, případně PHP).

Další informace:

- http://interval.cz/clanky/regularni-vyrazy-v-prikladech/ (vysvětlení v příkladech)
- http://www.perl.cz/
- http://www.linuxsoft.cz/article.php?id_article=675 (článek o Perlu na LinuxSoft)
- http://www.regularnivyrazy.info/regexp-programy.html (programy pro práci s regulárními výrazy)
- https://regex101.com/ (tester regulárních výrazů, hodí se, než regulární výraz použijeme v konkrétním programovacím jazyce)
- http://www.regularnivyrazy.info/regexp-tester.html (další tester)

2.2 Matematický základ

2.2.1 Množiny a jazyky

Ať se zaměříme na jakýkoliv teoretický model, vždy budeme pracovat s určitými *prvky* – podle potřeby to budou znaky, písmena, číslice, objekty, signály, symboly, piktogramy, atd.

Základními pojmy pro nás budou množina a posloupnost.

Definice 2.1 (Množina)

Množina *je soubor prvků, který může být konečný i nekonečný (pak hovoříme o konečné nebo nekonečné množině). Na pořadí prvků nezáleží a obvykle je každý prvek v množině pouze jednou (neopakuje se). Pokud umožníme opakování prvků, hovoříme o* multimnožině.

Pokud množinu reprezentujeme výčtem, uzavřeme prvky do složených závorek: $\{\ldots\}$. *Prázdnou množinu obvykle značíme symbolem* $\emptyset$.

Množiny prvků mohou být například tyto: $\{a, b, c\}$, $\{0, 1, \ldots, 9\}$, $\emptyset$, $\mathcal{R}$ (množina reálných čísel), $\{a, aa, aaa, \ldots\}$, atd. První tři uvedené jsou konečné, čtvrtá množina je prázdná, zbývající dvě nekonečné.

Definice 2.2 (Posloupnost, řetězec)

Posloupnost *(sekvence) na rozdíl od množiny určuje i pořadí prvků. Prvky se mohou opakovat, liší se vždy minimálně svým pořadím v posloupnosti.*

Pokud posloupnost reprezentujeme výčtem, uzavřeme ji do kulatých závorek: $(\ldots)$.

Posloupnost znaků nazýváme řetězec *(tyto znaky jsou zřetězeny za sebou), v tomto speciálním případě nepoužíváme závorky ani oddělovače prvků, jen jednotlivé prvky napíšeme za sebe.*

Určitým typem posloupnosti (co se týče počtu prvků) je uspořádaná dvojice, trojice, čtveřice, atd.

Dále budeme pracovat především s množinami slov, kterým budeme říkat *jazyky*. Následují definice související s jazyky:

Definice 2.3 (Abeceda)

Abeceda *je konečná množina prvků (objektů, znaků, symbolů, signálů apod.). Obvykle předpokládáme, že není prázdná. Abecedu často značíme velkým řeckým písmenem Sigma –* Σ.

Definice 2.4 (Slovo, prázdné slovo)

Slovo nad danou abecedou *je konečná posloupnost prvků z této abecedy, řetězec.*

Délka slova *je počet prvků v tomto slově, délku slova* w *zapisujeme* $|w|$. *Slovo o délce* 0 *nazýváme* prázdné slovo *a značíme* ε *(řecké epsilon), tedy platí* $|\varepsilon| = 0$. *Protože je slovo vždy konečné, je délka slova definována (tj. existuje) pro všechna slova.*

Slovo můžeme buď přímo vypsat, nebo je reprezentovat určitým speciálním znakem, obdobou proměnné. V teoretické informatice se pro reprezentaci obecného slova často používá písmeno w (případně s indexem, abychom rozlišili několik odlišných slov), píšeme $w \in \Sigma$ (to znamená: slovo w nad abecedou Σ, tedy slovo skládající se pouze z prvků této abecedy). Dále se totiž budeme zabývat nejen konkrétními slovy, ale především obecnými postupy (algoritmy), které mají platit „pro všechna slova daného jazyka“, a nebylo by technicky možné všechna slova tato jazyka vyjmenovat. Místo toho použijeme „proměnnou“ w.

Definice 2.5 (Jazyk)

Jazyk nad danou abecedou *je množina slov, která se skládají pouze z prvků dané abecedy. Jazyk obvykle značíme písmenem* L.

Počet slov jazyka L *také značíme svislicemi:* $|L|$. *Prázdný jazyk (neobsahující žádná slova) značíme* $\emptyset$, *tedy* $|\emptyset| = 0$.

Jazyk může být i nekonečný, v tom případě počet jeho slov je $\aleph_0$ (aleph 0, odpovídá počtu přirozených čísel, prvků množiny $\mathcal{N}$) nebo $\aleph_1$ (aleph 1, odpovídá počtu reálných čísel, prvků množiny $\mathcal{R}$).

Příklad 2.2

Následuje několik ukázek abeced, slov a jazyků nad nimi utvořených.

- $\Sigma_1 = \{a, b, c\}$ je tříprvková abeceda. Příklady jazyků nad touto abecedou:
  - $L_1 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
    jazyk všech slov nad abecedou Σ_1, která mají délku 2
  - $L_2 = \emptyset$
    prázdný jazyk, ten je nad jakoukoliv abecedou
  - $L_3 = \{\varepsilon\}$
    jazyk obsahující prázdné slovo, ten je taky nad jakoukoliv abecedou
  - $L_4 = \{a\}$, $L_5 = \{ab\}$
    dva jednoprvkové jazyky
  - $L_6 = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$
    nekonečný jazyk všech slov složených z písmene „a“ (všimněte si, že v jazyce najdeme i slovo ε, které je složeno z 0 symbolů a)
- $\Sigma_2 = \{0, 1, \ldots, 9\}$ je abeceda arabských číslic. Příklady jazyků nad touto abecedou:
  - $L_7 = \{0, 1, 2, \ldots, 9, 10, 11, \ldots, 99, 100, 101, 102, \ldots\}$
    jazyk obsahující všechna přirozená čísla, včetně nuly (můžeme označit jako $\mathcal{N}_0$)
  - $L_8 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
    tento jazyk je ve skutečnosti nad abecedou $\{0, 1\}$, obsahuje všechna binární čísla, která lze uložit do tří bitů; počet slov v abecedě je $|L_8| = 2^3 = 8$ (protože používáme dva znaky, které se mohou vyskytovat libovolně na třech různých místech)
  - $L_9 = \emptyset$, $L_{10} = \{\varepsilon\}$
    L_9 je prázdný jazyk (neobsahuje žádná slova), L_{10} je jazyk obsahující prázdné slovo (tj. jednoprvkový jazyk); $|L_9| = 0$, $|L_{10}| = 1$
- $\Sigma_3 = \{\alpha, \beta, \gamma, \delta, \ldots, \omega\}$ – abeceda malých řeckých písmen
- $\Sigma_4 = \{❀, ❁, ❃, ✤, ✺, ✻, ❁\}$ – abeceda zahradníka okrasné zahrady
- $\Sigma_5 = \{\cdot, -, /\}$ – Morseova abeceda

2.2.2 Krácení zápisu slov

Z praktických důvodů se snažíme, aby zápis slov a jazyků byl nejen co nejpřehlednější, ale také úsporný a praktický. Zápis a značení si ukážeme na příkladech.

Příklad 2.3

Protože pracujeme s řetězci, využijeme možnost zkrácení zápis zřetězení shodných prvků:

- $a^2, a^3, a^4, \ldots$
  takto zkracujeme zápis dvou, tří, čtyř, atd. prvků uvedených v základu, je to totéž jako $aa, aaa, aaaa, \ldots$
- $a^0 = b^0 = \ldots = \varepsilon = \varepsilon^0$
  slovo skládající se z nula prvků (jakýchkoliv) je prázdné slovo

- $(ab)^2 = abab$
  pozor, neznamená to „$aabb$“, závorka má přednost i zde
- $(50)^0 = \varepsilon$
  toto platí v případě, že 50 chápeme jako řetězec (slovo), zde se jedná o řetězcovou operaci
- $a^4(01)^3b^2 = aaaa010101bb$
- $(a^2)^3 = a^6$, podle vzorce $(x^m)^n = x^{m \cdot n}$

Definice 2.6 (Operace hvězdička, Kleeneho operátor, iterace)

Operace hvězdička (Kleeneho uzávěr, iterace) určuje nekonečné zřetězení daného prvku. Výsledkem této operace je vždy nekonečná množina.

Příklad 2.4

Iteraci často používáme při zápisu nekonečných množin:

- $a^* = \{\varepsilon, a, a^2, a^3, a^4, \ldots\}$
  určujeme, že prvek a se má v řetězci opakovat jakýkolivpočetkrát (nula opakování, jedno, dvě, atd.), výsledkem je množina všech řetězců obsahujících pouze symboly a (tato množina obsahuje i prázdné slovo ε s nula výskyty symbolu a)
- $(ab)^* = \{\varepsilon, ab, (ab)^2, (ab)^3, (ab)^4, \ldots\} = \{\varepsilon, ab, abab, ababab, abababab, \ldots\}$
- $a(abc)^* = \{a, aabc, a(abc)^2, a(abc)^3, \ldots\}$
- $(bb)^* = \{\varepsilon, bb, (bb)^2, (bb)^3, \ldots\} = \{\varepsilon, b^2, b^4, b^6, \ldots\}$
  takto určíme sudý počet prvků
- $b(bb)^* = (bb)^*b = \{b, bbb, b(bb)^2, b(bb)^3, \ldots\} = \{b, b^3, b^5, b^7, \ldots\}$
  lichý počet prvků
- $a^*(bb)^* = \{\varepsilon, a, aa, bb, abb, \ldots\}$
  jakýkoliv počet symbolů a následovaný sudým počtem symbolů b

2.2.3 Jak zapsat jazyk

Jazyk coby množinu slov definujeme obvykle formálním zápisem. U konečných jazyků s několika málo slovy stačí tato slova vypsat, ale u jazyků s velkým množstvím slov či dokonce nekonečných to nejde.

Jeden ze způsobů, jak zapsat nekonečný jazyk, jsme si ukázali v předchozím textu – pomocí hvězdičky, která určuje násobnost prvku, jehož je exponentem.

Příklad 2.5

$L = a^*$ je množina slov (tedy jazyk) nad abecedou $\Sigma = \{a\}$, v jazyce jsou všechna slova nad touto abecedou (všechna slova skládající se pouze ze symbolů a).

Další způsob zápisu nekonečných či hodně velkých množin je uvedení formálního předpisu a upřesňujících podmínek. Vše uzavřeme do množinových závorek (složených), první část určuje formální předpis (jak mají vypadat slova jazyka), po oddělovači (to je obvykle středník, svislice nebo dvojtečka, konvence připouštějí kteroukoliv z těchto možností) následuje seznam podmínek. V zápisu často používáme proměnné, které určují vazbu mezi formálním předpisem a upřesňujícími podmínkami.

V těchto skriptech je jako oddělovač používán středník.

Příklad 2.6

Ukážeme si množinový zápis jazyka s podmínkami:

- $L_1 = \{a^i \ ; \ i \geq 0\}$
  množina všech slov nad abecedou $\Sigma = \{a\}$ (pouze symboly a), počet těchto symbolů je větší nebo roven nule (tj. jakýkoliv), tento jazyk lze zapsat i jinak: $L_1 = a^*$
- $L_2 = \{a^i \ ; \ i > 0\}$
  množina všech slov nad abecedou $\Sigma = \{a\}$ takových, že jejich délka je větší než nula (rozdíl mezi jazyky L_1 a L_2 je pouze v tom, že v L_1 je prázdné slovo ε, kdežto v jazyce L_2 není, protože ε obsahuje nula symbolů a)
- $L_3 = \{a^n b^n \ ; \ n \geq 0\}$
  množina všech slov nad abecedou $\Sigma = \{a, b\}$ takových, že první polovina slova obsahuje pouze symboly a, druhá pouze symboly b, symbolů a a b je vždy stejný počet;
  $L_3 = \{\varepsilon, ab, a^2b^2, a^3b^3, a^4b^4, \ldots\}$
- $L_4 = \{a^i b^j \ ; \ i \geq 0, \ j \geq 1\}$
  množina všech slov nad abecedou $\Sigma = \{a, b\}$ takových, že první polovina slova obsahuje pouze symboly a, druhá pouze symboly b, počet symbolů a a b může být různý a symbol b musí být ve slově alespoň jeden (to jsou rozdíly oproti jazyku L_3);
  $L_4 = \{b, ab, bb, aab, abb, bbb, aaab, aabb, abbb, bbbb, \ldots\}$
- $L_5 = \{a^i b^{i+2} c^i \ ; \ i \geq 0\}$
  množina všech slov nad abecedou $\Sigma = \{a, b, c\}$ takových, že je opět stanoveno pořadí symbolů (nejdřív a, pak b a teprve potom c), navíc počet symbolů a a c ve slově je stejný, symbolů b je o dva víc;
  $L_5 = \{bb, ab^3c, a^2b^4c^2, a^3b^5c^3, a^4b^6c^4, \ldots\}$
- $L_6 = \left\{a^{2^i} \ ; \ i \geq 0\right\}$
  množina všech slov nad abecedou $\Sigma = \{a\}$ takových, že délka slova je vždy druhou mocninou některého čísla (a to je vždy větší nebo rovno nule) – pozor, $2^0 = 1$;
  $L_6 = \{a, a^2, a^4, a^8, a^{16}, a^{32}, \ldots\}$
- $L_7 = \{(01)^n 001^i \ ; \ n, i \geq 2\}$
  množina všech slov nad abecedou $\Sigma = \{0, 1\}$ takových, že v první části slova se střídají symboly 0 a 1 (nejméně dvakrát, protože $n \geq 2$), následují dvě nuly a jakýkoliv počet jedniček, opět nejméně dvě (protože $i \geq 2$);
  $L_7 = \{01010011, 010100111, 0101010011, 0101001111, \ldots\}$

2.2.4 Operace nad množinami – regulární operace

Protože budeme pracovat s jazyky, což jsou množiny slov, zopakujeme si množinové operace.

Definice 2.7 (Sjednocení jazyků)

Nechť L_1 a L_2 jsou jazyky nad abecedou Σ. Sjednocením jazyků L_1 a L_2 je takový jazyk L, který obsahuje právě slova patřící buď do jazyka L_1 nebo do jazyka L_2. Zapisujeme:

$$L = L_1 \cup L_2 = \{w \in \Sigma^* \ ; \ w \in L_1 \ \vee \ w \in L_2\} \quad \textit{(symbol } \vee \textit{ představuje logické „NEBO“)} \tag{2.1}$$

Příklad 2.7

Jsou dány tyto jazyky:

- $L_1 = \{\varepsilon, a, ab\}$
- $L_2 = \{ab, ba\}$
- $L_3 = \{a^i \ ; \ 2 \leq i \leq 6\} = \{a^2, a^3, a^4, a^5, a^6\}$

Sjednocení těchto jazyků po dvojicích bude následující:

- $L_1 \cup L_2 = \{\varepsilon, a, ab, ba\}$
  jednoduše do výsledného jazyka zahrneme všechna slova obou jazyků, slovo nacházející se v obou (ab) napíšeme jen jednou
- $L_1 \cup L_3 = \{\varepsilon, a, ab, a^2, a^3, a^4, a^5, a^6\} = \{ab\} \cup \{a^i \ ; \ 0 \leq i \leq 6\}$
  jak vidíme, z jazyka L_1 se nám dvě slova hodí k mírnému obohacení jazyka L_3, protože vyhovují základnímu předpisu pro slova toho jazyka, jen stačí mírně upravit část s podmínkou (rozsah pro i), ovšem nesmíme zapomenout ani na slovo ab
- $L_2 \cup L_3 = \{ab, ba, a^2, a^3, a^4, a^5, a^6\}$

Operace sjednocení je *komutativní* – to znamená, že operandy (zde množiny) kolem operátoru sjednocení můžeme jakkoliv přehazovat, platí: $L_1 \cup L_2 = L_2 \cup L_1$.

Poznámka:

Jak je to s prázdnou množinou? Pokud sjednocujeme jakoukoliv množinu s prázdnou množinou, nic se na této množině nezmění, je to jako když k číslu přičítáme nulu: $L \cup \emptyset = L$.

Definice 2.8 (Zřetězení slov, zřetězení jazyků)

Nechť $u = a_1a_2 \ldots a_m$, $v = b_1b_2 \ldots b_n$ jsou slova nad abecedou Σ. Jejich zřetězení označujeme $u \cdot v$, zkráceně uv, a definujeme jako slovo $uv = a_1a_2 \ldots a_mb_1b_2 \ldots b_n$. Jeho délka je $|uv| = m + n$.

Zřetězení jazyků L_1 a L_2 nad abecedou Σ je jazyk $L = L_1 \cdot L_2$ definovaný následovně:

$$L_1 \cdot L_2 = \{u \cdot v \ ; \ u \in L_1, \ v \in L_2\} \tag{2.2}$$

Operace zřetězení není komutativní, tedy obecně $w_1 \cdot w_2 \neq w_2 \cdot w_1$, $L_1 \cdot L_2 \neq L_2 \cdot L_1$. Takže pozor, žádné přehazování operandů kolem operátoru zřetězení.

Příklad 2.8

Vše si opět ukážeme na příkladech.

Zřetězení slov je intuitivní, tím se nemusíme podrobněji zabývat – například zřetězením slov $u = abc$, $v = xyz$ je slovo $u \cdot v = abc \cdot xyz = abcxyz$. V případě prázdného slova ε si stačí uvědomit, že se jedná o řetězec o délce nula. Tedy pro jakékoliv slovo w platí: $w \cdot \varepsilon = \varepsilon \cdot w = w$.

Soustřeďme se dále na zřetězení jazyků. Vezměme si následující jazyky:

- $L_1 = \{ab, cd\}$
- $L_2 = \{\varepsilon, xy, bb\}$
- $L_3 = \{a^n \ ; \ n \geq 1\}$
- $L_4 = \{\varepsilon\}$
- $L_5 = \{b^n \ ; \ n \geq 1\}$
- $L_6 = \{(01)^i \ ; \ i \geq 0\}$

Výsledkem zřetězení některých dvojic těchto jazyků jsou tyto jazyky:

- $L_1 \cdot L_2 = \{ab \cdot \varepsilon, ab \cdot xy, ab \cdot bb, cd \cdot \varepsilon, cd \cdot xy, cd \cdot bb\} = \{ab, abxy, abbb, cd, cdxy, cdbb\}$
- $L_2 \cdot L_1 = \{\varepsilon \cdot ab, \varepsilon \cdot cd, xy \cdot ab, xy \cdot cd, bb \cdot ab, bb \cdot cd\} = \{ab, cd, xyab, xycd, bbab, bbcd\}$
  zde na vlastní oči vidíme, že operace zřetězení není komutativní – stačí přehodit operandy (zde jazyky L_1 a L_2) a můžeme získat jiný výsledek
- $L_1 \cdot L_4 = \{ab \cdot \varepsilon, cd \cdot \varepsilon\} = \{ab, cd\} = L_1$
  pokud ke slovu jazyka L_1 přidáme nula znaků (slovo ε), získáme totéž slovo, nezmění se
- $L_3 \cdot L_4 = \{a^n \cdot \varepsilon \ ; \ n \geq 1\} = \{a^n \ ; \ n \geq 1\} = L_3$
  pokud každé slovo jazyka L_3 zřetězíme se slovem ε z jazyka L_4, získáme opět právě slova jazyka L_3
- $L_3 \cdot L_5 = \{a^n \ ; \ n \geq 1\} \cdot \{b^n \ ; \ n \geq 1\} = \{a^n b^i \ ; \ n, i \geq 1\}$
  tady pozor – ve výsledném slově budou nejdřív písmena a, pak písmena b, ovšem mezi počtem a a b není definována žádná vazba (tj. do jazyka $L_3 \cdot L_5$ patří slovo $aabb$, ale taky slovo $abbb$ nebo $aaaabb$), proto potřebujeme dva různé indexy n, i
- $L_2 \cdot L_2 = \{\varepsilon \cdot \varepsilon, \varepsilon \cdot xy, \varepsilon \cdot bb, xy \cdot \varepsilon, xy \cdot xy, xy \cdot bb, bb \cdot \varepsilon, bb \cdot xy, bb \cdot bb\}$
  $= \{\varepsilon, xy, bb, xyxy, xybb, bbxy, bbbb\}$
- $L_6 \cdot L_6 = \{(01)^i \ ; \ i \geq 0\} \cdot \{(01)^i \ ; \ i \geq 0\} = \left\{(01)^i(01)^k \ ; \ i, k \geq 0\right\} = \{(01)^i \ ; \ i \geq 0\} = L_6$

Poznámka:

Podívejme se, jak je to s prázdnou množinou. Zatímco u slov platí $w \cdot \varepsilon = w$, u jazyků a prázdné množiny je to následovně: $L \cdot \emptyset = \emptyset$.

Proč? Protože zřetězení jazyků je definováno jako množina obsahující výsledky zřetězení dvojic slov těchto jazyků, a pokud v jednom z jazyků ($\emptyset$) neexistuje žádné slovo, které bychom mohli zřetězit, je výsledkem prázdná množina. Představme si tuto situaci jako šachový turnaj: máme dvě družstva, z nichž jedno není schopno postavit žádného hráče. Pak můžeme turnaj rovnou zrušit, protože žádné dvojice protivníků nedokážeme vytvořit.

Ale pozor – jak jsme viděli v příkladu 2.8, platí $L \cdot \{\varepsilon\} = L$.

Než se zaměříme třetí regulární operaci – iteraci, definujeme n-tou mocninu slova a jazyka. Jedná se o rozšíření operace zřetězení na n prvků (slov, jazyků).

Definice 2.9 (n-tá mocnina slova a jazyka)

Nechť w je slovo nad abecedou Σ. Pak n-tou mocninou slova w je slovo $w^n = \underbrace{w \cdot w \cdot \ldots \cdot w}_{n\text{-krát}}$.
Přesněji (iterativní definice):

- $w^0 = \varepsilon$
- $w^n = w \cdot w^{n-1}$

Nechť L je jazyk nad abecedou Σ. Pak n-tou mocninou jazyka L je jazyk $L^n = \underbrace{L \cdot L \cdot \ldots \cdot L}_{n\text{-krát}}$.
Přesněji (iterativní definice):

- $L^0 = \{\varepsilon\}$
- $L^n = L \cdot L^{n-1}$

Jak vidíme, n-tá mocnina není nic těžkého – jen zřetězíme n-krát základ této mocniny. Iterace (Kleeneho uzávěr, také operace hvězdička) je určena podobně, jen jako n postupně dosazujeme všechna přirozená čísla (počítejme zde i číslo 0).

Definice 2.10 (Iterace, Kleeneho uzávěr)

Nechť w je slovo nad abecedou Σ. Pak iterací (Kleeneho uzávěrem) slova w je množina (jazyk)

$$w^* = \{w^n \ ; \ n \geq 0\} \tag{2.3}$$

Nechť L je jazyk nad abecedou Σ. Pak iterací (Kleeneho uzávěrem) jazyka L je jazyk

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots = \bigcup_{n=0}^{\infty} L^n \tag{2.4}$$

Příklad 2.9

Soustřeďme se dále na zřetězení jazyků. Vezměme si následující jazyky:

- $L_1 = \{\varepsilon\}$
- $L_2 = \{aa\}$
- $L_3 = \{ab, aa\}$
- $L_4 = \{\varepsilon, a\}$
- $L_5 = \{a^n \ ; \ n \geq 0\}$
- $L_6 = \{b^n \ ; \ n \geq 1\}$
- $L_7 = \{(01)^i \ ; \ i \geq 0\}$
- $L_8 = \{a^i b^j \ ; \ i, j \geq 1\}$
- $L_9 = \{a^i b^i \ ; \ i \geq 0\}$

Jak budou vypadat iterace těchto jazyků? Vyjdeme ze základního předpisu $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$ podle vzorce (2.4) v předchozí definici:

- $L_1^* = \{\varepsilon\} \cup \{\varepsilon\} \cup \{\varepsilon\} \cup \ldots = \{\varepsilon\}$
- $L_2^* = \{\varepsilon\} \cup \{aa\} \cup \{(aa)^2\} \cup \{(aa)^3\} \cup \ldots = \{(aa)^n \ ; \ n \geq 0\}$
- $L_3^* = \{\varepsilon\} \cup \{ab, aa\} \cdot \{ab, aa\} \cup \{ab, aa\}^3 \cup \{ab, aa\}^4 \cup \ldots$
  $= \{\varepsilon\} \cup \{ab, aa\} \cup \{abab, abaa, aaab, aaaa\}$
  $\cup \{ababab, ababaa, abaaab, abaaaa, aaabab, aaabaa, aaaaab, aaaaaa\} \cup \ldots$
  nebudeme si zbytečně komplikovat život a prostě zapíšeme $\{ab, aa\}^*$

- $L_4^* = \{\varepsilon\} \cup \{\varepsilon, a\} \cup \{\varepsilon, a, aa\} \cup \{\varepsilon, a, aa, aaa\} \cup \ldots = a^* = \{a^i \ ; \ i \geq 0\}$
- $L_5^* = \{\varepsilon\} \cup \{a^n \ ; \ n \geq 0\} \cup \{a^n \ ; \ n \geq 0\} \cdot \{a^n \ ; \ n \geq 0\} \cup \{a^n \ ; \ n \geq 0\}^3 \ldots$
  $= \{\varepsilon\} \cup a^* \cup a^* \cdot a^* \cup a^* \cdot a^* \cdot a^* \cup \ldots = a^* = \{a^n \ ; \ n \geq 0\} = L_5$
  jak vidíme, pro tento jazyk platí $L_5^* = L_5$
- $L_6^* = \{\varepsilon\} \cup \{b^n \ ; \ n \geq 1\} \cup \{b^n \ ; \ n \geq 1\} \cdot \{b^n \ ; \ n \geq 1\} \cup \{b^n \ ; \ n \geq 1\}^3 \ldots$
  $= b^* = \{b^n \ ; \ n \geq 0\} = L_6 \cup \{\varepsilon\}$
  pozor, toto je jiný případ než u jazyka L_5, mezi L_6 a L_6^* je rozdíl v jednom slově
- $L_7^* = \{\varepsilon\} \cup \{(01)^i \ ; \ i \geq 0\} \cup \{(01)^i \ ; \ i \geq 0\} \cdot \{(01)^i \ ; \ i \geq 0\} \cup \ldots = \{(01)^i \ ; \ i \geq 0\} = L_7$
- $L_8^* = \{\varepsilon\} \cup \{a^i b^j \ ; \ i, j \geq 1\} \cup \{a^i b^j \ ; \ i, j \geq 1\} \cdot \{a^i b^j \ ; \ i, j \geq 1\} \cup \ldots$
  $= \left\{(a^i b^j)^k \ ; \ i, j \geq 1, \, k \geq 0\right\}$
  výsledný jazyk vypadá o něco složitěji, ale ve skutečnosti se jeho zápis dá hodně zjednodušit: $L_8^* = \{a, b\}^*$ – stačí si uvědomit, že ve slovech jazyka se budou střídat písmena a a b, v jakémkoliv množství, jakýkolivpočetkrát
- $L_9^* = \{\varepsilon\} \cup \{a^i b^i \ ; \ i \geq 0\} \cup \{a^i b^i \ ; \ i \geq 0\} \cdot \{a^i b^i \ ; \ i \geq 0\} \cup \ldots = \left\{(a^i b^i)^k \ ; \ i, k \geq 0\right\}$
  tento jazyk nelze zjednodušit tak jako předchozí, jen můžeme některý z indexů nechat jít až od 1 místo od 0

Poznámka:

A co prázdná množina (u nás prázdný jazyk)? Platí $\emptyset^* = \{\varepsilon\} \cup \emptyset \cup \emptyset^2 \cup \ldots$, tedy $\emptyset^* = \{\varepsilon\}$. Jinak řečeno: Cokoliv na hvězdičku je jazyk obsahující minimálně prázdné slovo.

V této části kapitoly o regulárních výrazech jsme se zabývali pouze *regulárními operacemi* – to je sjednocení, zřetězení a iterace. Tyto operace jsou odděleny od ostatních z jednoho velmi důležitého důvodu – právě tyto operace se používají při tvorbě regulárních výrazů, jak uvidíme o několik stránek dále.

2.2.5 Ostatní množinové operace

Pozitivní iterace je definována téměř stejně jako iterace, jen do výsledného jazyka automaticky nedoplňujeme prázdné slovo.

Definice 2.11 (Pozitivní iterace)

Nechť w je slovo nad abecedou Σ. Pak pozitivní iterací slova w je množina (jazyk)

$$w^+ = \{w^n \ ; \ n \geq 1\} \tag{2.5}$$

Nechť L je jazyk nad abecedou Σ. Pak pozitivní iterací jazyka L je jazyk

$$L^+ = L^1 \cup L^2 \cup \ldots = \bigcup_{n=1}^{\infty} L^n \tag{2.6}$$

V samotných definicích se rozdíl mezi iterací a pozitivní iterací projevuje změnou v dolní hranici indexu n – místo hodnoty 0 je zde hodnota 1. Z toho vyplývá, že pokud v původním jazyce nebylo prázdné slovo, nebude ani v jeho pozitivní iteraci. Nicméně – pokud v původním jazyce slovo ε je, pak bude samozřejmě i v pozitivní iteraci jazyka.

Poznámka:

Symbol + používaný v exponentu při zápisu pozitivní iterace je poněkud zrádný, může se plést s binárním operátorem + (což zjistíme, až definujeme regulární výrazy). Proto ho nebudeme používat, pokud to nebude vysloveně nutné. Ono to vlastně ani nutné není, protože platí:

$$L^+ = L \cdot L^* \tag{2.7}$$

Příklad 2.10

Srovnáme iteraci a pozitivní iteraci u několika jazyků:

Jazyk L	**Iterace** L^*	**Pozitivní iterace** L^+
$\{a\}$	$a^* = \{a^i \ ; \ i \geq 0\}$	$a^+ = \{a^i \ ; \ i \geq 1\}$
$\{\varepsilon\}$	$\{\varepsilon\}$	$\{\varepsilon\}$
$\{\varepsilon, a\}$	$a^* = \{a^i \ ; \ i \geq 0\}$	$a^* = \{a^i \ ; \ i \geq 0\}$
$\{a, b\}$	$\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$	$\{a, b\}^+ = \{a, b, aa, ab, ba, bb, aaa, \ldots\}$
$\{a^i \ ; \ i \geq 0\}$	$\{a^i \ ; \ i \geq 0\}$	$\{a^i \ ; \ i \geq 0\}$
$\{a^i \ ; \ i \geq 1\}$	$\{a^i \ ; \ i \geq 0\}$	$\{a^i \ ; \ i \geq 1\}$

Definice 2.12 (Průnik jazyků)

Nechť L_1 a L_2 jsou jazyky nad abecedou Σ. Průnikem jazyků L_1 a L_2 je takový jazyk L, který obsahuje právě slova patřící jak do jazyka L_1, tak i do jazyka L_2. Zapisujeme:

$$L = L_1 \cap L_2 = \{w \in \Sigma^* \ ; \ w \in L_1 \ \wedge \ w \in L_2\} \quad \textit{(symbol } \wedge \textit{ je logické „A ZÁROVEŇ“)} \tag{2.8}$$

Příklad 2.11

Jsou dány tyto jazyky:

- $L_1 = \{\varepsilon, a, aa, ab\}$
- $L_2 = \{ab, ba\}$
- $L_3 = \{a^i \ ; \ i \geq 0\}$

Průnik těchto jazyků po dvojicích bude následující:

- $L_1 \cap L_2 = \{ab\}$
  do výsledného jazyka zahrneme pouze ta slova, která jsou v obou, zde je to jen slovo ab

- $L_1 \cap L_3 = \{\varepsilon, a, aa\}$
  tyto dva jazyky mají společná tři slova
- $L_2 \cap L_3 = \emptyset$
  tyto jazyky nemají žádné společné slovo, jsou *disjunktní* (jejich průnik je prázdný)

Průnik je také komutativní operace, tj. $L_1 \cap L_2 = L_2 \cap L_1$.

Poznámka:

O průniku s prázdnou množinou platí, že $L \cap \emptyset = \emptyset$ – výsledkem průniku dvou jazyků je množina slov nacházejících se v obou jazycích zároveň, ale v prázdném jazyce žádná slova nejsou.

Definice 2.13 (Rozdíl jazyků)

Nechť L_1 a L_2 jsou jazyky nad abecedou Σ. Rozdílem jazyků L_1 a L_2 je takový jazyk L, který obsahuje právě ta slova patřící jak do jazyka L_1, která se nenacházejí v jazyce L_2. Zapisujeme:

$$L = L_1 - L_2 = \{w \in \Sigma^* \ ; \ w \in L_1 \ \wedge \ w \notin L_2\} \tag{2.9}$$

O operaci rozdílu jazyků platí totéž co o podobné aritmetické operaci rozdílu čísel: není to komutativní operace.

Příklad 2.12

Jsou dány tyto jazyky:

- $L_1 = \{\varepsilon, a, aa, ab\}$
- $L_2 = \{ab, ba\}$
- $L_3 = \{a^i \ ; \ i \geq 0\}$

Rozdíl těchto jazyků po dvojicích bude následující:

- $L_1 - L_2 = \{\varepsilon, a, aa\}$
  $L_2 - L_1 = \{ba\}$
  jako základ vezmeme vždy první uvedený jazyk (před operátorem $-$), odstraníme z něj všechna slova, která jsou v druhém uvedeném jazyce
- $L_1 - L_3 = \{ab\}$
  $L_3 - L_1 = \{a^i \ ; \ i \geq 3\}$
  v druhém případě jsme posunuli spodní hranici pro exponent i na hodnotu 3, abychom vyloučili slova a^0, a^1, a^2, která se nacházejí v obou jazycích
- $L_2 - L_3 = \{ab, ba\} = L_2$
  $L_3 - L_2 = \{a^i \ ; \ i \geq 0\} = L_3$
  tyto jazyky jsou *disjunktní*, proto žádná slova nevyřazujeme

Nad toutéž abecedou Σ může být definováno mnoho různých jazyků, nicméně jeden z nich je vždy nejúplnější – jazyk všech slov, která lze z prvků dotyčné abecedy utvořit. Tento jazyk značíme Σ^*.

Definice 2.14 (Doplněk jazyka)

Nechť L je jazyk nad abecedou Σ. Doplňkem jazyka L (vzhledem k Σ^*) je jazyk obsahující všechna slova nad abecedou Σ kromě slov jazyka L:

$$\overline{L} = \Sigma^* - L = \{w \in \Sigma^* \ ; \ w \notin L\} \tag{2.10}$$

Příklad 2.13

Je důležité si uvědomit, že doplněk se vztahuje vždy k určité nadmnožině, obvykle se tedy jedná o množinu Σ^*. Ukažme si vytvoření doplňku jazyka na několika příkladech:

- $L_1 = \{a^i \ ; \ i \geq 1\}$ je jazyk nad abecedou $\Sigma_1 = \{a\}$
  $\overline{L_1} = \Sigma_1^* - L_1 = \{a^i \ ; \ i \geq 0\} - \{a^i \ ; \ i \geq 1\} = \{\varepsilon\}$
- $L_2 = \{a^i \ ; \ i \geq 0\}$ je jazyk nad abecedou $\Sigma_2 = \{a\}$
  $\overline{L_2} = \Sigma_2^* - L_2 = \{a^i \ ; \ i \geq 0\} - \{a^i \ ; \ i \geq 0\} = \emptyset$
- $L_3 = \{w \in \{a,b\}^* \ ; \ |w| < 5\}$ je jazyk nad abecedou $\Sigma_3 = \{a,b\}$
  $\overline{L_3} = \Sigma_3^* - L_3 = \{w \in \{a,b\}^* \ ; \ |w| \geq 5\}$

Definice 2.15 (Zrcadlový obraz, reverze)

Nechť $w = a_1 a_2 \dots a_n$ je slovo nad abecedou Σ. Zrcadlovým obrazem (reverzí) slova w je slovo $w^R = a^n \dots a^2 a^1$. Je zřejmé, že platí $(w^R)^R = w$.

Nechť L je jazyk nad abecedou Σ. Zrcadlovým obrazem (reverzí) jazyka L je jazyk obsahující reverzi všech slov jazyka L:

$$L^R = \left\{w \in \Sigma^* \ ; \ w^R \in L\right\} \tag{2.11}$$

Vytvořit zrcadlový obraz jazyka je snadné, stačí zrcadlově převrátit všechna slova. Výsledný jazyk má stejný počet slov jako původní, zrcadlovým obrazem prázdného jazyka je opět prázdný jazyk.

Příklad 2.14

Vytvoříme zrcadlový obraz několika slov a jazyků:

- $w_1 = abcd, \ w_1^R = dcba$
- $w_2 = \varepsilon, \ w_2^R = \varepsilon = w_2$
  tady není co převracet, proto $w_2^R = w_2$
- $w_3 = \text{jelenovipivonelej}, \ w_3^R = \text{jelenovipivonelej} = w_3$
  slovo, jehož reverze je stejná jako původní slovo, se nazývá *palindrom*, může to být i věta (předchozí slovo je vlastně taky palindrom)
- $L_1 = \{abcd\}, \ L_1^R = \{dcba\}$

- $L_2 = \{abc, xyz, 0123\}$, $L_2^R = \{cba, zyx, 3210\}$
  v reverzi jazyka provedeme reverzi jednotlivých slov
- $L_3 = \{a^n \ ; \ n \geq 0\}$, $L_3^R = \{a^n \ ; \ n \geq 0\} = L_3$
  jazyky nad jednoprvkovou abecedou jsou vždy shodné se svou reverzí
- $L_4 = \{abc, cba, 012, 210\}$, $L_4^R = \{cba, abc, 210, 012\} = L_4$
  dvojice slov jazyka jsou navzájem reverzní, tedy platí $L_4^R = L_4$ (uspořádání zde neřešíme)
- $L_5 = \{a^i b^j \ ; \ i, j > 0\}$, $L_5^R = \{b^j a^i \ ; \ i, j > 0\}$

Definice 2.16 (Homomorfismus)

Homomorfismus je zobrazení $h : \Sigma \to \Delta^*$ *splňující* homomorfní podmínky*:*

- $h(\varepsilon) = \varepsilon$
- $h(a \cdot x) = h(a) \cdot h(x)$, *kde* $a \in \Sigma$, $x \in \Sigma^*$

Abeceda Σ *je vstupní abecedou, abeceda* Δ *je výstupní abecedou zobrazení.*

Homomorfismus slova definujeme jako zobrazení $h : \Sigma^* \to \Delta^*$ *s výše uvedenými podmínkami. Homomorfismus aplikovaný na jazyk* L *je zobrazení* $h(L) = \{h(w) \ ; \ w \in L\}$.

Je důležité si uvědomit, že předchozí definice homomorfismu se vztahuje pouze na homomorfismus aplikovaný na znaky a řetězce a operaci zřetězení.

První homomorfní podmínka určuje, že zobrazení má zachovávat *neutrální prvek* (nulový prvek). Neutrální prvek je takový prvek množiny (zde množiny řetězců), který při zřetězení s jakýmkoliv jiným prvkem množiny nijak neovlivní výsledek. U řetězců a operace zřetězení je to prázdné slovo ε, protože $\varepsilon \cdot w = w \cdot \varepsilon = w$ pro jakékoliv slovo $w \in \Sigma^*$. Pokud jako základní množinu vezmeme množinu celých čísel a jako operaci sčítání, neutrálním prvkem bude číslo 0, protože $0 + n = n + 0 = n$ pro jakékoliv celé číslo n.

Druhá homomorfní podmínka stanovuje zachování výsledku operace zřetězení – homomorfismus zřetězení odpovídá zřetězení homomorfismů (tj. je jedno, kterou z operací provedeme dřív – zřetězení nebo homomorfismus).

Příklad 2.15

Určeme si tyto tři homomorfismy:

- $f(a) = abc$
  $f(b) = cc$
  $f(c) = \varepsilon$
- $g(a) = 000$
  $g(b) = 0100$
  $g(c) = 11$
- $h(a) = ❀$
  $h(b) = ❀$
  $h(c) = ✺$

Nyní podle definovaných homomorfismů zobrazíme následující slova a jazyky:

- $w_1 = aabbcc$
  $f(w_1) = abc \cdot abc \cdot cc \cdot cc \cdot \varepsilon \cdot \varepsilon = abcabccccc$
  $g(w_1) = 000 \cdot 000 \cdot 0100 \cdot 0100 \cdot 11 \cdot 11 = 000000010001001111$
  $h(w_1) = ❀ \cdot ❀ \cdot ❀ \cdot ❀ \cdot ✺ \cdot ✺ = ❀❀❀❀✺✺$
- $w_2 = \varepsilon$ $\quad f(w_2) = \varepsilon, \ g(w_2) = \varepsilon, \ h(w_2) = \varepsilon$ (homomorfismus zachovává prázdné slovo)

- $w_3 = bcb$ $\quad f(w_3) = cc \cdot \varepsilon \cdot cc = cccc$
  $\quad g(w_3) = 0100 \cdot 11 \cdot 0100 = 0100110100$
  $\quad h(w_3) = ❀ \cdot ✺ \cdot ❀ = ❀✺❀$
- $L_1 = \{a, ab, cc\}$ $\quad f(L_1) = \{abc\,,\ abc \cdot cc\,,\ \varepsilon \cdot \varepsilon\} = \{abc, abccc\,,\ \varepsilon\}$
  $\quad g(L_1) = \{000\,,\ 000 \cdot 0100\,,\ 11 \cdot 11\} = \{000\,,\ 0000100\,,\ 1111\}$
  $\quad h(L_1) = \{❀\,,\ ❀ \cdot ❀\,,\ ✺ \cdot ✺\} = \{❀\,,\ ❀❀\,,\ ✺✺\}$
- $L_2 = \{a^n\ ;\ n \geq 0\}$ $\quad f(L_2) = \{(abc)^n\ ;\ n \geq 0\}$
  $\quad g(L_2) = \{(000)^n\ ;\ n \geq 0\} = \{0^{3n}\ ;\ n \geq 0\}$
  $\quad h(L_2) = \{❀^n\ ;\ n \geq 0\}$
- $L_3 = \{a^i c^i\ ;\ i > 0\}$ $\quad f(L_3) = \{(abc)^i \varepsilon^i\ ;\ i > 0\} = \{(abc)^i\ ;\ i > 0\}$
  $\quad g(L_3) = \{(000)^i (11)^i\ ;\ i > 0\} = \{0^{3i}1^{2i}\ ;\ i > 0\}$
  $\quad h(L_3) = \left\{❀^i ✺^i\ ;\ i > 0\right\}$

Poznámka:

Právě díky homomorfním podmínkám můžeme toto zobrazení uplatňovat „po znacích“ a nejsme nuceni definovat výsledky pro celá slova jazyka.

V algebře se pojem „homomorfismus“ často zkracuje na „morfismus“ – je to totéž.

Další zobrazení, které si představíme, je rozšířením homomorfismu – v případě homomorfismu je výsledkem uplatnění zobrazení vždy slovo, u substituce je to množina slov (jazyk).

Definice 2.17 (Substituce)

Substituce je zobrazení $s : \Sigma \to 2^{\Delta^*}$ *ze vstupní abecedy* Σ *do množiny všech podmnožin množiny* Δ^* *splňující homomorfní podmínky:*

- $s(\varepsilon) = \varepsilon$
- $s(a \cdot x) = s(a) \cdot s(x)$, *kde* $a \in \Sigma$, $x \in \Sigma^*$

Substituce slova je zobrazení $s : \Sigma^* \to 2^{\Delta^*}$ *splňující výše uvedené podmínky. Substituce jazyka* L *je definována pomocí substituce slov jazyka* L*:*

$$s(L) = \bigcup_{w \in L} s(w) \tag{2.12}$$

Jak vidíme, rozdíl mezi homomorfismem a substitucí je pouze ve výsledku – u substituce budeme mít za rovnítkem místo jediného slova množinu slov.

Příklad 2.16

Určeme si tato tři substituční zobrazení:

- $f(a) = \{00, 11\}$
  $f(b) = \{0^i\ ;\ i > 0\}$
- $g(a) = \{ab, ba\}$
  $g(b) = \{\varepsilon\}$
- $h(a) = \{a^n\ ;\ n \geq 0\}$
  $h(b) = \left\{a^{2^i}\ ;\ i \geq 0\right\}$

Nyní podle definovaných substitucí zobrazíme následující slova a jazyky:

- $w_1 = aab$
$$\begin{aligned} f(w_1) &= \{00,11\} \cdot \{00,11\} \cdot \{0^i \ ; \ i > 0\} \\ &= \{0000,0011,1100,1111\} \cdot \{0^i \ ; \ i > 0\} \\ g(w_1) &= \{ab,ba\} \cdot \{ab,ba\} \cdot \{\varepsilon\} = \{abab,abba,baab,baba\} \\ h(w_1) &= \{a^n \ ; \ n \geq 0\} \cdot \{a^n \ ; \ n \geq 0\} \cdot \left\{a^{2^i} \ ; \ i \geq 0\right\} \\ &= \{\varepsilon,a,a^2,a^3,\ldots\} \cdot \{\varepsilon,a,a^2,a^3,\ldots\} \cdot \{a,a^2,a^4,a^8,\ldots\} = \{a^n \ ; \ n \geq 1\} \end{aligned}$$
- $w_2 = bb$
$$\begin{aligned} f(w_2) &= \{0^i \ ; \ i > 0\} \cdot \{0^i \ ; \ i > 0\} = \{0^i \ ; \ i > 0\} \\ g(w_2) &= \{\varepsilon\} \cdot \{\varepsilon\} = \{\varepsilon\} \\ h(w_2) &= \left\{a^{2^i} \ ; \ i \geq 0\right\} \cdot \left\{a^{2^i} \ ; \ i \geq 0\right\} = \left\{a^{2^i+2^j} \ ; \ i,j \geq 0\right\} \\ &= \{a,a^2,a^4,a^8,\ldots\} \cdot \{a,a^2,a^4,a^8,\ldots\} \end{aligned}$$
- $L_1 = \{ab,aa\}$
$$\begin{aligned} f(L_1) &= \{00,11\} \cdot \{0^i \ ; \ i > 0\} \cup \{00,11\} \cdot \{00,11\} \\ &= \{00,11\} \cdot \{0^i \ ; \ i > 0\} \cup \{0000,0011,1100,1111\} \\ &= \{00,11\} \cdot \{0^i \ ; \ i > 0\} \cup \{0011,1111\} \\ &\quad \text{z druhé množiny vyřadíme vše, co je již obsaženo v první} \\ g(L_1) &= \{ab,ba\} \cdot \{\varepsilon\} \cup \{ab,ba\} \cdot \{ab,ba\} \\ &= \{ab,ba,abab,abba,baab,baba\} \\ h(L_1) &= \{a^n \ ; \ n \geq 0\} \cdot \left\{a^{2^i} \ ; \ i \geq 0\right\} \cup \{a^n \ ; \ n \geq 0\} \cdot \{a^n \ ; \ n \geq 0\} \\ &= \{\varepsilon,a,a^2,\ldots\} \cdot \{a,a^2,a^4,a^8,\ldots\} \cup \{\varepsilon,a,a^2,\ldots\} \cdot \{\varepsilon,a,a^2,\ldots\} \\ &= \{a^n \ ; \ n \geq 0\} \end{aligned}$$
- $L_2 = \{b^n \ ; \ n \geq 0\}$
$$\begin{aligned} f(L_2) &= \left\{(0^i)^n \ ; \ i,n \geq 0\right\} = \{0^{i \cdot n} \ ; \ i,n \geq 0\} = \left\{0^k \ ; \ k \geq 0\right\} \\ g(L_2) &= \{\varepsilon^n \ ; \ n \geq 0\} = \{\varepsilon\} \\ h(L_2) &= \left\{\left(a^{2^i}\right)^n \ ; \ i,n \geq 0\right\} = \left\{a^k \ ; \ k \geq 0\right\} \\ &\quad \text{protože pro } i = 0 \text{ dostaneme } \{a^{1 \cdot n} \ ; \ n \geq 0\} \end{aligned}$$

2.2.6 Vlastnosti množinových a řetězcových operací

V předchozím textu byla již zmíněna vlastnost komutativity některých operací. Pro úplnost zde uvedeme definici komutativity operace nad množinou (řetězců či množin řetězců), a to pro obecnou operaci $\circ$ za kterou si můžeme dosadit například sjednocení nebo průnik.

Definice 2.18 (Komutativita)

Nechť $\circ$ je binární operace definovaná nad množinou A. Operace $\circ$ je komutativní, pokud platí:

$$x \circ y = y \circ x, \ \textit{kde } x,y \in A \tag{2.13}$$

U binární komutativní operace tedy lze zaměnit operandy bez změny výsledku. Už víme, že komutativní jsou například tyto operace:

- sjednocení množin řetězců, protože $X \cup Y = Y \cup X$
- průniku množin řetězců, protože $X \cap Y = Y \cap X$

Naopak komutativní nejsou tyto operace:

- zřetězení prvků/řetězců, protože obecně $u \cdot v \neq v \cdot u$ (třebaže v některých konkrétních případech může platit rovnost)
- zřetězení množin (jazyků), protože obecně $X \cdot Y \neq Y \cdot X$
- rozdíl množin řetězců, protože obecně $X - Y \neq Y - X$

Z aritmetických operací jsou komutativní sčítání a násobení, naopak nejsou komutativní odčítání a dělení. Komutativitu uvažujeme pouze u binárních operací, tedy u jiných tuto vlastnost nemá smysl stanovovat (například iterace či doplněk jsou unární operace).

Další vlastností, kterou uvažujeme u binárních operací, je asociativita.

Definice 2.19 (Asociativita)

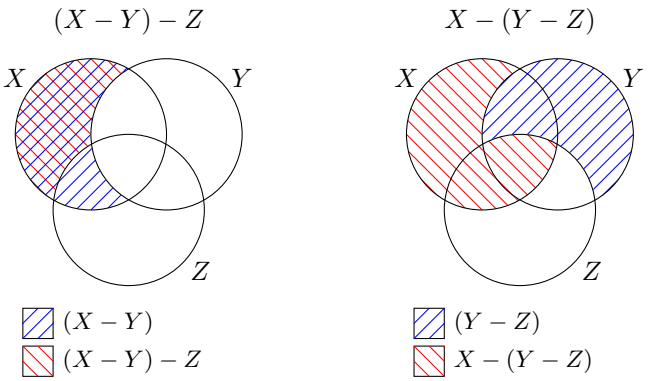
Nechť $\circ$ je binární operace definovaná nad množinou A. Operace $\circ$ je asociativní, pokud platí:

$$(x \circ y) \circ z = x \circ (y \circ z), \text{ kde } x, y, z \in A \tag{2.14}$$

Pokud je operace asociativní, můžeme ji dle předpisu v definici „přezávorkovat“. Asociativní jsou například tyto operace:

- sjednocení množin řetězců, protože $(X \cup Y) \cup Z = X \cup (Y \cup Z)$
- průnik množin řetězců, protože $(X \cap Y) \cap Z = X \cap (Y \cap Z)$
- zřetězení prvků/řetězců, protože $(u \cdot v) \cdot w = u \cdot (v \cdot w)$
- zřetězení množin (jazyků), protože $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$

Asociativní není operace rozdílu množin řetězců, protože obecně $(X - Y) - Z \neq X - (Y - Z)$, jak vidíme na Vennových diagramech na obrázku 2.1.



Obrázek 2.1: Operace rozdílu množin není asociativní

Definice 2.20 (Distributivita)

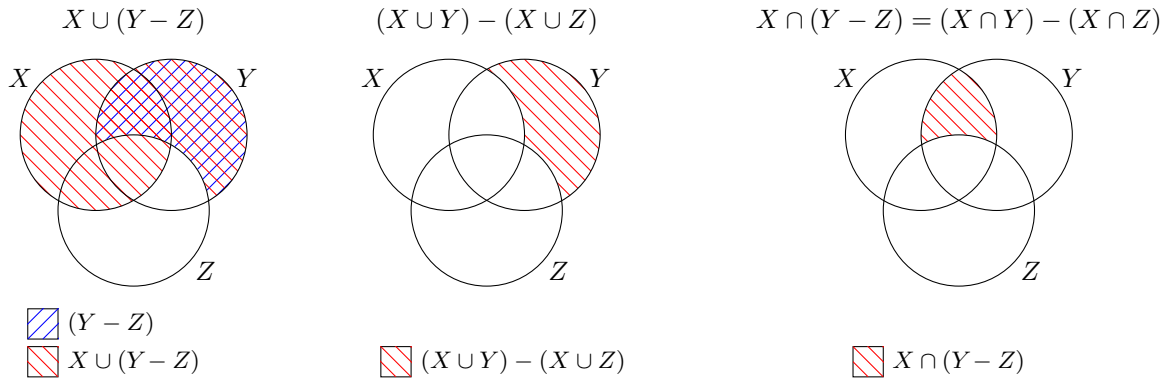
Nechť $\circ_1$ a $\circ_2$ jsou binární operace definované nad množinou A. Operace $\circ_1$ je distributivní vzhledem k operaci $\circ_2$ na dané množině, pokud platí:

$$x \circ_1 (y \circ_2 z) = (x \circ_1 y) \circ_2 (x \circ_1 z), \text{ kde } x, y, z \in A \tag{2.15}$$

Distributivitu běžně využíváme v aritmetice – operace součinu (násobení reálných čísel) je distributivní vzhledem k operaci sčítání na množině reálných čísel: $x \cdot (y+z) = (x \cdot y)+(x \cdot z)$. V případě množinových operací je to následovně:

- operace sjednocení je distributivní vzhledem k průniku množin:
$$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$$
- operace průniku je distributivní vzhledem k sjednocení množin:
$$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$$
- operace zřetězení je distributivní vzhledem k sjednocení i průniku množin:
$$X \cdot (Y \cup Z) = (X \cdot Y) \cup (X \cdot Z)$$
$$X \cdot (Y \cap Z) = (X \cdot Y) \cap (X \cdot Z)$$
- operace sjednocení není distributivní vzhledem k rozdílu množin, kdežto operace průniku ano:
$$X \cup (Y - Z) \neq (X \cup Y) - (X \cup Z)$$
$$X \cap (Y - Z) = (X \cap Y) - (X \cap Z)$$

Naopak operace rozdílu není distributivní vzhledem k sjednocení ani průniku.



Obrázek 2.2: Distributivita sjednocení a průniku vzhledem k rozdílu množin

Co se týče aritmetických operací, je zajímavé, že zatímco operace násobení čísel je distributivní vzhledem ke sčítání, ale naopak to neplatí (operace sčítání čísel není distributivní vzhledem k násobení). Podobnými vztahy bychom se mohli zabývat ještě dlouho, nicméně to je oblast patřící spíše do předmětu zabývajícího se algebrou.

Další užitečné vzorce, které se často používají v důkazech o množinách, jsou De Morganova pravidla. Tato pravidla jsou postavena na vlastnostech průniku, sjednocení a doplňku množin, také je možné je použít v případě logických operací (konjunkce, disjunkce, negace). Jejich autorem je britský matematik August De Morgan žijící v 19. století.

Definice 2.21 (De Morganova pravidla)

Nechť A a B jsou množiny. Pak platí následující vztahy:

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \tag{2.16}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B} \tag{2.17}$$

Tyto vztahy se dají zapsat i trochu jinak, například pro první: $A \cup B = \overline{\overline{A} \cap \overline{B}}$.

2.2.7 Pár řetězcových operací navíc

Následují definice několika operací nad jazyky, které se používají zejména při zpracování textu.

Definice 2.22 (Levý a pravý derivát jazyka)

Levý derivát jazyka L podle slova x je množina všech slov takových, že pokud je k nim zleva operací zřetězení přidáno slovo x, patří do jazyka L.

$$\delta_x^l(L) = \{w \in \Sigma^* \ ; \ x \cdot w \in L\} \tag{2.18}$$

Pravý derivát jazyka L podle slova x je množina všech slov takových, že pokud je k nim zprava operací zřetězení přidáno slovo x, patří do jazyka L.

$$\delta_x^r(L) = \{w \in \Sigma^* \ ; \ w \cdot x \in L\} \tag{2.19}$$

Jak vidíme, levý derivát vlastně pracuje s předponami (prefixy) slov. Je dán jazyk a dále řetězec, v jazyce hledáme všechna slova, která tímto řetězcem začínají. Výsledkem je množina takto nalezených slov s tím, že dotyčnou předponu (řetězec na začátku slova) odstraníme. Slova původního jazyka, která hledanou předponu vůbec nemají, se do výsledku nedostanou.

S pravým derivátem je to podobné, jen zde pracujeme s příponami (postfixy) místo předpon. Zjistíme, která slova daného jazyka končí předepsaným řetězcem (příponou), tato slova vybereme a odstraníme příponu.

Příklad 2.17

Pokud $L = \{abc, abbba, ccbd, acab, cba\}$, pak

- $\delta_{ab}^l(L) = \{c, bba\}$
- $\delta_a^l(L) = \{bc, bbba, cab\}$
- $\delta_{ba}^r(L) = \{abb, c\}$

Pokud $L = \{(ab)^i(ba)^i \ ; \ i \geq 2\}$, pak

- $\delta_{ab}^l(L) = \{(ab)^{i-1}(ba)^i \ ; \ i \geq 2\}$
- $\delta_{baba}^r(L) = \{(ab)^i(ba)^{i-2} \ ; \ i \geq 2\} = \{(ab)^{i+2}(ba)^i \ ; \ i \geq 0\}$
- $\delta_{(ba)^4}^r(L) = \{(ab)^{i+4}(ba)^i \ ; \ i \geq 0\}$

Všimněte si, že některá slova jazyka L se ve výsledném jazyce vůbec neprojevila.

Levý a pravý kvocient je zobecněním levého a pravého derivátu.

Definice 2.23 (Levý a pravý kvocient)

Levý kvocient *jazyka L_1 vzhledem k jazyku L_2 je sjednocením levých derivátů jazyka L_1 vzhledem ke všem slovům jazyka L_2.*

$$\begin{aligned} L_2 \backslash L_1 &= \{w \in \Sigma^* \ ; \ \text{existuje } x \in L_2 \text{ tak, že } x \cdot w \in L_1\} \\ &= \left\{w \in \delta_x^l(L_1) \ ; \ x \in L_2\right\} \end{aligned} \tag{2.20}$$

Pravý kvocient *jazyka L_1 vzhledem k jazyku L_2 je sjednocením pravých derivátů jazyka L_1 vzhledem ke všem slovům jazyka L_2.*

$$\begin{aligned} L_1 / L_2 &= \{w \in \Sigma^* \ ; \ \text{existuje } x \in L_2 \text{ tak, že } w \cdot x \in L_1\} \\ &= \{w \in \delta_x^r(L_1) \ ; \ x \in L_2\} \end{aligned} \tag{2.21}$$

Poznámka:

Vypadá to složitě, ale stačí si uvědomit, že tím „důležitějším“ jazykem je ten, který máme ve značení výše vzhledem k lomítku nebo opačnému lomítku (v definici je značen vždy L_1), kdežto z toho, který je ve značení níže, bereme předpony nebo přípony.

Příklad 2.18

Jsou dány tyto jazyky:

$L_1 = \{ab, aab, ac\}$ $\quad$ $L_3 = a^*$ $\quad$ $L_5 = (ab)^*$ $\quad$ $L_7 = \{\varepsilon, abc, ab\}$

$L_2 = \{a^i bc^i \ ; \ i \geq 0\}$ $\quad$ $L_4 = a^*b^*$ $\quad$ $L_6 = (ab)^+$

Stanovíme tyto levé a pravé kvocienty:

- $L_1 \backslash L_2 = \{c, cc\}$
  všimněte si, že opravdu jde o konečný jazyk, třebaže L_2 je nekonečný
- $L_2 \backslash L_1 = \emptyset$
  protože žádné slovo z jazyka L_2 není prefixem žádného slova jazyka L_1; pozor, nejkratší slovo z jazyka L_2 je b, nikoliv ε
- $L_3 \backslash L_2 = \{a^m bc^i \ ; \ 0 \leq m \leq i\}$
  postup: vezmeme slovo z L_2, například a^4bc^4, a použijeme jako prefixy postupně slova $\varepsilon, a, a^2, a^3, a^4$ z jazyka L_3 – všechna do délky počtu symbolů a ve slově z jazyka L_2, pak další slovo, atd.
- $L_4 \backslash L_2 = L_2 \cup \{a^m bc^i \ ; \ 0 \leq m \leq i\} \cup \{c^i \ ; \ i \geq 0\}$
  první část: z L_4 zvolíme ε; druhá část: z L_4 zvolíme slova a^*; třetí část: z L_4 zvolíme a^*b
  protože $L_2 \subset \{a^m bc^i \ ; \ 0 \leq m \leq i\}$, výsledek můžeme zjednodušit; jak?
- $L_5 \backslash L_2 = L_2 \cup \{c\}$
  pokud $x = \varepsilon$, pak získáme L_2; pokud $x = \{ab\}$, získáme $\{c\}$; další slova z L_4 nelze použít
- $L_6 \backslash L_2 = \{c\}$
- $L_7 \backslash L_2 = L_2 \cup \{\varepsilon, c\}$
- $L_1 / L_5 = L_1 \cup \{\varepsilon, a\}$
  pozor, teď už počítáme pravý kvocient
- $L_1 / L_6 = \{\varepsilon, a\}$

2.3 Regulární výrazy

Konečně se dostáváme k regulárním výrazům. Následující definice regulárního výrazu je *iterativní* – definujeme nejdřív počáteční podmínky (bázi), což obvykle znamená jakousi počáteční množinu, ze které se skládají další prvky definované (obvykle nekonečné) množiny, a následně pomocí iterace (rekurze) stanovíme pravidla pro vytváření těchto dalších prvků.

Definice 2.24 (Regulární výraz)

Definujeme pomocnou množinu $\Phi = \{\emptyset, \varepsilon, +, \cdot, ^*, (,)\}$.

Množina $RV(\Sigma)$ *všech regulárních výrazů nad abecedou* Σ *je nejmenší množina slov taková, že platí*

- *slova se skládají ze symbolů abecedy* $\Sigma \cup \Phi$*, přičemž* Σ *a* Φ *jsou disjunktní,*
- $\emptyset \in RV(\Sigma)$, $\varepsilon \in RV(\Sigma)$, $a \in RV(\Sigma)$ *pro každé* $a \in \Sigma$,
- *jestliže* $\alpha, \beta \in RV(\Sigma)$*, pak taky*
  $(\alpha + \beta) \in RV(\Sigma)$,
  $(\alpha \cdot \beta) \in RV(\Sigma)$,
  $(\alpha)^* \in RV(\Sigma)$.

Bází je v této definici prázdná množina, jednoprvková množina obsahující prázdné slovo a prvky množiny Σ. Poslední bod seznamu určuje iterativní podmínky určující, jak mají vypadat další prvky množiny všech regulárních výrazů.

Poznámka:

Regulární výraz označuje množinu řetězců s danou vlastností, jazyk je také množina řetězců (slov) s danou vlastností. Proto platí, že *každý regulární výraz určuje některý jazyk.*

Regulární výraz	Jazyk
$\emptyset$	$\emptyset$, tedy prázdný jazyk
ε	$\{\varepsilon\}$ (jazyk obsahující jen slovo s nulovou délkou)
$a,\ a \in \Sigma$	$\{a\}$ (jazyk obsahující jen slovo a s délkou 1)
$\alpha + \beta$	$\{\alpha\} \cup \{\beta\}$ (sjednocení)
$\alpha \cdot \beta$	$\{\alpha\} \cdot \{\beta\}$ (zřetězení)
$(\alpha)^*$	$\{\alpha\}^*$ (iterace)

Tabulka 2.1: Vztah mezi zápisem regulárních výrazů a jazyků

Teď už vidíme, proč v předchozím textu byly operace sjednocení, zřetězení a iterace odděleny v samostatné sekci s názvem *regulární operace* – právě pomocí těchto operací jsou tvořeny regulární výrazy.

Příklad 2.19

Ukážeme si několik regulárních jazyků a ekvivalentních regulárních výrazů.

$L_1 = \{a^i b^j \ ; \ i, j \geq 0\}$
$R_1 = a^* b^*$

$L_2 = \left\{a^k \ ; \ k \geq 1\right\}$
$R_2 = aa^*$

$L_3 = \{a, b\}^*$
$R_3 = (a + b)^*$

$L_4 = \{a^i c(ab)^j \ ; \ i, j \geq 0\}$
$R_4 = a^* c(ab)^*$

$L_5 = \{1^{2i} w \ ; \ i > 0, \ w \in \{0, 1\}^*\}$
$R_5 = (11)(11)^*(0 + 1)^*$

$L_6 = \{a^i b \ ; \ i > 0\} \cup \{b^i a \ ; \ i \geq 0\}$
$R_6 = aa^* b + b^* a$

$L_7 = \{\varepsilon\} \cup (\{ab^4 a^i \ ; \ i \geq 0\} \cup \{b^2 a^{2i+1} \ ; \ i \geq 0\}) \cdot \{ca^i \ ; \ i \geq 0\}$
$R_7 = \varepsilon + (abbbba^* + bba(aa)^*) \cdot ca^*$

U posledního z uvedených jazyků bychom si měli dát pozor na prioritu operátorů – zřetězení má vyšší prioritu než sjednocení (podobně jako v aritmetice má násobení vyšší prioritu než sčítání), proto při tvoření slov patřících do tohoto jazyka nejdřív řešíme zřetězení (váže více), a až potom sjednocení.

Z předchozího příkladu vyplývá, že zápis pomocí regulárního výrazu bývá obvykle kratší a často i přehlednější. Proto se tento způsob zápisu používá častěji, pokud je to možné.

Poznámka:

Pozor – každý regulární výraz můžeme vyjádřit pomocí množinového zápisu, ale naopak to neplatí. Například pro jazyky $\{a^{2^n} \ ; \ n \geq 0\}$, $\{a^i b^i \ ; \ i \geq 0\}$ a mnohé další neexistuje ekvivalentní regulární výraz, protože slova těchto jazyků nelze souhrnně vyjádřit pouze pomocí regulárních operací.

Kapitola 3

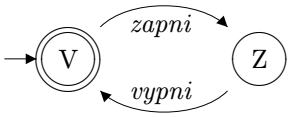
Konečné automaty

Nejdřív se budeme zabývat nejjednodušším typem modelů, které byly zmíněny v kapitole 1.1.1 o matematických kořenech teoretické informatiky, konečnými automaty, a také se podíváme na základy práce s jednoduchými gramatikami, se kterými jsme se už trochu seznámili v kapitole 1.1.2.

3.1 Konečný automat

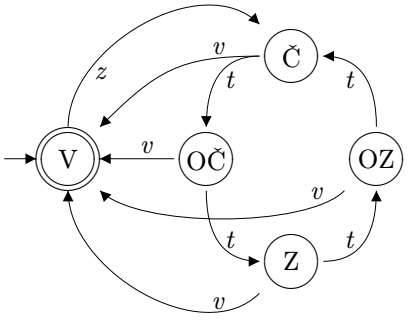
Jak už víme, konečný automat je matematický model, nicméně tento model si můžeme představit i na příkladech z praxe.

Na obrázcích 3.1, 3.2 a 3.3 vidíme několik jednoduchých diagramů (orientovaných grafů) konečných automatů. Diagram přehledně (u jednodušších automatů) zobrazuje *přechody mezi stavy* (šipky) a *signály* (symboly), které jsou při tomto přechodu načítány a zpracovávány (ohodnocení šipek).



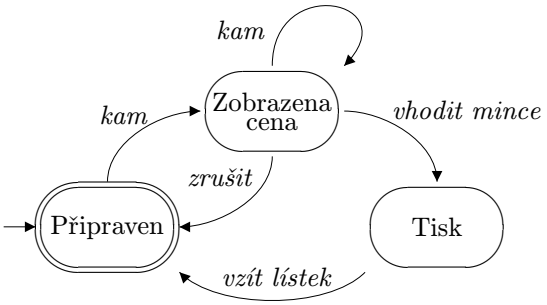
Popis:
V . . . . . vypnuto
Z . . . . . zapnuto

Obrázek 3.1: Elektrický spotřebič jako konečný automat



Popis:
V . . . . . vypnuto
Č, Z . . . . . červená, zelená
OČ . . . . . oranžová od červené
OZ . . . . . oranžová od zelené

Obrázek 3.2: Semafor jako konečný automat



Obrázek 3.3: Automat na jízdenky

Konečný automat je vždy v některém (vnitřním) stavu, který si můžeme představit jako proměnnou nabývající předem stanovených hodnot. Pracuje jednoduše tak, že na základě informace o svém momentálním stavu a dále podle signálu, který dostal (symbolu na vstupu) se přesune do některého jiného stavu a zároveň se posune na vstupu, tedy očekává další signál (je připraven načíst další symbol z pásky).

3.1.1 Definice konečného automatu

Konečný automat jsme si zatím představili jako teretický model, který postupně zpracovává zadaný vstup (v každém kroku jeden signál/symbol/znak/objekt/atd.), a zároveň mění svůj vnitřní stav (tj. posouvá se mezi různými stavy). Nyní uvedeme formální definici konečného automatu.

Definice 3.1 (Konečný automat)

Konečný automat *je uspořádaná pětice* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, *kde je*

Q ... *neprázdná konečná množina stavů*
Σ ... *neprázdná konečná abeceda (množina signálů)*
δ ... *přechodová funkce, definovaná níže*
q_0 ... *počáteční stav,* $q_0 \in Q$
F ... *množina koncových stavů,* $F \subseteq Q$, $F \neq \emptyset$

Přechodová funkce δ *konečného automatu (dále KA)* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *je definována takto:*
$\delta : \ Q \times \Sigma \to Q$ *(zápis pomocí množin)*
$\delta(q, a) = r$, *kde* $q, r \in Q$, $a \in \Sigma$ *(symbolický zápis)*

V této definici je stanoveno, jak určujeme základní atributy konkrétního konečného automatu – do jakých stavů se může dostat (například zapnuto/vypnuto), s čím dokáže pracovat (s jakými signály, znaky, objekty apod., které dostává na svůj vstup), co s nimi provádí (δ-funkce), za jakých okolností (ve kterém svém stavu) může ukončit svou činnost (například jen tehdy, když je vypnutý). Předpis přechodové funkce může vypadat například takto:
$\delta(vypnuto, zapni) = zapnuto$
To znamená, že když je stroj právě vypnutý (je ve stavu vypnuto) a dostane signál „zapni“, přejde do stavu zapnuto, tedy se zapne.

Poznámka:

V definici je nejen důležité pořadí a značení prvků zmíněné uspořádané pětice, ale také slova „neprázdná“, „konečná“. Definici sice není nutné znát doslova zpaměti, ale je třeba vždy zařadit vše, nic nevynechat.

Přechodovou funkcí se budeme zabývat i nadále. Každý její předpis nám říká, co máme provést v jednom kroku činnosti automatu. Vždy je důležité uvědomit si, podle čeho se rozhodujeme, co bude provedeno. Tedy předpis určuje:

- v závorce za symbolem δ je to, podle čeho se rozhodujeme:
  - ve kterém stavu je právě automat a
  - co právě čteme na vstupu (jaký signál automat dostal)
- za rovnítkem je to, jak máme reagovat: změnou stavu automatu.

Každý stroj potřebuje svou paměť (například běžný počítač má operační paměť, pevný disk, výměnná paměťová média, apod.), v případě konečného automatu se jedná o paměť pro stav automatu a o vstupní pásku. Shrňme si tedy, co si musí konečný automat neustále pamatovat:

- momentální stav,
- nepřečtenou část vstupní pásky (ta přečtená je už zpracovaná, není třeba ji mít v paměti).

Tyto informace jsou uloženy v *konfiguraci* automatu, která se postupně mění.

Definice 3.2 (Konfigurace, počáteční a koncová konfigurace)

Označme Σ^* množinu všech slov, která lze utvořit ze symbolů abecedy Σ. Konfigurace KA je uspořádaná dvojice (q, w), kde $q \in Q$, $w \in \Sigma^*$ (nepřečtená část vstupní pásky).

Dále definujeme tyto pojmy:

- Počáteční konfigurace *je konfigurace (q_0, w_0), kde q_0 je počáteční stav automatu a w_0 je startovací (počáteční) slovo*
- Koncová konfigurace *je konfigurace (q_f, ε), kde $q_f \in F$*

Konečný automat je v počáteční konfiguraci, pokud se nachází v počátečním stavu a na vstupu je celý vstupní řetězec (tj. ještě nezačal pracovat). Konečný automat je v koncové konfiguraci, jestliže je v některém z koncových stavů (kterémkoliv z nich) a celý vstup je přečtený.

Definice 3.3 (Přechod mezi konfiguracemi)

Relaci přechodu mezi konfiguracemi značíme symbolem $\vdash$ a definujeme ji takto:

$$(q_i, aw) \vdash (q_j, w) \quad \overset{def}{\iff} \quad \delta(q_i, a) = q_j, \quad kde\ q_i, q_j \in Q,\ a \in \Sigma,\ w \in \Sigma^* \tag{3.1}$$

Jak vidíme, přechodová funkce δ je „činnou“, dynamickou částí definice konečného automatu, protože určuje, co se má provést v jednom kroku činnosti automatu, jinými slovy – při přechodu mezi dvěma konfiguracemi automatu.

V definici (v symbolickém zápisu) stojí, že od konfigurace (q_i, aw) přecházíme ke konfiguraci (q_j, w), a to podle předpisu $\delta(q_i, a) = q_j$. V předpisu je stanoveno, že přecházíme ze stavu q_i (ten je v první konfiguraci) do stavu q_j (ten je v druhé konfiguraci), a to tehdy, když na pásce čteme symbol a (ten byl v tomto kroku „přečten“, zpracován). Jak vidíme, symbol a je v první konfiguraci na začátku nepřečtené části vstupu (je tam aw), v druhé konfiguraci symbol a zmizel (byl přečten). Řetězec w je prostě zbývající nepřečtená část vstupu.

3.1.2 Zpracování slova a jazyk automatu

Zatím máme definován jeden krok výpočtu konečného automatu – relaci *Přechodu mezi konfiguracemi*. Automat takových kroků vždy provede tolik, kolik vyžaduje zpracování vstupu.

Definice 3.4 (Výpočet slova v konečném automatu)

Výpočet slova v konečném automatu je posloupnost konfigurací začínající počáteční konfigurací s daným slovem a končící některou koncovou konfigurací, vztah mezi sousedními konfiguracemi je dán relací přechodu $\vdash$.

Příklad 3.1

Výpočet slova v konečném automatu – semaforu na obrázku 3.2 na straně 31 – může být například takový:

$(V, ztttttv) \vdash (\check{C}, ttttttv) \vdash (O\check{C}, ttttv) \vdash (Z, tttv) \vdash (OZ, ttv) \vdash (\check{C}, tv) \vdash$
$\vdash (Z, v) \vdash (V, \varepsilon)$

V jiném konečném automatu se stavy $Q = \{q_0, q_1, \ldots, q_3\}$, stav q_3 patří do množiny koncových stavů a přechodová funkce obsahuje mimo jiné předpis $\delta(q_0, a) = q_3$ (a další), může výpočet slova vypadat třeba takto:

$(q_0, abcd) \vdash (q_3, bcd) \vdash (q_1, cd) \vdash (q_3, d) \vdash (q_3, \varepsilon)$

Základem tedy je, že v každém kroku automat zpracovává právě jeden symbol ze vstupu (a ten symbol během zpracování zmizí, tedy vstup se v každém kroku zkracuje), dále v každém kroku podle předpisu měníme stav automatu.

Výpočet končí úspěchem pouze tehdy, když je posledním prvkem posloupnosti koncová konfigurace (tj. stav z množiny koncových stavů a celý vstup je přečtený). Pokud tomu tak není a nelze provést další krok (poslední konfigurace není koncová a ve funkci δ již není předpis, který bychom mohli použít), výpočet skončí neúspěchem.

Poznámka:

Uvědomme si, jaký je vlastně rozdíl mezi funkcí a relací – u funkce nás zajímá její výsledek (z určitého definičního oboru, často to bývá číslo, nebo konkrétně u δ-funkce označení stavu), kdežto u u relace (jinak řečeno vztahu) nás zajímá, zda daný prvek do relace patří či nikoliv (tedy výsledkem je hodnota z množiny $\{\text{true}, \text{false}\}$).

Než budeme pokračovat v definicích souvisejících s automaty, uvedeme pár pomocných definic, které možná známe z algebry:

Definice 3.5 (Reflexivita a tranzitivita relace)

Nechť R *je relace na množině* M. R *je reflexivní, jestliže pro každý prvek množiny* M *platí, že je v relaci se sebou samým, tj.* $\forall x \in M : \; x \; \mathrm{R} \; x$.

Relace R *je tranzitivní, jestliže pro jakékoliv tři prvky množiny* M *platí, že pokud je první v relaci s druhým a druhý s třetím, pak první prvek je v relaci s třetím:*

$\forall x, y, z \in M : \; x \; \mathrm{R} \; y \quad \wedge \quad y \; \mathrm{R} \; z \quad \Rightarrow \quad x \; \mathrm{R} \; z$

Pro účely důkazu (ale i další účely) zavedeme následující označení:

$$(q_i, \alpha) \vdash^k (q_j, \beta) \tag{3.2}$$

Toto označení znamená, že z konfigurace (q_i, α) se po provedení k přechodů uplatněním δ-funkce automatu přesuneme do konfigurace (q_j, β). Číslo k určující počet provedených přechodů je přirozené číslo včetně 0 (hodnota 0 znamená, že nebyl proveden žádný krok), tedy $k \in \mathcal{N}_0$.

Když do předchozí definice dosadíme „naši“ relaci přechodu mezi konfiguracemi, získáme toto tvrzení:

Věta 3.1 (Reflexivita a tranzitivita relace přechodu mezi konfiguracemi)

Relace $\vdash$ *přechodu mezi konfiguracemi v konečném automatu je reflexivní a tranzitivní.*

Důkaz: Nejdřív se zaměřme na *reflexivitu*. Při použití označení uvedeného ve vzorci (3.2) můžeme napsat:

$$(q_i, \alpha) \vdash^0 (q_i, \alpha)$$

(za k jsme dosadili číslo 0). Tento vztah je platný pro jakoukoliv konfiguraci v konečném automatu, poněvadž odpovídá situaci, kdy v určitém stavu automatu čekáme na provedení dalšího kroku, proto je relace přechodu mezi konfiguracemi reflexivní.

Nyní k *tranzitivitě*. Předpokládejme, že v (libovolném) konečném automatu existují tyto dvě posloupnosti konfigurací:

$$\begin{aligned} (q_i, \alpha) \quad &\vdash^k \quad (q_j, \beta) \\ (q_j, \beta) \quad &\vdash^m \quad (q_p, \gamma) \end{aligned}$$

Znamená to, že z konfigurace (q_i, α) se lze přesunout po k krocích do konfigurace (q_j, β) a z konfigurace (q_j, β) po m krocích do konfigurace (q_p, γ).

Pokud jsou tyto vztahy platné, pak v grafu automatu existuje cesta ze stavu q_i do stavu q_j a hrany (přechodů) na této cestě jsou ohodnoceny postupně prvními k symboly z řetězce α, přičemž po odstranění tohoto k-prvkového prefixu z řetězce α vznikne řetězec β. Podobně v grafu existuje cesta ze stavu q_j do stavu q_p s ohodnocením hran na cestě prvními m symboly řetězce β, a odstraněním tohoto m-prvkového prefixu vznikne řetězec γ.

Předpokládejme tedy, že oba vztahy z předpokladu platí zároveň:

$$(q_i, \alpha) \vdash^k (q_j, \beta) \quad \wedge \quad (q_j, \beta) \vdash^m (q_p, \gamma)$$

tedy existuje cesta v grafu automatu ze stavu q_i do stavu q_j podle prefixu slova α a zároveň cesta ze stavu q_j do stavu q_p podle prefixu slova β. To ale znamená, že v grafu existuje cesta ze stavu q_i do stavu q_p (přes stav q_j) s ohodnocením k-znakového prefixu slova α a následně m-prvkového prefixu slova β (což je vlastně $k+m$-znakový prefix slova α – zamyslete se nad důvodem). Proto můžeme napsat

$$(q_i, \alpha) \vdash^k (q_j, \beta) \quad \wedge \quad (q_j, \beta) \vdash^m (q_p, \gamma) \quad \Rightarrow \quad (q_i, \alpha) \vdash^{k+m} (q_p, \gamma)$$

a relace přechodu mezi konfiguracemi je tranzitivní. □

Definice 3.6 (Reflexivní a tranzitivní uzávěr relace přechodu mezi konfiguracemi)

Reflexivní a tranzitivní uzávěr relace $\vdash$ přechodu mezi konfiguracemi konečného automatu označujeme $\vdash^*$ a rozumíme pod ním jakýkoliv počet (i 0) opakování uplatnění relace přechodu mezi konfiguracemi.

Tranzitivní uzávěr relace $\vdash$ označujeme $\vdash^+$ a rozumíme pod ním jakýkoliv počet opakování uplatnění relace přechodu mezi konfiguracemi, nejméně jeden.

Zejména reflexivní a tranzitivní uzávěr relace přechodu $\vdash^*$ budeme často používat v případě, že budeme chtít u konečného automatu „zkrátit“ zápis dlouhé posloupnosti přechodů mezi konfiguracemi. Uplatní se i v následujících definicích.

Definice 3.7 (Rozpoznání (přijímání) slova konečným automatem)

Konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ rozpoznává (přijímá) slovo w, pokud existuje posloupnost výpočtu tohoto slova v automatu, tedy pokud se lze z počáteční konfigurace (q_0, w_0) postupným uplatňováním relací přechodu dostat do některé koncové konfigurace:

$$(q_0, w) \ \vdash^* \ (q_f, \varepsilon), \quad \textit{kde } q_f \in F$$

Definice 3.8 (Jazyk konečného automatu)

Jazyk konečného automatu $\mathcal{A}$ je množina všech slov w, která automat přijímá:

$$L(\mathcal{A}) = \{w \in \Sigma^* \ ; \ (q_0, w) \ \vdash^* \ (q_f, \varepsilon), \ \textit{kde } q_f \in F\} \tag{3.3}$$

Automat $\mathcal{A}$ rozpoznává jazyk L_j, pokud přijímá právě *slova jazyka L_j (tj. přijímá všechna slova jazyka, ale nepřijímá žádné slovo do jazyka nepatřící).*

Značíme $L_j = L(\mathcal{A})$ (jazyk L_j je rozpoznáván automatem $\mathcal{A}$, je jeho jazykem).

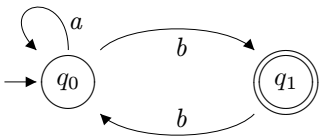
Konečný automat můžeme vyjádřit třemi různými způsoby:

- pomocí diagramu: tento způsob je pro automaty s několika stavy nejnázornější,
- pomocí δ-funkce: tento způsob je matematicky nejvhodnější, ale jeho názornost je poněkud nižší,
- tabulkou přechodů (přechodovou tabulkou): přehledný, zejména tehdy, když je jen málo signálů, na které automat reaguje.

Příklad 3.2

Ukážeme si všechny možné zápisy konečného automatu. Základní specifikace je následující:
$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \delta,\ q_0,\ \{q_1\})$

Diagram:



δ-funkce:

$\delta(q_0, a) = q_0$
$\delta(q_0, b) = q_1$
$\delta(q_1, b) = q_0$

Tabulka přechodů:

stav\vstup	a	b
$\rightarrow q_0$	q_0	q_1
$\leftarrow q_1$	-	q_0

V *reprezentaci pomocí diagramu* značíme stavy kružnicemi nebo elipsami, přechody orientovanými cestami v tomto grafu. Přímo vidíme, kam který přechod vede, také se v menším automatu lépe hledají cykly.

Reprezentace pomocí δ-funkce má jedno specifikum – narozdíl od ostatních reprezentací není přímo ze zápisu funkce jasné, který stav je počáteční a které stavy jsou koncové. Proto je nedílnou součástí zápisu specifikace $\mathcal{A} = (\{q_0, q_1\},\ \ldots$, kterou máme zapsánu na začátku tohoto příkladu.

Reprezentace pomocí přechodové tabulky se používá nejen pro svou přehlednost i v případě rozsáhlejších automatů, ale je praktická také v některých algoritmech. Jak vidíme, je třeba šipkami poznačit, který stav je počáteční (šipka vedoucí „k automatu“ $\rightarrow$) a které stavy jsou koncové (šipka vedoucí „od automatu“ $\leftarrow$).

Jeden z možných výpočtů v automatu:
$(q_0, aab) \vdash (q_0, ab) \vdash (q_0, b) \vdash (q_1, \varepsilon)$ proto $aab \in L(\mathcal{A})$

Předchozí výpočet byl úspěšný, jednalo se o slovo patřící do jazyka rozpoznávaného automatem. Nicméně automat může mít na vstupu i slova nepatřící do jeho jazyka, například:

- $(q_0, bba) \vdash (q_1, ba) \vdash (q_0, a) \vdash (q_0, \varepsilon)$
  tato konfigurace není koncová: $q_0 \notin F$, proto $bba \notin L(\mathcal{A})$
- $(q_0, ba) \vdash (q_1, a) \vdash$???
  dál nelze pokračovat (není vhodný předpis v δ-funkci, ve stavu q_1 nelze reagovat na symbol a), ale konfigurace není koncová, slovo nebylo zpracováno, proto $ba \notin L(\mathcal{A})$

Jazyk automatu:
$L(\mathcal{A}) = \{a^n b\ ;\ n \geq 0\} \cdot \{(ba^*b)^i\ ;\ i \geq 0\} \quad = \quad a^*b(ba^*b)^*$

Zbývá poslední definice této sekce, která nám umožní stanovit, kdy jsou dva (obecně různě definované) konečné automaty ekvivalentní.

Definice 3.9 (Ekvivalence automatů)

Konečné automaty $\mathcal{A}_1$ a $\mathcal{A}_2$ jsou (jazykově) ekvivalentní, pokud jsou shodné jazyky, které rozpoznávají: $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

Ekvivalentní automaty se tedy mohou lišit například v označení a počtu stavů, dokonce mohou mít odlišnou δ-funkci. Důležité je jen to, že generují naprosto stejné množiny slov.

3.2 Nedeterminismus

V základní definici konečného automatu existuje pro každé slovo přijímané automatem *právě jedna cesta* v diagramu automatu, a tedy výpočet je vždy jednoznačný (tomu říkáme *deterministický*). Výhodou tohoto postupu je, že takto vytvořený konečný automat se snadněji programuje, protože v klasickém programování je jednoznačnost nutnou podmínkou.

Někdy je však jednodušší vytvořit automat, který tuto vlastnost nemá, tedy pro některá přijímaná slova může existovat více různých cest v diagramu. Zde si takový automat definujeme a ukážeme si také způsob převedení na původní formu.

Definice 3.10 (Nedeterministický konečný automat)

Nedeterministický konečný automat (NKA) je takový konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$*, kde* $\delta : \ Q \times \Sigma \to \mathcal{P}(Q)$*. Ostatní součásti definice jsou stejné jako u (deterministického) konečného automatu.*

$\mathcal{P}(Q)$ je potenční množina množiny Q, je to množina všech jejích podmnožin (včetně prázdné množiny a také samotné množiny Q). Můžeme se také setkat se zápisem 2^Q, ten znamená totéž.

V některých stavech na určitý signál může existovat *více než jedna možnost* jak reagovat, dokonce pro některá slova může v grafu automatu existovat více různých cest od počátečního stavu do některého koncového.

V *deterministickém automatu* (DKA) pro jedno slovo existuje právě jedna cesta v grafu (je automatem rozpoznáváno) nebo žádná cesta (slovo není v jazyce automatu).

Definice 3.11 (Jazyk rozpoznávaný NKA)

Jazyk rozpoznávaný nedeterministickým konečným automatem je

$$L(\mathcal{A}) = \{w \in \Sigma^* \ ; \ (q_0, w) \ \vdash^* \ (q_f, \varepsilon), \ q_f \in F\} \tag{3.4}$$

Poznámka:

Jazyk rozpoznávaný nedeterministiským konečným automatem je tedy množina všech slov nad abecedou Σ, pro která existuje alespoň jeden výpočet (cesta v grafu automatu) od počátečního do kteréhokoliv koncového stavu.

Definice vypadá prakticky stejně jako u deterministického automatu, rozdíl je „uvnitř“, v definici δ-funkce.

Věta 3.2

Nechť $\mathcal{A}$ *je nedeterministický konečný automat. Potom existuje deterministický konečný automat* $\mathcal{A}'$ *takový, že* $L(\mathcal{A}) = L(\mathcal{A}')$ *(tj. rozpoznávají stejný jazyk).*

Než se pustíme do důkazu této věty, podíváme se na postup konstrukce a následně si konstrukci ukážeme na příkladech. Pak bude následovat důkaz této věty.

Postup konstrukce: Potřebujeme obecný postup (algoritmus) toho, jak pro jakýkoliv nedeterministický automat vytvořit ekvivalentní deterministický konečný automat. Označme:
$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ nedeterministický (ten máme)
$\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ deterministický (ten chceme vytvořit)

Stavy nového automatu budou odpovídat množinám stavů původního. Pro každou rovnost $\delta(q_i, a) = \{q_j, q_k\}$ stavy (množiny) $\{q_i\}, \{q_j, q_k\}$ budou patřit ke stavům nového automatu.
Postup:

- $Q' = \{M \ ; \ M \subseteq Q\} = \mathcal{P}(Q) = 2^Q$ – nová množina stavů bude množinou všech podmnožin původní množiny stavů,
- $q_0' = \{q_0\}$ – počáteční stav je jednoprvková množina obsahující původní počáteční stav,
- $M \in F' \ \Leftrightarrow \ M \cap F \neq \emptyset$ – koncové stavy jsou všechny, které (coby množiny) obsahují alespoň jeden původní koncový stav,
- $\delta'(M, a) = \{q \ ; \ q \in \delta(p, a), \ p \in M\} = \bigcup_{q \in M} \delta(q, a)$ – sjednotíme všechny možné reakce v daném stavu na daný signál do jediné množiny, ta bude novým stavem. □

Příklad 3.3

Nejdřív si ujasníme, jak vlastně budou vypadat stavy nově vytvářeného automatu. Pokud původní automat $\mathcal{A}$ má množinu stavů $Q = \{X, Y, Z\}$, pak množina stavů ekvivalentního deterministického automatu je tato:

$$\Big\{ \ \emptyset, \ \{X\}, \ \{Y\}, \ \{Z\}, \ \{X, Y\}, \ \{X, Z\}, \ \{Y, Z\}, \ \{X, Y, Z\} \ \Big\}$$

Pokud například máme v původním automatu předpis $\delta(X, a) = \{X, Z\}$ (tj. ze stavu X na signál a se lze přesunout do stavu Z nebo zůstat ve stavu X), pak v novém automatu vytvoříme předpis $\delta'(\{X\}, a) = \{X, Z\}$, který již je plně deterministický.

Ovšem nestačí pouze podle existujících předpisů vytvořit obdobné s přidáním množinových závorek (nic jiného jsme vlastně zatím neudělali), ještě zbývá přidat další předpisy δ-funkce. Všimněte si, že zde by například neexistoval předpis pro to, jak se chovat ve stavu $\{X, Z\}$. Tento stav ve výsledném automatu rozhodně existuje (dokonce do něj vede přechod ze stavu $\{X\}$), ale jaké přechody vedou z tohoto stavu?

Pokud se jedná o množinu (nový stav) $\{X, Z\}$, zřejmě budou vést nějaké přechody ze stavu X a ze stavu Z. Pokud v původním automatu bylo možné určitým způsobem pokračovat z některého z těchto dvou stavů, bude možné stejným způsobem pokračovat i ze stavu $\{X, Z\}$. Tento stav vznikl sjednocením stavů X a Z, tedy provedeme operaci sjednocení i na přechodech vedoucích z těchto původních stavů. Například jestliže v původním automatu existoval předpis $\delta(X, b) = \{Y\}$ a $\delta(Z, b) = \{Y, Z\}$ (a žádné jiné předpisy pro tyto stavy a signál b už neexistovaly), vytvoříme v novém automatu přechod $\delta'(\{X, Z\}, b) = \{Y\} \cup \{Y, Z\} = \{Y, Z\}$.

Takto budeme postupovat pro všechny přechody v původním automatu, tedy nám stačí ovládat především operaci sjednocení množin.

Pokud pracujeme s reprezentací δ-funkce ve tvaru tabulky, můžeme jednoduše postupovat tak, že v tabulce původního automatu „uzávorkujeme“ ohodnocení řádků a buňky do množinových závorek a pak pro každou množinu z buněk, kterou není ohodnocen žádný řádek, přidáme řádek tabulky a doplníme obsah buněk na daném řádku.

Jak určit obsah nové buňky: pokud je řádek ohodnocen množinou $\{q_i, q_j, \ldots\}$, pak do buňky sepíšeme obsah buněk na řádcích $\{q_i\}$, $\{q_j\}$, ... v daném sloupci, tedy vlastně sjednocujeme řádky jednotlivých prvků množiny.

Příklad 3.4

$L = (a+b)^*bb \;=\; \{\{a,b\}^n bb \;;\; n \geq 0\}$

$\mathcal{A} = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$
$\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$

Původní nedeterministický KA:

$\mathcal{A}$	a	b
$\rightarrow q_0$	q_0	q_0, q_1
q_1	-	q_2
$\leftarrow q_2$	-	-

Ekvivalentní deterministický KA:

$\mathcal{A}'$	a	b
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\leftarrow \{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$\leftarrow \{q_0, q_2\}$	$\{q_0\} \cup \emptyset$	$\{q_0, q_1\} \cup \emptyset$
$\leftarrow \{q_1, q_2\}$	$\emptyset \cup \emptyset$	$\{q_2\} \cup \emptyset$
$\leftarrow \{q_0, q_1, q_2\}$	$\{q_0\} \cup \emptyset \cup \emptyset$	$\{q_0, q_1\} \cup \{q_2\} \cup \emptyset$

Sjednotíme, odstraníme nepotřebné stavy:

$\mathcal{A}'$	a	b
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$\leftarrow \{q_0, q_1, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$

První tři řádky tabulky přechodů jsme jednoduše přejali s doplněním množinových závorek, příp. symbolů prázdné množiny pro buňky, které původně byly prázdné. Další řádky jsou již kombinací původních stavů o různém počtu prvků, přičemž symbol prázdné množiny v tomto případě nebudeme řadit do množiny stavů vytvářeného automatu. Obsah buněk nových řádků vytvoříme uplatněním operace sjednocení.

Protože množina stavů je zbytečně rozsáhlá, provedli jsme odstranění nepotřebných stavů. Později si ukážeme algoritmus, zde nám postačí vědět, že vyřadit lze stavy, do kterých nevede cesta z počátečního stavu, a stavy, ze kterých nevede cesta do žádného koncového stavu. Můžeme to provést, protože vyřazené stavy by se nijak neprojevily na jazyce generovaném automatem.

Množina stavů automatu $\mathcal{A}'$ je po odstranění nepotřebných stavů:
$Q' = \Big\{\{q_0\},\ \{q_0, q_1\},\ \{q_0, q_1, q_2\}\Big\}$, je tříprvková.

Množina koncových stavů: $F' = \Big\{\{q_0, q_1, q_2\}\Big\}$, je pouze jednoprvková.

Důkaz (Věta 3.2): Nechť $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je nedeterministický konečný automat. Dle postupu uvedeného v konstrukci za dokazovanou větou jsme sestrojili deterministický automat $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$. Nyní dokážeme, že automaty $\mathcal{A}$ a $\mathcal{A}'$ jsou ekvivalentní, tedy že platí $L(\mathcal{A}) = L(\mathcal{A}')$, neboli: $\forall w \in \Sigma^* : \ w \in L(\mathcal{A}) \Leftrightarrow w \in L(\mathcal{A}')$.

Použijeme důkaz matematickou indukcí podle délky rozpoznávaného slova w, resp. podle počtu kroků zpracování slova (to je u konečných automatů totéž).

Báze: Prověříme případ $|w| = 0$, tj. $w = \varepsilon$. Jsou dvě možnosti: $\varepsilon \in L(\mathcal{A})$ nebo $\varepsilon \notin L(\mathcal{A})$.

- $\varepsilon \in L(\mathcal{A}) \quad \Rightarrow \quad q_0 \in F \quad \Rightarrow \quad \{q_0\} \in F' \quad \Rightarrow \quad \varepsilon \in L(\mathcal{A}')$
- $\varepsilon \notin L(\mathcal{A}) \quad \Rightarrow \quad q_0 \notin F \quad \Rightarrow \quad \{q_0\} \notin F' \quad \Rightarrow \quad \varepsilon \notin L(\mathcal{A}')$

Tedy pro $|w| = 0$ věta platí:
$(q_0, \varepsilon) \vdash^0 (q_f, \varepsilon),$ kde $q_f \in F \quad \Leftrightarrow \quad (\{q_0\}, \varepsilon) \vdash^0 (M_f, \varepsilon),$ kde $M_f \in F'$

Předpoklad: Předpokládejme, že věta platí pro $w \in \Sigma^*$ takové, že $|w| = k$.

Krok indukce: Použijeme $w \in \Sigma^*$ takové, že $w = va, \ |w| = k + 1, \ |v| = k, \ a \in \Sigma$.
Pokud $w \in L(\mathcal{A})$, pak $(q_0, va) \vdash^k (q_m, a) \vdash (q_f, \varepsilon),$ kde $q_m \in Q, \ q_f \in F$
$\Rightarrow$ v automatu $\mathcal{A}' : \ (\{q_0\}, va) \vdash^k (M_m, a) \quad \wedge \quad q_m \in M_m,$ kde $M_m \in Q'$
$\Rightarrow \ \exists\, \delta'(M_m, a) = P,$ kde $q_f \in P,$ proto $P \in F'$
$\Rightarrow$ v automatu $\mathcal{A}' : \ (\{q_0\}, va) \vdash^k (M_m, a) \vdash (P, \varepsilon),$ kde $M_m \in Q', \ P \in F'$
Pokud $w \notin L(\mathcal{A})$, pak neexistuje posloupnost konfigurací v automatu $\mathcal{A}$, která by byla výpočtem slova w. Protože funkce δ' obsahuje pouze sjednocení přechodů realizovatelných funkcí δ, také v automatu $\mathcal{A}'$ neexistuje posloupnost konfigurací, která by byla výpočtem slova w.

Z toho vyplývá, že $w \in L(\mathcal{A}) \ \Leftrightarrow \ w \in L(\mathcal{A}')$ pro kterékoliv slovo $w \in \Sigma^*$. □

Věta 3.3

Množiny jazyků generovaných deterministickými a nedeterministickými konečnými automaty jsou navzájem ekvivalentní.

Důkaz: V předchozí větě jsme dokázali vztah NKA $\subseteq$ DKA, tedy že pro každý nedeterministický automat dokážeme sestrojit ekvivalentní deterministický. Tedy stačí dokázat i opačný vztah – DKA $\subseteq$ NKA, pak dokážeme, že obě množiny jsou ekvivalentní.

Tento důkaz je však triviální, stačí si uvědomit, že deterministické konečné automaty jsou vlastně speciálním případem nedeterministických, kde je ve výsledku δ-funkce vždy jednoprvková množina. Tedy platí:
Pokud $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je deterministický automat, ekvivalentním nedeterministickým automatem bude $\mathcal{A}' = (Q, \Sigma, \delta', q_0, F)$, kde je
$\delta'(q, a) = \{r\}$ pro každý předpis $\delta(q, a) = r, \quad q, r \in Q, \ a \in \Sigma$ (jen uzávorkujeme výsledek).

Tím jsme ukázali, že množiny jazyků generovaných deterministickými a nedeterministickými konečnými automaty jsou ekvivalentní. □

Poznámka:

Důsledkem předchozí věty je, že pokud se nám nedaří navrhnout deterministický konečný automat, můžeme navrhnout nedeterministický a uvedeným postupem ho převést na deterministický.

Příklad 3.5

Sestavíme konečný automat rozpoznávající jazyk $L = \{if, then, this\}$. Je to konečný jazyk nad abecedou $\Sigma = \{i, f, t, h, e, n, s\}$ obsahující tři slova. Budeme chtít, aby pro každé z těchto slov existoval zvláštní koncový stav (tj. množina F bude tříprvková). Sestrojíme nejdřív nedeterministický konečný automat a pro názornost použijeme všechny tři možnosti vyjádření:

$\mathcal{A} = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$

		i	f	t	h	e	n	s
$\rightarrow$	q_0	q_1		q_2, q_5				
	q_1		K_1					
	q_2				q_3			
	q_3					q_4		
	q_4						K_2	
	q_5				q_6			
	q_6	q_7						
	q_7							K_3
$\leftarrow$	K_1							
$\leftarrow$	K_2							
$\leftarrow$	K_3							

$Q = \{q_0, q_1, \ldots, q_7, K_1, K_2, K_3\}$

$F = \{K_1, K_2, K_3\}$

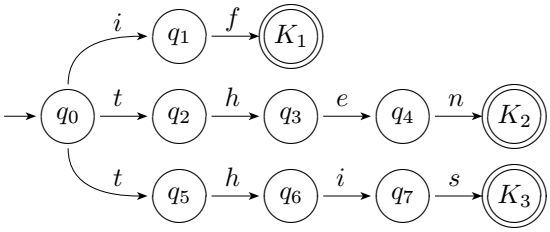
$\delta(q_0, i) = \{q_1\}$ $\quad$ $\delta(q_4, n) = \{K_2\}$

$\delta(q_0, t) = \{q_2, q_5\}$ $\quad$ $\delta(q_5, h) = \{q_6\}$

$\delta(q_1, f) = \{K_1\}$ $\quad$ $\delta(q_6, i) = \{q_7\}$

$\delta(q_2, h) = \{q_3\}$ $\quad$ $\delta(q_7, s) = \{K_3\}$

$\delta(q_3, e) = \{q_4\}$



Jak vidíme, automat je nedeterministický (to poznáme třeba podle tabulky, tam jsou v buňce na řádku q_0 v sloupci t dva prvky). Podle δ-funkce to poznáme také snadno – ve výsledku pro předpis $\delta(q_0, t)$ je dvouprvková množina. U diagramu to možná není viditelné na první pohled, ale stačí si všimnout, ze ze stavu q_0 vycházejí dva přechody se stejným označením.

Převod nedeterministického KA na ekvivalentní deterministický bude probíhat takto:

- doplníme množinové závorky,
- vznikl nový stav – $\{q_2, q_5\}$, pro tento stav potřebujeme v tabulce nový řádek (resp. předpis v δ-funkci a kružnici v diagramu) včetně doplnění reakcí na různé signály v tomto stavu,
- pokud splněním požadavku v předchozí odrážce tohoto seznamu vznikl další nový stav, provedeme pro něj totéž co pro $\{q_2, q_5\}$ (nový stav a reakce na signály), což bude probíhat tak dlouho, dokud budou vznikat další „kombinované“ stavy.

Správně bychom měli předem vytvořit množiny se všemi možnými kombinacemi o různém počtu prvků z původní množiny stavů, čímž bychom vytvořili úplnou množinu stavů nového automatu, ale „osekáním“ postupu do výše naznačené podoby si zjednodušíme práci, protože nebudeme zbytečně vytvářet stavy, do kterých nevede cesta z počátečního stavu (tj. nedostupné). Je zřejmé, že když se stav objeví v některé buňce tabulky přechodů, je pravděpodobně dostupný (ale ne nutně), ovšem pokud se v žádné buňce neobjeví, je nutně nedostupný a nemá smysl se jím zabývat.

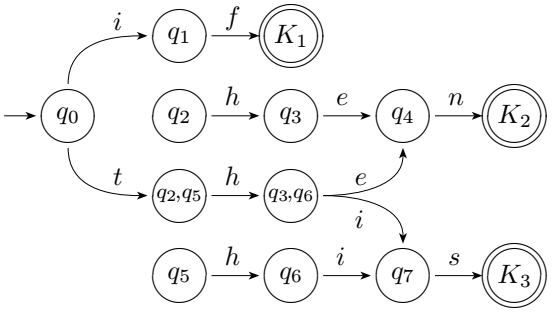
Tedy přidáme nový stav $\{q_2, q_5\}$ a další „navázané“ stavy:

$\mathcal{A}' = (Q',\ \Sigma,\ \delta',\ \{q_0\},\ F')$

	i	f	t	h	e	n	s
$\rightarrow\ \{q_0\}$	$\{q_1\}$		$\{q_2, q_5\}$				
$\{q_1\}$		$\{K_1\}$					
$\{q_2\}$				$\{q_3\}$			
$\{q_3\}$					$\{q_4\}$		
$\{q_4\}$						$\{K_2\}$	
$\{q_5\}$				$\{q_6\}$			
$\{q_6\}$	$\{q_7\}$						
$\{q_7\}$							$\{K_3\}$
$\{q_2, q_5\}$				$\{q_3, q_6\}$			
$\{q_3, q_6\}$	$\{q_7\}$				$\{q_4\}$		
$\leftarrow\ \{K_1\}$							
$\leftarrow\ \{K_2\}$							
$\leftarrow\ \{K_3\}$							

$\delta'(\{q_0\}, i) = \{q_1\}$
$\delta'(\{q_0\}, t) = \{q_2, q_5\}$
$\delta'(\{q_1\}, f) = \{K_1\}$
$\delta'(\{q_2\}, h) = \{q_3\}$
$\delta'(\{q_3\}, e) = \{q_4\}$
$\delta'(\{q_4\}, n) = \{K_2\}$
$\delta'(\{q_5\}, h) = \{q_6\}$
$\delta'(\{q_6\}, i) = \{q_7\}$
$\delta'(\{q_7\}, s) = \{K_3\}$
$\delta'(\{q_2, q_5\}, h) = \{q_3, q_6\}$
$\delta'(\{q_3, q_6\}, i) = \{q_7\}$
$\delta'(\{q_3, q_6\}, e) = \{q_4\}$

$$Q' = \Big\{\{q_0\}, \{q_1\}, \ldots, \{q_7\}, \{K_1\}, \{K_2\}, \{K_3\}, \{q_2, q_5\}, \{q_3, q_6\}\Big\}$$
$$F' = \Big\{\{K_1\}, \{K_2\}, \{K_3\}\Big\}$$



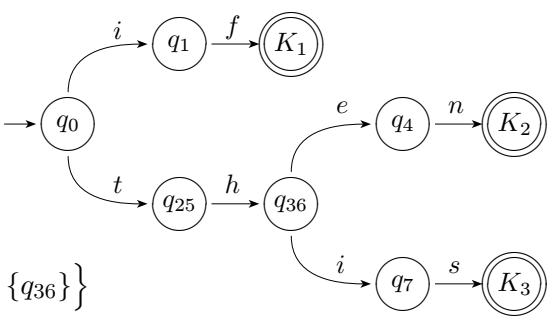
Především na diagramu vidíme, že stavy $\{q_2\}, \{q_3\}, \{q_5\}$ a $\{q_6\}$ se staly nedostupnými (nevede *do nich* žádná cesta z počátečního stavu), tyto stavy tedy můžeme odstranit.

Pokud bychom použili „složitější“ postup (vytvořili bychom množinu stavů jako potenční množinu obsahující všechny možné kombinace původních stavů), museli bychom odstraňovat také nadbytečné stavy (takové, *ze kterých* nevede cesta do žádného koncového stavu).

Aby byl výsledný automat přehlednější, můžeme nově vzniklé „velké“ stavy přeznačit, například místo $\{q_2, q_5\}$ budeme psát jen $\{q_{25}\}$. Dalším zjednodušením zápisu by pak bylo odstranění závorek, které po naznačené úpravě už nebudou potřeba, nicméně to už necháme na čtenáři. Výsledný deterministický automat bez nedostupných stavů:

$\mathcal{A}'' = (Q'',\ \Sigma,\ \delta'',\ \{q_0\},\ F')$

$\delta''(\{q_0\}, i) = \{q_1\}$
$\delta''(\{q_0\}, t) = \{q_{25}\}$
$\delta''(\{q_1\}, f) = \{K_1\}$
$\delta''(\{q_4\}, n) = \{K_2\}$
$\delta''(\{q_7\}, s) = \{K_3\}$
$\delta''(\{q_{25}\}, h) = \{q_{36}\}$
$\delta''(\{q_{36}\}, i) = \{q_7\}$
$\delta''(\{q_{36}\}, e) = \{q_4\}$



$$Q'' = \Big\{\{q_0\}, \{q_1\}, \{q_4\}, \{q_7\}, \{K_1\}, \{K_2\}, \{K_3\}, \{q_{25}\}, \{q_{36}\}\Big\}$$
$$F' = \Big\{\{K_1\}, \{K_2\}, \{K_3\}\Big\}$$

	i	f	t	h	e	n	s
$\rightarrow \{q_0\}$	$\{q_1\}$		$\{q_{25}\}$				
$\{q_1\}$		$\{K_1\}$					
$\{q_4\}$						$\{K_2\}$	
$\{q_7\}$							$\{K_3\}$
$\{q_{25}\}$				$\{q_{36}\}$			
$\{q_{36}\}$	$\{q_7\}$				$\{q_4\}$		
$\leftarrow \{K_1\}$							
$\leftarrow \{K_2\}$							
$\leftarrow \{K_3\}$							

Výsledný deterministický automat má o dva stavy méně než původní nedeterministický, a do koncových stavů jsme v tomto případě nezasahovali.

Podle předchozího příkladu můžeme usoudit, že nedeterminismus často vzniká tak, že více slov jazyka začíná stejným podřetězcem (tj. má stejný prefix). Tento typ nedeterminismu se řeší velice jednoduše – začátek cesty v grafu takovýchto slov jednoduše shrneme (sjednotíme). Ovšem nedeterminismus může být i v něčem úplně jiném, pak nezbývá než se spolehnout na algoritmus.

3.3 Totální automat

Obvykle není nutné, aby automat dokázal v každém stavu reagovat na jakýkoliv signál, ale za určitých okolností se tato vlastnost může hodit. Představme si třeba situaci, kdy programátor chce napsat program dostatečně robustní, který by dokázal reagovat na jakýkoliv vstup, v případě chybného vstupu chybovým hlášením (tedy přechodem do chybového stavu s patřičným ošetřením).

Totální (úplný) automat je takový automat, který v každém stavu *jednoznačně* reaguje na *jakýkoliv* signál. Předně si uvědomme, že takový automat nutně musí být deterministický. Kdyby nebyl, pak by nemohl jednoznačně reagovat a o nějakém programování bychom vůbec nemohli uvažovat (pouze to, co je deterministické nebo lze reprezentovat jakýmkoliv deterministickým způsobem, je naprogramovatelné). Pokud náš automat již tuto vlastnost má, není problém provést *zúplnění*.

Definice 3.12 (Totální automat)

Totální (úplný) konečný automat je deterministický konečný automat, ve kterém lze ve všech stavech reagovat na kterýkoliv symbol (signál) abecedy, tj. přechodová funkce δ je totální*:*

$$\forall\,(q,a) \in (Q \times \Sigma)\ \ \exists!\, p \in Q:\ \ \delta(q,a) = p \tag{3.5}$$

Značení $\exists!$ znamená „existuje právě jeden“. Celý vzorec (3.5) nám říká, že ke každé uspořádané dvojici stavu a signálu existuje právě jeden stav, který získáme uplatněním δ-funkce na tuto uspořádanou dvojici.

Věta 3.4

Ke každému (deterministickému) konečnému automatu $\mathcal{A}$ existuje totální automat $\mathcal{A}'$ takový, že $L(\mathcal{A}) = L(\mathcal{A}')$.

Postup konstrukce: K danému konečnému automatu sestrojíme ekvivalentní totální (úplný) automat takto:

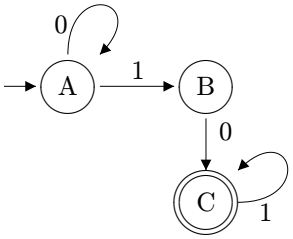
- převedeme automat na deterministický,
- vytvoříme nový stav $\emptyset$, který bude fungovat jako „odpadkový koš“ (tento stav můžeme pojmenovat i jinak, označení $\emptyset$ je spíše tradice),
- pokud z některého stavu nevede žádný přechod označený signálem $a \in \Sigma$, pak přidáme takovýto přechod (vedoucí do stavu $\emptyset$),
- přidáme smyčku (přechod začínající a končící ve stejném stavu) u stavu $\emptyset$ pro každý symbol abecedy. □

Příklad 3.6

Je dán deterministický konečný automat $\mathcal{A} = (\{A, B, C\},\ \{0, 1\},\ \delta,\ A,\ \{C\})$ (se třemi stavy, z nichž jeden je koncový), s těmito parametry:

$\delta(A, 0) = A$
$\delta(A, 1) = B$
$\delta(B, 0) = C$
$\delta(C, 1) = C$

$\mathcal{A}$	0	1
→ A	A	B
B	C	-
← C	-	C

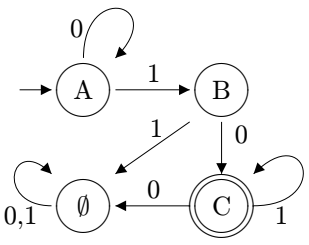


Především na tabulce přechodů je na první pohled zřejmé, že tento automat není úplný (totální) – ve stavu B nelze reagovat na signál 1, ve stavu C na signál 0. Zúplnění provedeme takto:

- úplný automat označíme $\mathcal{A}' = (Q',\ \{0, 1\},\ \delta',\ A,\ \{C\})$, změna tedy bude jen v množině stavů a δ-funkci,
- vytvoříme nový stav $\emptyset$, proto $Q' = Q \cup \{\emptyset\} = \{A, B, C, \emptyset\}$,
- do δ' zkopírujeme všechny přechody z δ a navíc přidáme přechody $\delta'(B, 1) = \emptyset$ a $\delta'(C, 0) = \emptyset$,
- přidáme další nové přechody $\delta'(\emptyset, 0) = \emptyset$ a $\delta'(\emptyset, 1) = \emptyset$, protože i v nově přidaném stavu je třeba reagovat na všechny signály abecedy.

$\delta'(A, 0) = A$ $\quad$ $\delta'(C, 0) = \emptyset$
$\delta'(A, 1) = B$ $\quad$ $\delta'(C, 1) = C$
$\delta'(B, 0) = C$ $\quad$ $\delta'(\emptyset, 0) = \emptyset$
$\delta'(B, 1) = \emptyset$ $\quad$ $\delta'(\emptyset, 1) = \emptyset$

$\mathcal{A}'$	0	1
→ A	A	B
B	C	$\emptyset$
← C	$\emptyset$	C
$\emptyset$	$\emptyset$	$\emptyset$



Důkaz (Věta 3.4): To, že vytvořený automat $\mathcal{A}'$ je úplný, je zřejmé. V každém stavu reaguje na kterýkoliv symbol abecedy, čehož jsme docílili přidáním stavu $\emptyset$ a příslušných nových přechodů.

Je třeba dokázat, že pokud použijeme výše naznačený postup, pak původní (neúplný) a výsledný (zúplněný) automat budou ekvivalentní. Předpokládejme, že $\mathcal{A}$ je deterministický.

Nejdřív dokážeme vztah $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ a pak opačný vztah $L(\mathcal{A}) \supseteq L(\mathcal{A}')$, čímž bude dokázána ekvivalence těchto jazyků a tedy i automatů.

Dokazujeme $L(\mathcal{A}) \subseteq L(\mathcal{A}')$:
Vezměme jakékoliv slovo $w \in L(\mathcal{A})$.
$\Rightarrow$ existuje právě jeden výpočet slova w v $\mathcal{A}$:
$\mathcal{A}$: $(q_0, w) \vdash^* (q_f, \varepsilon), \quad q_f \in F$
$\Rightarrow$ všechny přechody z funkce δ existují také ve funkci δ' v automatu $\mathcal{A}'$:
$\mathcal{A}'$: $(q_0, w) \vdash^* (q_f, \varepsilon), \quad q_f \in F$
$\Rightarrow$ $w \in L(\mathcal{A}')$

Dokazujeme $L(\mathcal{A}) \supseteq L(\mathcal{A}')$:
Vezměme jakékoliv slovo $w \notin L(\mathcal{A}),\ w \in \Sigma^*$.
$\Rightarrow$ neexistuje žádný výpočet slova w v $\mathcal{A}$
$\wedge$ v automatu $\mathcal{A}'$ jsme sice přidali nové přechody, ale všechny vedou do stavu $\emptyset$
$\Rightarrow$ pro každé $w \notin L(\mathcal{A}),\ w \in \Sigma^*$ existuje následující posloupnost přechodů:
$\mathcal{A}'$: $(q_0, w) \vdash^* (r, a\beta) \vdash (\emptyset, \beta),\ r \in Q,\ a \in \Sigma,\ \beta \in \Sigma^*,$ použito $\delta'(r, a) = \emptyset$
$\Rightarrow$ výpočet může pokračovat jediným možným způsobem:
$\mathcal{A}'$: $(q_0, w) \vdash^* (r, a\beta) \vdash (\emptyset, \beta) \vdash^* (\emptyset, \varepsilon),$ přičemž $(\emptyset, \varepsilon)$ není koncová konfigurace, $\emptyset \notin F$
$\Rightarrow$ $w \notin L(\mathcal{A}')$

Z toho vyplývá, že $w \in L(\mathcal{A}) \Leftrightarrow w \in L(\mathcal{A}')$, tedy původní a nově vytvořený automat jsou navzájem ekvivalentní a uvedený postup lze použít pro zúplnění konečného automatu. □

Převod konečného automatu na totální je potřeba v důkazech některých vět týkajících se konečných automatů, a také je to potřebný požadavek pro případ, že vytváříme konečný automat, který budeme převádět do programového kódu.

3.4 Redukce stavů konečného automatu

Na předchozích stránkách jsme zatím neformálně použili pojmy nedosažitelného a nadbytečného stavu konečného automatu a ukázali jsme si, jak je v jednodušších automatech poznat. Nyní tyto pojmy formálně definujeme a představíme si postupy (algoritmy) pro jejich nalezení.

Definice 3.13 (Nedosažitelný stav KA)

Nedosažitelný stav v konečném automatu $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *je stav* q *takový, že neexistuje žádná posloupnost přechodů*

$$(q_0, \alpha) \vdash^* (q, \beta), \quad \alpha, \beta \in \Sigma^* \tag{3.6}$$

Tedy stav q *není dosažitelný z počátečního stavu* q_0 *automatu* $\mathcal{A}$.

Definice 3.14 (Nadbytečný stav KA)

Nadbytečný stav v konečném automatu $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *je stav* q *takový, že neexistuje žádná posloupnost přechodů*

$$(q, \alpha) \vdash^* (q_f, \varepsilon), \quad q_f \in F, \ \alpha \in \Sigma^* \tag{3.7}$$

Tedy ze stavu q *nevede cesta do žádného koncového stavu automatu* $\mathcal{A}$.

Poznámka:

Všimněte si, že definice nedosažitelného a nadbytečného stavu automatu jsou *čistě syntaktické*, týkají se pouze a jenom struktury automatu – je naprosto jedno, na které signály se při přechodu mezi stavy reaguje, je důležité jen to, zda mezi určitými stavy vede či nevede cesta. V důkazech budeme tedy pracovat s obdobou grafových algoritmů, protože ve skutečnosti se jedná o důkaz, zda mezi dvěma uzly grafu (stavem q a buď počátečním nebo některým koncovým stavem) existuje či neexistuje cesta v grafu (diagramu automatu).

Definice 3.15 (Redukovaný automat)

Redukovaný konečný automat (automat s redukovanou množinou stavů) je automat bez nedosažitelných a nadbytečných stavů.

3.4.1 Odstranění nedosažitelných stavů

Vytvoříme množinu stavů, ke kterým je možné se dostat z počátečního stavu – postupujeme od počátečního stavu.

Věta 3.5

Ke každému deterministickému konečnému automatu $\mathcal{A}$ *lze sestrojit ekvivalentní konečný automat* $\mathcal{A}'$ *bez nedosažitelných stavů.*

Postup konstrukce: Vytvoříme množinu stavů, které *jsou dosažitelné* z počátečního stavu. Stavy, které do této množiny nezařadíme, odstraníme. Nechť tedy $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$.

- vytvoříme množinu S_0 obsahující počáteční stav automatu, $S_0 = \{q_0\}$,
- vytvoříme množinu S_1 takovou, že bude obsahovat to, co množina S_0 (tedy $\{q_0\}$), a dále všechny stavy, do kterých vede přechod ze stavů množiny S_0 (tj. stavy, do kterých vede přechod ze stavu q_0),
- postupně vytváříme množiny S_i tak, že do S_i zařadíme nejdřív obsah množiny S_{i-1} a pak přidáme všechny stavy, do kterých vede přechod z některého stavu z množiny S_{i-1},
- končíme, když už se nic nedá přidat, tedy $S_i = S_{i-1}$, výsledkem je nová množina stavů.

Podle tohoto postupu je S_i množinou všech stavů, do kterých vede cesta z počátečního stavu o délce maximálně i. Postupujeme iterativně podle následujícího vzorce:

$$S_0 = \{q_0\} \tag{3.8}$$

$$S_i = S_{i-1} \cup \{q \in Q \ ; \ \delta(p,a) \ni q, \text{ kde } p \in S_{i-1}, \ a \in \Sigma\}, \quad i \geq 1 \tag{3.9}$$

Položme $S_{def} = S_i$ pro i takové, že $S_i = S_{i-1}$. Vytváříme $\mathcal{A}' = (Q', \Sigma, \delta', q_0, F')$, kde

- $Q' = S_{def}, \quad F' = F - (Q - S_{def})$,
- $\delta'(p,a) = \delta(p,a) \ \forall p \in S_{def}, \ a \in \Sigma$.

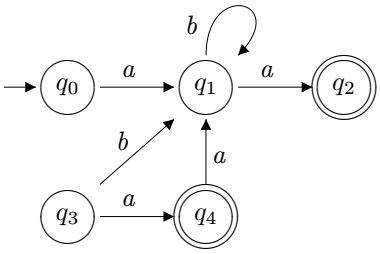
Co se změny v zápisu týče, záleží na způsobu reprezentace automatu. Pokud se jedná o tabulku přechodů, jednoduše odstraníme všechny řádky tabulky patřící stavům, které se ve vytvořené množině nenacházejí. Při reprezentaci pomocí δ-funkce odstraníme odpovídající části, kde nedosažitelné stavy jsou výchozím bodem některého přechodu. V diagramu odstraníme odpovídající kružnici s daným stavem a všechny přechody z ní vedoucí. □

Příklad 3.7

Odstraníme nepotřebné stavy tohoto deterministického konečného automatu:
$\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4\}, \ \{a,b\}, \ \delta, \ q_0, \ \{q_2, q_4\})$

$\delta(q_0,a) = q_1$
$\delta(q_1,a) = q_2$
$\delta(q_1,b) = q_1$
$\delta(q_3,a) = q_4$
$\delta(q_3,b) = q_1$
$\delta(q_4,a) = q_1$

$\mathcal{A}$	a	b
$\rightarrow q_0$	q_1	-
q_1	q_2	q_1
$\leftarrow q_2$	-	-
q_3	q_4	q_1
$\leftarrow q_4$	q_1	-



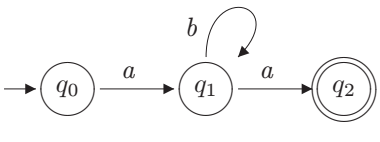
Iterativním postupem (podle délky cesty v grafu) vytvoříme množiny S_i:

- $S_0 = \{q_0\}$
  báze indukce; hledáme stavy dosažitelné z počátečního stavu, zde potřebujeme „začátek hledaných cest“
- $S_1 = \{q_0, q_1\}$
  ze stavu q_0 je možné se provedením jednoho kroku přesunout pouze do stavu q_1
- $S_2 = \{q_0, q_1, q_2\}$ ze stavů v množině S_1 je možné se provedením max. dvou kroků přesunout také do stavu q_2
- $S_3 = \{q_0, q_1, q_2\} \ = \ S_2$
  ze stavů v S_2 už do žádného dalšího stavu cesta nevede, končíme, $S_{def} = S_2 = S_3$

Prvky množiny S_{def} použijeme jako novou množinu stavů, vyřadíme stavy q_3, q_4:
$\mathcal{A}' = (\{q_0, q_1, q_2\}, \ \{a,b\}, \ \delta', \ q_0, \ \{q_2\})$

$\delta'(q_0,a) = q_1$
$\delta'(q_1,a) = q_2$
$\delta'(q_1,b) = q_1$

$\mathcal{A}'$	a	b
$\rightarrow q_0$	q_1	-
q_1	q_2	q_1
$\leftarrow q_2$	-	-



Vyřadili jsme dva stavy, diagram automatu se zpřehlednil a také se nám jednodušeji určuje jazyk rozpoznávaný automatem: $L(\mathcal{A}) = L(\mathcal{A}') = ab^*a$

Důkaz (Věta 3.5): Dokážeme, že výše popsaný algoritmus na daném automatu odstraňuje nedostupné stavy a že původní a nově vytvořený automat jsou ekvivalentní.

Co se týče funkčnosti odstraňování nedostupných stavů, důkaz je triviální – stačí použít matematickou indukci podle výše naznačeného způsobu konstrukce množiny S_{def} (tento typ důkazu se vždy nabízí, pokud dokazujeme platnost takto určeného postupu). Bází je množina obsahující pouze počáteční stav, ten je sám ze sebe samozřejmě dosažitelný. Krok indukce by pak zaručil, že do množiny S_{i+1} se navíc oproti množině S_i dostanou pouze stavy, do kterých se lze dostat jedním krokem z prvků množiny S_i, tedy platí:
$S_0 \subseteq S_1 \subseteq \ldots \subseteq S_i \subseteq S_{i+1} \subseteq \ldots \subseteq S_{def}$
(správně by mezi prvky měl být symbol $\subset$ a nikoliv $\subseteq$, protože následující množina je až na poslední dvojici vždy vlastní podmnožinou předchozí, ale vzhledem k tomu, že za určitých okolností už S_0 může být výsledkem, přidržíme se obecnějšího symbolu).

Je tento algoritmus konečný? Ano, protože množina Q je konečná, $S_{def} \subseteq Q$ a zároveň algoritmus končí tehdy, když „už není co přidat“, tedy je provedeno maximálně tolik kroků, kolik je stavů v Q.

Zbývá ukázat, že jazyk automatu zůstane nezměněn.

Dokazujeme $L(\mathcal{A}) \subseteq L(\mathcal{A}')$:
Vezměme jakékoliv slovo $w \in L(\mathcal{A})$.
$\Rightarrow$ existuje právě jeden výpočet slova w v $\mathcal{A}$:
$\mathcal{A}$: $(q_0, w) \vdash^* (q_f, \varepsilon), \quad q_f \in F,$
$\Rightarrow$ $\exists$ cesta v grafu $(q_0, \ldots, q_f)$
$\Rightarrow$ všechny uzly (stavy) na této cestě jsou dosažitelné z počátečního stavu, patří do S_{def}
(důkaz by bylo možné provést matematickou indukcí podle délky cesty)
cesta v grafu $(q_0, \ldots, q_f)$ existuje také v grafu pro automat $\mathcal{A}'$
$\mathcal{A}'$: $(q_0, w) \vdash^* (q_f, \varepsilon), \quad q_f \in F'$
$\Rightarrow$ $w \in L(\mathcal{A}')$

Dokazujeme $L(\mathcal{A}) \supseteq L(\mathcal{A}')$:
Vezměme jakékoliv slovo $w \notin L(\mathcal{A})$.
$\Rightarrow$ neexistuje výpočet slova w v $\mathcal{A}$, žádná posloupnost konfigurací a tedy ani cesta v grafu automatu $\mathcal{A}$ $(q_0, \ldots, q_f)$
$\wedge$ v δ-funkci $\mathcal{A}'$ jsme žádné nové přechody neumožnili (nepřidali)
$\Rightarrow$ $w \notin L(\mathcal{A}')$

Dokázali jsme, že popsaný postup odstranění nedostupných stavů je správný, korektní a úplný. □

3.4.2 Odstranění nadbytečných stavů

Pro redukci množiny stavů automatu je ještě třeba odstranit nadbytečné stavy, tedy stavy, ze kterých nevede v grafu automatu cesta do žádného koncového stavu.

Předpokládáme, že již bylo provedeno odstranění nedostupných stavů – to je důležité, protože se již nechceme zabývat stavy, ze kterých sice vede cesta do některého koncového stavu, ale jen do takového, který není dosažitelný z počátečního stavu (například v předchozím příkladu je stav q_4 koncový, ale není dosažitelný z počátečního stavu, a kdybychom předem neprovedli odstranění nedostupných stavů, algoritmus odstranění nadbytečných stavů bychom vpodstatě prováděli zbytečně).

Při odstraňování nadbytečných stavů budeme postupovat podobně jako u nedostupných, jen v opačném směru. Bází nám bude množina koncových stavů a v jednotlivých krocích budeme přidávat stavy, ze kterých lze jedním krokem dosáhnout některého ze stavů množiny dříve zařazených prvků.

Věta 3.6

Ke každému deterministickému konečnému automatu $\mathcal{A}$ bez nedosažitelných stavů lze sestrojit ekvivalentní automat $\mathcal{A}'$ bez nadbytečných stavů, tedy s redukovanou množinou stavů.

Postup konstrukce: Vytvoříme množinu stavů, ze kterých *je dosažitelný* některý koncový stav. Stavy, které do této množiny nezařadíme, odstraníme. Nechť tedy $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$.

- vytvoříme množinu E_0 obsahující koncové stavy automatu, $E_0 = F$,
- vytvoříme množinu E_1 takovou, že bude obsahovat všechny prvky množiny E_0 a dále všechny stavy, ze kterých vede přechod do některého stavu množiny E_0,
- postupně vytváříme množiny E_i tak, že do E_i zařadíme nejdřív obsah množiny E_{i-1} a pak přidáme všechny stavy, ze kterých vede přechod do některého stavu z množiny E_{i-1},
- končíme, když už se nic nedá přidat, tedy $E_i = E_{i-1}$, výsledkem je nová množina stavů.

Podle tohoto postupu je E_i množinou všech stavů, ze kterých vede cesta do některého koncového stavu o délce maximálně i. Postupujeme iterativně podle následujícího vzorce:

$$E_0 = F \tag{3.10}$$

$$E_i = E_{i-1} \cup \{q \in Q \ ; \ \delta(q,a) \ni p, \text{ kde } p \in E_{i-1}, \ a \in \Sigma\}, \quad i \geq 1 \tag{3.11}$$

Položme $E_{def} = E_i$ pro i takové, že $E_i = E_{i-1}$. Vytváříme $\mathcal{A}' = (Q', \Sigma, \delta', q_0, F)$, kde

- $Q' = E_{def}$,
- $\delta'(p,a) = \delta(p,a) = q$, kde $p, q \in E_{def}$, $a \in \Sigma$.

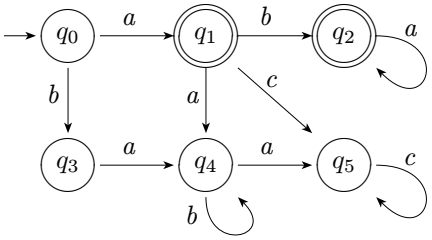
Oproti postupu pro odstranění nedostupných stavů si zde musíme dát větší pozor na to, aby „omylem“ nezůstal některý z přechodů, které mají být vyřazeny, zejména v tabulce přechodů. □

Příklad 3.8

Odstraníme nepotřebné stavy tohoto deterministického konečného automatu (nedosažitelné stavy již byly odstraněny):
$\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4, q_5\},\ \{a, b, c\},\ \delta,\ q_0,\ \{q_1, q_2\})$

$\delta(q_0, a) = q_1$ $\quad$ $\delta(q_2, a) = q_2$
$\delta(q_0, b) = q_3$ $\quad$ $\delta(q_3, a) = q_4$
$\delta(q_1, a) = q_4$ $\quad$ $\delta(q_4, a) = q_5$
$\delta(q_1, b) = q_2$ $\quad$ $\delta(q_4, b) = q_4$
$\delta(q_1, c) = q_5$ $\quad$ $\delta(q_5, c) = q_5$

	a	b	c
$\rightarrow q_0$	q_1	q_3	-
$\leftarrow q_1$	q_4	q_2	q_5
$\leftarrow q_2$	q_2	-	-
q_3	q_4	-	-
q_4	q_5	q_4	-
q_5	-	-	q_5



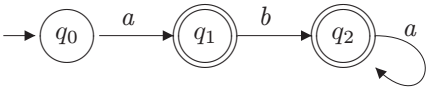
Iterativním postupem (podle délky cesty v grafu) vytvoříme množiny E_i:

- $E_0 = F = \{q_1, q_2\}$
  báze indukce; hledáme stavy ze kterých se dá dosáhnout koncového stavu, zde potřebujeme „konec hledaných cest“
- $E_1 = \{q_1, q_2, q_0\}$
  ze stavu q_0 je možné se provedením jednoho kroku přesunout do stavu q_1, který již máme v E_0
- $E_2 = \{q_1, q_2, q_0\}\ =\ E_1$ už není co přidat, končíme, $E_{def} = E_1 = E_2$

Prvky množiny E_{def} použijeme jako novou množinu stavů, vyřadíme stavy q_3, q_4, q_5:
$\mathcal{A}' = (\{q_0, q_1, q_2\},\ \{a, b\},\ \delta',\ q_0,\ \{q_1, q_2\})$

$\delta(q_0, a) = q_1$
$\delta(q_1, b) = q_2$
$\delta(q_2, a) = q_2$

	a	b
$\rightarrow q_0$	q_1	q_3
$\leftarrow q_1$		q_2
$\leftarrow q_2$	q_2	-



Všimněte si, že v tomto případě se zredukovala i abeceda automatu – na signál c se již nereaguje v žádném stavu automatu, tedy není důvod tento signál v abecedě nechávat.

Vyřadili jsme tři stavy, diagram automatu se zpřehlednil, jazyk rozpoznávaný automatem je $L(\mathcal{A}) = L(\mathcal{A}') = a + ab^*a$

Důkaz (Věta 3.6): Dokážeme, že výše popsaný algoritmus na daném automatu odstraňuje nadbytečné stavy a že původní a nově vytvořený automat jsou ekvivalentní.

Co se týče funkčnosti odstraňování nadbytečných stavů, důkaz je obdobný jako v předchozím případě – stačí použít matematickou indukci podle výše naznačeného způsobu konstrukce množiny E_{def}, tentokrát jdeme „proti směru cest“ v automatu. Bází je množina obsahující všechny koncové stavy. Krok indukce by pak zaručil, že do množiny E_{i+1} se navíc oproti množině E_i dostanou pouze stavy, ze kterých se lze dostat jedním krokem do prvků množiny E_i, a platí:
$E_0 \subseteq E_1 \subseteq \ldots \subseteq E_i \subseteq E_{i+1} \subseteq \ldots \subseteq E_{def}$

Tento algoritmus je konečný ze stejného důvodu, proč je konečný algoritmus pro odstranění nedostupných symbolů – množina Q je konečná, $E_{def} \subseteq Q$, je provedeno maximálně tolik kroků, kolik je stavů v Q.

Zbývá ukázat, že jazyk automatu zůstane nezměněn.

Dokazujeme $L(\mathcal{A}) \subseteq L(\mathcal{A}')$:
Vezměme jakékoliv slovo $w \in L(\mathcal{A})$.
$\Rightarrow$ existuje právě jeden výpočet slova w v $\mathcal{A}$:
$\mathcal{A}$: $(q_0, w) \vdash^* (q_f, \varepsilon), \quad q_f \in F,$
$\Rightarrow$ $\exists$ cesta v grafu $(q_0, \ldots, q_f)$
$\Rightarrow$ ze všech uzlů (stavů) na této cestě vede cesta do stavu $q_f \in F$, tedy patří do E_{def}
(důkaz by bylo možné provést matematickou indukcí podle délky cesty)
$\Rightarrow$ cesta v grafu $(q_0, \ldots, q_f)$ existuje také v grafu pro automat $\mathcal{A}'$
$\mathcal{A}'$: $(q_0, w) \vdash^* (q_f, \varepsilon), \quad q_f \in F$
$\Rightarrow$ $w \in L(\mathcal{A}')$

Dokazujeme $L(\mathcal{A}) \supseteq L(\mathcal{A}')$:
Vezměme jakékoliv slovo $w \notin L(\mathcal{A})$.
$\Rightarrow$ neexistuje výpočet slova w v $\mathcal{A}$, žádná posloupnost konfigurací a tedy ani cesta v grafu automatu $\mathcal{A}$ $(q_0, \ldots, q_f)$
$\wedge$ v δ-funkci $\mathcal{A}'$ jsme žádné nové přechody neumožnili (nepřidali)
$\Rightarrow$ $w \notin L(\mathcal{A}')$

Dokázali jsme, že popsaný postup odstranění nadbytečných stavů je správný, korektní a úplný, čímž je dokázána správnost celého postupu redukce množiny stavů automatu. □

3.5 Uzávěrové vlastnosti – regulární operace

V předchozím textu jsme pod pojmem *jazyk* chápali množinu slov. Protože v dále budeme pracovat s jazyky obecně a členit je do skupin podle jejich vlastností, definujeme si další pojem:

Definice 3.16 (Třída jazyků)

Třída jazyků je množina jazyků splňujících stanovené kritérium.

Například třída jazyků rozpoznávaných konečnými automaty je množina všech jazyků, pro které lze sestrojit konečný automat takový, který by je rozpoznával. Do této třídy patří i jazyk určený regulárním výrazem $a^*b(ba^*b)^*$ z příkladu 3.8 na straně 37 a mnoho dalších. Naopak do této třídy nepatří například jazyk $\{a^n b^n c^n \ ; \ n \geq 0\}$, protože pro něj nelze sestrojit konečný automat.

Zde nás budou zajímat uzávěrové vlastnosti této třídy jazyků.

Definice 3.17 (Uzavřenost třídy jazyků vzhledem k operaci φ)

Třída jazyků je uzavřená vzhledem k operaci φ, pokud po uplatnění této operace na jazyky z dané třídy výsledný jazyk patří opět do této třídy.

Možné operace, které z tohoto hlediska zkoumáme, jsou sjednocení, zřetězení, iterace, pozitivní iterace, průnik, doplněk, zrcadlový obraz (reverze), homomorfismus, substituce.

Uzavřenost třídy jazyků (v našem případě třídy jazyků rozpoznávaných konečnými automaty) vzhledem k operaci sjednocení například určuje, zda výsledek uplatnění sjednocení na jakékoliv dva jazyky z této třídy je opět jazyk z této třídy. U konečných automatů vlastnost uzavřenosti většinou dokazujeme jednoduše tak, že pro výsledný jazyk zkonstruujeme konečný automat – pak rozhodně dotyčný jazyk patří do třídy jazyků rozpoznávaných konečným automatem.

3.5.1 Sjednocení

U každé probírané operace vždy nejdřív vyslovíme větu, naznačíme postup konstrukce, ukážeme si postup na příkladu a poté větu dokážeme.

Věta 3.7

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci sjednocení.

Postup konstrukce: Jak bylo výše naznačeno, budeme předpokládat, že máme dva (jakékoliv, obecné) jazyky a k nim konečné automaty (dva, protože operace sjednocení je binární), sestrojíme konečný automat rozpoznávající sjednocení původních dvou jazyků. Označme naše dva jazyky L_1 a L_2, automaty pak $\mathcal{A}_1$ a $\mathcal{A}_2$. Předpokládáme, že $L_1 = L(\mathcal{A}_1),\ L_2 = L(\mathcal{A}_2)$:

- $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$,
- $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2), \quad Q_1 \cap Q_2 = \emptyset$.

Podmínka $Q_1 \cap Q_2 = \emptyset$ je důležitá – pokud není splněna, musíme v jednom z automatů přeznačit stavy (přejmenovat je), aby splněna byla.

Základní myšlenka je následující: výsledný automat má rozpoznávat jak slova jazyka L_1, tak i slova jazyka L_2. Nebude nutné konstruovat automat jako celek – použijeme metodu „rozděl a panuj“, což znamená, že se nebudeme zabývat tím, čím se už zabývá „někdo jiný“ (nebudeme odznova vytvářet všechny stavy a přechody, využijeme v maximální míře to, co už je vytvořeno v automatech A_1 a A_2).

Vytváříme jazyk $L = L_1 \cup L_2$ a automat $\mathcal{A} = (Q, \Sigma, \delta, s_0, F),\ L = L(\mathcal{A})$, přičemž

- stav s_0 je nový, tedy musí platit $s_0 \notin Q_1,\ s_0 \notin Q_2$ (taky můžeme napsat $s_0 \notin Q_1 \cup Q_2$),
- $Q = Q_1 \cup Q_2 \cup \{s_0\}$,
- $F = F_1 \cup F_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$.

Všimněte si, že jsme přidali pouze jediný stav, jinak využijeme stavy původních automatů. Podobně to bude i s δ-funkcí. Využijeme přechody z původních automatů a jen přidáme inicializaci (reakce ve stavu s_0) – takto:

Ze stavu s_0 přecházíme do těch stavů, do kterých se přechází z původních počátečních stavů q_0^1 a q_0^2.

Takže pokud v automatu $\mathcal{A}_1$ existuje přechod $\delta_1(q_0^1, a) = p$, také v automatu $\mathcal{A}$ budeme mít přechod $\delta(s_0, a) \ni p$ (ovšem ten původní neodstraníme). Obecný zápis:

$$\delta(p,a) = \begin{cases} \delta_1(p,a) \ ; \ p \in Q_1, \ a \in \Sigma \\ \delta_2(p,a) \ ; \ p \in Q_2, \ a \in \Sigma \\ \delta_1(q_0^1,a) \cup \delta_2(q_0^2,a) \ ; \ p = s_0, \ a \in \Sigma \end{cases}$$

Funkce δ je definována podle příslušnosti stavu p, ze kterého v daném kroku vycházíme – tato definice je typická pro případy vlastností souvisejících s prvky navzájem disjunktních množin (tady jsou tři: $Q_1, \ Q_2, \{s_0\}$).

První řádek předpisu říká, že pokud stav p patří do množiny Q_1 (a tedy přísluší do prvního automatu), reagujeme v něm stejně jako v prvním automatu (podle funkce δ_1). Druhý řádek předpisu je podobný, určuje, že ve stavech z automatu $\mathcal{A}_2$ reagujeme stejně jako ve stavu $\mathcal{A}_2$ (tj. podle funkce δ_2). Třetí řádek se vztahuje k nově vytvořenému stavu, který není v žádném z původních automatů – říká, že ve stavu s_0 se chováme stejně jako ve stavech q_0^1 a q_0^2. □

Příklad 3.9

Provedeme sjednocení následujících dvou jazyků:

$L_1 = \{a^i b^j \ ; \ i > 0, j \geq 0\} = aa^*b^* \quad \mathcal{A}_1 = (\{p_0, p_1, p_2\}, \ \{a, b\}, \ \delta_1, p_0, \ \{p_1, p_2\})$

$L_2 = \{a^i c^j \ ; \ i > 0, j \geq 0\} = aa^*c^* \quad \mathcal{A}_2 = (\{q_0, q_1, q_2\}, \ \{a, c\}, \ \delta_2, q_0, \ \{q_1, q_2\})$

Automaty pro jazyky L_1 a L_2:

$\mathcal{A}_1$	a	b	c
$\rightarrow p_0$	p_1		
$\leftarrow p_1$	p_1	p_2	
$\leftarrow p_2$		p_2	

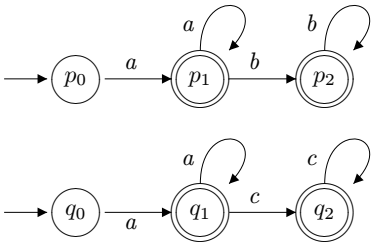
$\mathcal{A}_2$	a	b	c
$\rightarrow q_0$	q_1		
$\leftarrow q_1$	q_1		q_2
$\leftarrow q_2$			q_2

$\delta_1(p_0,a) = p_1 \quad \delta_1(p_1,b) = p_2$

$\delta_1(p_1,a) = p_1 \quad \delta_1(p_2,b) = p_2$

$\delta_2(q_0,a) = q_1 \quad \delta_2(q_1,c) = q_2$

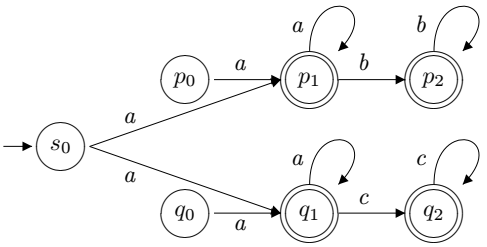
$\delta_2(q_1,a) = q_1 \quad \delta_2(q_2,c) = q_2$



Podle výše popsaného algoritmu provedeme sjednocení (vše z předchozích automatů přejmeme, přidáme nový počáteční stav, z něj vycházejí takové přechody, jaké vycházejí z původních počátečních stavů), výsledkem je tento automat:

$\mathcal{A}$	a	b	c
$\rightarrow s_0$	p_1, q_1		
p_0	p_1		
$\leftarrow p_1$	p_1	p_2	
$\leftarrow p_2$		p_2	
q_0	q_1		
$\leftarrow q_1$	q_1		q_2
$\leftarrow q_2$			q_2

$\delta(s_0,a) = \{p_1, q_1\}$
$\delta(p_0,a) = \{p_1\}$
$\delta(p_1,a) = \{p_1\}$
$\delta(p_1,b) = \{p_2\}$
$\delta(p_2,b) = \{p_2\}$
$\delta(q_0,a) = \{q_1\}$
$\delta(q_1,a) = \{q_1\}$
$\delta(q_1,c) = \{q_2\}$
$\delta(q_2,c) = \{q_2\}$



Výsledný automat rozpoznává tento jazyk:

$L = L_1 \cup L_2 = \{a^i b^j \ ; \ i > 0, j \geq 0\} \cup \{a^i c^j \ ; \ i > 0, j \geq 0\} = aa^*b^* + aa^*c^* = aa^*(b^* + c^*)$

Postup můžeme použít i na nedeterministické automaty, a dokonce výsledný automat $\mathcal{A}$ je nedeterministický (protože se chová nedeterministicky ve stavu s_0).

V úvahu by přicházela jedna úprava – na první pohled vidíme, že stavy p_1 a q_1 se staly nedostupnými (nejsou dosažitelné z počátečního stavu), tedy je můžeme odstranit.

Důkaz (Věta 3.7): Dokážeme, že výše popsaný algoritmus vytvoří automat rozpoznávající jazyk, který je sjednocením jazyků původních automatů, tedy že $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

Dokazujeme $L(\mathcal{A}_1) \cup L(\mathcal{A}_2) \subseteq L(\mathcal{A})$:

Vezměme jakékoliv slovo $w \in L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

$\Rightarrow$ slovo w patří buď do jazyka L_1 nebo do jazyka L_2 (nebo obou)
bez újmy na obecnosti předpokládejme, že $w \in L_1$

$\Rightarrow$ $\exists$ posloupnost konfigurací v $\mathcal{A}_1$
$(q_0^1, w) \vdash (p, w') \vdash \ldots \vdash (q_f, \varepsilon),\ p \in Q_1, q_f \in F_1$

$\Rightarrow$ první použitý přechod v $\mathcal{A}_1$ je definován jako $\delta_1(q_0^1, x) = p$, přičemž $w = x \cdot w'$

$\Rightarrow$ v $\mathcal{A}$ existuje $\delta(s_0, x) \ni \{p\}$

$\wedge$ do δ-funkce v automatu $\mathcal{A}$ jsme doplnili vše z funkce δ_1

$\Rightarrow$ $\exists$ posloupnost konfigurací v $\mathcal{A}$
$(s_0, w) \vdash (p, w') \vdash \ldots \vdash (q_f, \varepsilon),\ q_f \in f$

$\Rightarrow$ $w \in L(\mathcal{A})$

Dokazujeme $L(\mathcal{A}_1) \cup L(\mathcal{A}_2) \supseteq L(\mathcal{A})$:

Vezměme jakékoliv slovo $w \in L(\mathcal{A})$.

$\Rightarrow$ $\exists$ posloupnost konfigurací v $\mathcal{A}$
$(s_0, w) \vdash (p, w') \vdash \ldots \vdash (q_f, \varepsilon),\ p \in Q_1 \cup Q_2, q_f \in F_1 \cup F_2$

$\wedge$ do stavu s_0 nevede žádný přechod

$\Rightarrow$ pro všechny přechody použité v jakémkoliv výpočtu (kromě prvního přechodu) platí:
pokud $\delta(u, a) = v$ a $u \in Q_1$, pak také $v \in Q_1$
pokud $\delta(u, a) = v$ a $u \in Q_2$, pak také $v \in Q_2$

$\Rightarrow$ jestliže $p \in Q_1$, pak $w \in L(\mathcal{A}_1)$, jestliže $p \in Q_2$, pak $w \in L(\mathcal{A}_2)$

Z toho vyplývá, že jazyk automatu $\mathcal{A}$ je sjednocením jazyků automatů $\mathcal{A}_1$ a $\mathcal{A}_2$. □

3.5.2 Zřetězení

Další důležitou regulární operací je zřetězení. Podobně jako sjednocení (a iteraci) budeme tuto operaci potřebovat například při konstrukci automatu podle regulárního výrazu, tímto způsobem dokážeme zkonstruovat automat i pro hodně složitý regulární výraz.

Věta 3.8

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci zřetězení.

Postup konstrukce: Sestrojíme konečný automat rozpoznávající zřetězení jazyků jiných dvou konečných automatů. Označme naše dva jazyky L_1 a L_2, automaty pak $\mathcal{A}_1$ a $\mathcal{A}_2$. Předpokládáme, že $L_1 = L(\mathcal{A}_1),\ L_2 = L(\mathcal{A}_2)$:

- $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$,
- $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2), \quad Q_1 \cap Q_2 = \emptyset$.

Podmínka $Q_1 \cap Q_2 = \emptyset$ je stejná jako v předchozím případě.

Základní myšlenka je následující: výsledný automat má rozpoznávat slova, jejichž první část je vždy slovo patřící do jazyka L_1, zbytek je slovo patřící do jazyka L_2. Opět použijeme metodu „rozděl a panuj“, tedy v co největší míře použijeme to, co už existuje v automatech A_1 a A_2. Jakékoliv slovo $w \in L(\mathcal{A})$ se skládá ze dvou řetězců $w = u \cdot v$, kde $u \in L(\mathcal{A}_1),\ v \in L(\mathcal{A}_2)$. Nejdřív zpracujeme podslovo u (simulujeme výpočet v automatu $\mathcal{A}_1$), pak podslovo v (simulujeme výpočet v $\mathcal{A}_2$).

Vytváříme jazyk $L = L_1 \cdot L_2$ a automat $\mathcal{A} = (Q, \Sigma, \delta, s_0, F),\ L = L(\mathcal{A})$, přičemž oba původní automaty „zřetězíme“ za sebou – každý výpočet půjde nejdřív přes stavy automatu $\mathcal{A}_1$, pak přes stavy automatu $\mathcal{A}_2$.

- jako počáteční stav použijeme stav q_0^1, začínáme v automatu $\mathcal{A}_1$,
- $Q = Q_1 \cup Q_2$, nové stavy nepřidáváme,
- $F = \begin{cases} F_2 \ ; \ \varepsilon \notin L_2 \\ F_1 \cup F_2 \ ; \ \varepsilon \in L_2 \end{cases}$ Jestliže v jazyce L_2 je prázdné slovo, pak může nastat případ, kdy $w = w \cdot \varepsilon$, a tedy všechny symboly slova $w \in L$ mají být ve skutečnosti rozpoznány v automatu $\mathcal{A}_1$; pak je třeba skončit už stavech z množiny F_1.
- $\Sigma = \Sigma_1 \cup \Sigma_2$.

Co se týče δ-funkce, využijeme přechody z původních automatů a jen přidáme přechody umožňující přechod ze stavů prvního původního automatu do druhého:

$$\delta(p,a) = \begin{cases} \delta_1(p,a) \ ; \ p \in Q_1 - F_1,\ a \in \Sigma & \ldots\ldots\ldots\ldots\ldots \text{jsme v první části slova, } u \\ \delta_1(p,a) \cup \delta_2(q_0^2, a) \ ; \ p \in F_1,\ a \in \Sigma & \ldots\ldots\ldots\ldots\ldots \text{jsme na rozhraní } u \text{ a } v \\ \delta_2(p,a) \ ; \ p \in Q_2,\ a \in \Sigma & \ldots\ldots\ldots\ldots\ldots \text{jsme v druhé části slova, } v \end{cases}$$

První řádek předpisu říká, že pokud stav p patří do množiny Q_1 a zároveň nikoliv do F_1, reagujeme v něm stejně jako v prvním automatu (podle funkce δ_1). Druhý řádek předpisu určuje, že pokud jsme v některém původním koncovém stavu automatu $\mathcal{A}_1$, reagujeme buď stejně jako v automatu $\mathcal{A}_1$ (i z koncového stavu může vést cesta pro další výpočet), nebo se přesuneme do druhého původního automatu (pokud označíme $w = u \cdot v$, pak jsme zpracovali podslovo u a začínáme zpracovávat slovo v). Třetí řádek stanovuje, že při zpracování druhé části slova přejímáme přechody z δ_2 v druhém původním automatu.

V druhém řádku předpisu je důležité, že ke stavům původně příslušejícím do F_1 přidáváme reakce, které „kopírujeme“ z počátečního stavu automatu $\mathcal{A}_2$, stavu q_0^2. □

Příklad 3.10

Provedeme zřetězení následujících dvou jazyků:

$L_1 = \{a^i b \ ; \ i \geq 0\} = a^* b$ $\qquad \mathcal{A}_1 = (\{p_0, p_1\},\ \{a, b\},\ \delta_1, p_0,\ \{p_1\})$

$L_2 = \left\{(aba)^i c^{\{0,1\}} \ ; \ i \geq 0\right\} = (aba)^*(\varepsilon + c)$ $\qquad \mathcal{A}_2 = (\{q_0, \ldots, q_3\},\ \{a, b, c\},\ \delta_2, q_0,\ \{q_0, q_1\})$

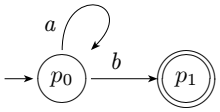
Automaty pro jazyky L_1 a L_2:

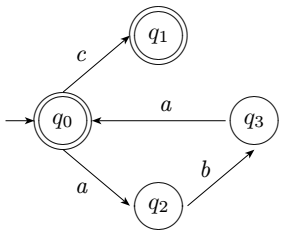
$\mathcal{A}_1$	a	b	c
$\to p_0$	p_0	p_1	
$\leftarrow p_1$			

$\delta_1(p_0, a) = p_0$
$\delta_1(p_0, b) = p_1$

$\mathcal{A}_2$	a	b	c
$\leftrightarrow q_0$	q_2		q_1
$\leftarrow q_1$			
q_2		q_3	
q_3	q_0		

$\delta_2(q_0, a) = q_2$
$\delta_2(q_0, c) = q_1$
$\delta_2(q_2, b) = q_3$
$\delta_2(q_3, a) = q_0$

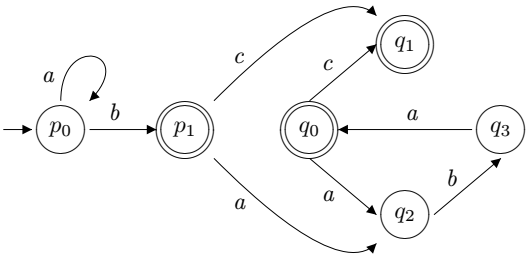




Podle výše popsaného algoritmu provedeme zřetězení, výsledkem je tento automat:
$\mathcal{A} = (\{p_0, p_1, q_0, q_1, q_2, q_3\},\ \{a, b, c\},\ \delta,\ p_0,\ \{p_1, q_0, q_1\})$

$\mathcal{A}$	a	b	c
$\to p_0$	p_0	p_1	
$\leftarrow p_1$	q_2		q_1
$\leftarrow q_0$	q_2		q_1
$\leftarrow q_1$			
q_2		q_3	
q_3	q_0		

$\delta(p_0, a) = p_0$
$\delta(p_0, b) = p_1$
$\delta(p_1, a) = q_2$
$\delta(p_1, c) = q_1$
$\delta(q_0, a) = q_2$
$\delta(q_0, c) = q_1$
$\delta(q_2, b) = q_3$
$\delta(q_3, a) = q_0$



Je třeba si dát pozor na jednu věc – v tomto příkladu platí, že $\varepsilon \in L_2$, a tedy do množiny koncových stavů F zařadíme i stavy z množiny F_1. Pokud by však $\varepsilon \notin L_2$, do množiny F bychom zařadili pouze stavy patřící do F_2 (takže stav p_2 by nebyl koncový).

Všimněte si, že zde nám vyšel deterministický automat (ale pozor, to není pravidlo – na jednom místě v definici δ-funkce máme sjednocení, proto i tehdy, když jsou oba automaty $\mathcal{A}_1$ a $\mathcal{A}_2$ deterministické, může být obecně výsledkem automat nedeterministický).
Výsledný automat rozpoznává tento jazyk:
$L = L_1 \cdot L_2 = \{a^i b\ ;\ i \geq 0\} \cdot \left\{(aba)^i c^{\{0,1\}}\ ;\ i \geq 0\right\} = a^* b \cdot (aba)^*(\varepsilon + c)$

Důkaz (Věta 3.8): Dokážeme, že výše popsaný algoritmus vytvoří automat rozpoznávající jazyk, který je zřetězením jazyků původních automatů, tedy že $L(\mathcal{A}) = L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.
Dokazujeme $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2) \subseteq L(\mathcal{A})$:
Vezměme jakékoliv slovo $w \in L(\mathcal{A}_1) \cdot L(\mathcal{A}_2),\ w = u \cdot v,\ u \in L(\mathcal{A}_1),\ v \in L(\mathcal{A}_2)$ (tj. slovo w lze rozdělit na dvě části $w = u \cdot v$ takové, že $u \in L_1$ a $v \in L_2$).

$\Rightarrow$ $\exists$ posloupnost konfigurací v $\mathcal{A}_1$ a v $\mathcal{A}_2$
$(q_0^1, u) \vdash \ldots \vdash (q_f^1, \varepsilon),\ q_f^1 \in F_1$
$(q_0^2, v) \vdash (q_x, v') \vdash \ldots \vdash (q_f^2, \varepsilon),\ q_x \ni \delta_2(q_0^2, a),\ q_f^2 \in F_2,\ v = a \cdot v'$
$\wedge$ $\delta(q_f^1, x) = \delta_1(q_f^1, x) \cup \delta_2(q_0^2, x)$, kde $x \in \Sigma$
$\Rightarrow$ $\exists$ posloupnost konfigurací v $\mathcal{A}$
$(q_0^2, u \cdot v) \vdash \ldots \vdash (q_f^1, av') \vdash (q_x, v') \vdash \ldots \vdash (q_f^2, \varepsilon),\ q_f^2 \in F_2$
$\Rightarrow$ $w \in L(\mathcal{A})$

Dokazujeme $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2) \supseteq L(\mathcal{A})$:
Vezměme jakékoliv slovo $w \in L(\mathcal{A})$.
$\Rightarrow$ všechny cesty v grafu automatu $\mathcal{A}$ vedou přes minimálně jeden stav z množiny F_1
nejdřív předpokládejme, že $\varepsilon \notin L_2$
$\Rightarrow$ $\exists$ posloupnost konfigurací v $\mathcal{A}$
$(q_0^1, w) \vdash \ldots \vdash (q_f^1, av') \vdash (q_x, v') \vdash \ldots (q_f^2, \varepsilon),\ q_f^1 \in F_1,\ q_f^2 \in F_2,\ q_x \ni \delta_2(q_0^2, a),\ w = u \cdot a \cdot v'$
$\Rightarrow$ v automatu $\mathcal{A}_1$ existuje posloupnost konfigurací
$(q_0^1, u) \vdash \ldots \vdash (q_f^1, \varepsilon)$
$\wedge$ v automatu $\mathcal{A}_2$ existuje posloupnost konfigurací
$(q_0^2, av') \vdash \ldots \vdash (q_f^1, \varepsilon)$
$\Rightarrow$ pro slovo w jsme nalezli rozdělení $w = u \cdot v = u \cdot a \cdot v'$ takové, že $u \in L(\mathcal{A}_1)$ a $v \in L(\mathcal{A}_2)$

dále předpokládejme, že $\varepsilon \in L_2$
$\Rightarrow$ $F_1 \subseteq F$ protože v případě, že $\varepsilon \in L_2,\ F = F_1 \cup F_2$
$\wedge$ pro slovo w existují dvě možnosti: $w \in L(\mathcal{A}_1)$ nebo $w \notin L(\mathcal{A}_1)$
první možnost: v $\mathcal{A}$ existuje posloupnost konfigurací
$(q_0^1, w) \vdash \ldots \vdash (q_f^1, \varepsilon), \quad q_f^1 \in F \quad \wedge \quad q_f^1 \in F_1$
$\Rightarrow \quad w = w \cdot \varepsilon, \quad w \in L(\mathcal{A}_1),\ \varepsilon \in L(\mathcal{A}_2)$
druhá možnost: dokázala by se stejně jako v případě, že $\varepsilon \notin L_2$
$\Rightarrow$ u obou možností jsme ukázali, že w lze rozložit tak, že první část $\in L_1$ a druhá část $\in L_2$

Z toho vyplývá, že jazyk automatu $\mathcal{A}$ je zřetězením jazyků automatů $\mathcal{A}_1$ a $\mathcal{A}_2$. □

3.5.3 Iterace

Iterace na rozdíl od předchozích operací není binární, takže v popisu konstrukce a v důkazu budeme předpokládat jeden původní automat místo dvou.

Věta 3.9

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci iterace (Kleeneho uzávěru, operaci hvězdička).

Postup konstrukce: Vezměme jakýkoliv konečný automat $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ rozpoznávající jazyk L_1. Sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, s_0, F)$ takový, že $L(\mathcal{A}) = L_1^* = \bigcup_{i=0}^{\infty} L_1^i$.

Jak vidíme, iterace je vlastně speciálním případem zřetězení s tím, že zřetězujeme pořád tentýž jazyk, ovšem jakýkolivpočetkrát. Proto se i v postupu inspirujeme u algoritmu pro zřetězení, ale navíc je třeba zajistit, aby přidání slova ε do výsledného jazyka nezpůsobilo „předčasné“ ukončení výpočtu. Tedy postupujeme následovně:

- vytvoříme nový počáteční stav s_0 (musí platit $s_0 \notin Q_1$),
- přidáme přechody vedoucí ze stavu s_0, v tomto stavu se automat bude chovat jako v původním počátečním stavu,
- $Q = Q_1 \cup \{s_0\}$,
- $F = F_1 \cup \{s_0\}$ (i tehdy, když do jazyka L_1 nepatřilo slovo ε, ve výsledném jazyce bude),

- zajistíme iteraci – v koncových stavech přidáme reakce stejné, jaké jsou v počátečním stavu (tj. tytéž, jaké jsme přidávali k s_0).

Zápis přechodové funkce:

$$\delta(p,a) = \begin{cases} \delta_1(p,a)\ ;\ p \in Q_1 - F_1,\ a \in \Sigma \\ \delta_1(p,a) \cup \delta_1(q_0^1,a)\ ;\ p \in F_1,\ a \in \Sigma \\ \delta_1(q_0^1,a)\ ;\ p = s_0,\ a \in \Sigma \end{cases}$$

První řádek předpisu říká, že ve stavu p z množiny Q_1, který není koncový, se chováme naprosto stejně jako v původním automatu. Pokud se jedná o některý z původních koncových stavů, pak kromě původního chování přidáme ještě chování převzaté od původního počátečního stavu – tyto přechody nám zajišťují „návrat“ na začátek s tím, že ve zřetězení slov jazyka L_1 přecházíme na další slovo v pořadí. Třetí řádek určuje, jak se má automat chovat na začátku výpočtu – stejně jako v původním počátečním stavu. □

Poznámka:

Vytvoření nového počátečního stavu se může zdát zbytečné a ve většině případů také zbytečné je, nicméně v některých případech je nutné. Uvědomme si, že tímto krokem zajistíme, aby se počáteční stav nevyskytoval v jiné než v počáteční konfiguraci (u jakékoliv cesty v grafu je vždy jen na začátku, dál už ne). Proto se do počátečního stavu nemůžeme vracet. Jestliže v jazyce původního automatu nebylo slovo ε a zároveň do tohoto stavu vedla některá cesta, pak bychom prostým „zakroužkováním“ počátečního stavu (při zařazování slova ε do jazyka) přidali do výsledného jazyka i taková slova, která tam nemají co dělat.

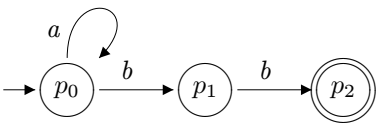
Proto budeme postupovat takto: jestliže do původního počátečního stavu nevedou žádné cesty, nemusíme vytvářet nový počáteční stav, použijeme původní (abychom zbytečně nevyráběli nedosažitelné stavy, protože přesně tím se původní počáteční stav stane). Jestliže však do původního počátečního stavu vede cesta, pak vytvoříme nový počáteční stav, do kterého žádná taková cesta nepovede.

Příklad 3.11

Postup si ukážeme na automatu rozpoznávajícím jazyk $L_1 = \{a^i bb\ ;\ i \geq 0\} = a^*bb$. Tento jazyk je rozpoznáván automatem $\mathcal{A}_1 = (\{p_0,p_1,p_2\},\ \{a,b\},\ \delta_1,\ p_0,\ \{p_2\})$ určeným takto:

$\mathcal{A}_1$	a	b
$\rightarrow p_0$	p_0	p_1
p_1		p_2
$\leftarrow p_2$		

$$\delta_1(p_0,a) = p_0$$
$$\delta_1(p_0,b) = p_1$$
$$\delta_1(p_1,b) = p_2$$



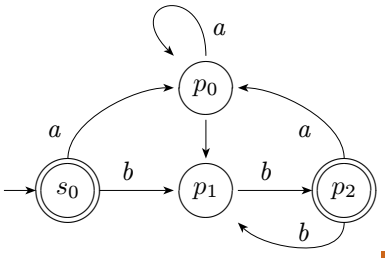
Sestrojíme automat $\mathcal{A}$ rozpoznávající jazyk $L(\mathcal{A}) = L_1^*$: $\mathcal{A} = (\{p_0,p_1,p_2,s_0\},\ \{a,b\},\ \delta,\ s_0,\ \{s_0,p_2\})$.

Protože stav p_0 nepatří do množiny koncových stavů a zároveň do něj vede cesta (třebaže ze sebe sama), musíme vytvořit nový počáteční stav s_0, ve kterém budeme reagovat stejně jako v původním p_0 (co vede ze stavu p_0, to povede také ze stavu s_0). Stav s_0 bude patřit do množiny koncových stavů, protože v jazyce $L(\mathcal{A})$ musí být i prázdné slovo.

Následně přidáme nové přechody vedoucí z koncových stavů (v tomto případě tedy ze stavu p_2, stav s_0 je už v tomto ohledu vyřešen) – opět „zkopírujeme“ přechody z počátečního stavu.

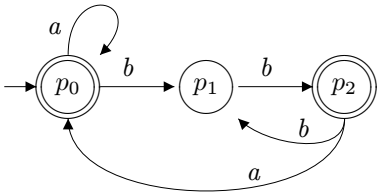
$\mathcal{A}$	a	b
$\leftrightarrow s_0$	p_0	p_1
p_0	p_0	p_1
p_1		p_2
$\leftarrow p_2$	p_0	p_1

$\delta(s_0, a) = p_0$ $\quad$ $\delta(p_1, b) = p_2$
$\delta(s_0, b) = p_1$ $\quad$ $\delta(p_2, a) = p_0$
$\delta(p_0, a) = p_0$ $\quad$ $\delta(p_2, b) = p_1$
$\delta(p_0, b) = p_1$



Poznámka:

Zamysleme se nad tím, proč vlastně v některých případech musíme vytvořit nový počáteční stav. Pokud bychom v předchozím příkladu tento krok neprovedli a vytvořili bychom tento automat – viz vpravo, pak tento automat rozpoznával například slovo a^2, které do jazyka $L(\mathcal{A})$ patřit nemá (ten jazyk je $(a^*bb)^*$, což znamená, že za každou posloupností symbolů a musí následovat sudý počet symbolů b, nejméně dva). Pokud bychom tedy tuto chybu udělali, „obohatili“ bychom výsledný jazyk o taková slova, která v něm nemají co dělat.



Důkaz (Věta 3.9): Dokážeme, že výše popsaný algoritmus vytvoří automat rozpoznávající jazyk, který je iterací jazyka původního automatu, tedy že $L(\mathcal{A}) = L(\mathcal{A}_1)^*$. Použijeme důkaz matematickou indukcí (protože máme potenciálně nekonečnou posloupnost zřetězení) takto:

1. Protože platí $L^* = \bigcup_{i=0}^{\infty} L^i$, matematická indukce se bude týkat různého počtu zřetězení jazyka, tedy postupně probereme L^0, L^1, atd.
2. Vždy zvlášť vydělíme případ prázdného slova, včetně báze indukce.

Ke konečnému automatu $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ rozpoznávajícímu jazyk L_1 sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, s_0, F)$ tak, jak bylo popsáno v konstrukci před příkladem.

Báze: *Dokazujeme* $L_1^0 \subseteq L(\mathcal{A})$:
To je triviální – $L_1^0 = \{\varepsilon\}$ a víme, že $s_0 \in F$ (proto $\varepsilon \in L(\mathcal{A})$, tedy $L_1^0 \subseteq L(\mathcal{A})$.

Dokazujeme $L_1^1 \subseteq L(\mathcal{A})$: rozlišíme dva případy – $|w| = 0$ a $|w| \geq 1$. První případ je opět triviální, protože $s_0 \in F$. Pokud $\varepsilon \in L_1$, pak taky $\varepsilon \in L(\mathcal{A})$.

Pokud $|w| \geq 1$, označme $w = a \cdot w',\ a \in \Sigma$.
$\Rightarrow$ v automatu $\mathcal{A}_1$ existuje výpočet
$(q_0^1, w) \vdash (q_x, w') \vdash \ldots \vdash (q_f, \varepsilon),\ q_x \in Q_1,\ q_f \in F_1$
$\Rightarrow$ v $\mathcal{A}_1$ existuje předpis $\delta_1(q_0^1, a) \ni q_x$
$\Rightarrow$ v $\mathcal{A}$ existuje předpis $\delta(s_0, a) \ni q_x$
$\Rightarrow$ v automatu $\mathcal{A}$ existuje výpočet
$(s_0, w) \vdash (q_x, w') \vdash \ldots \vdash (q_f, \varepsilon),\ q_x \in Q,\ q_f \in F$
$\Rightarrow$ $w \in L(\mathcal{A})$ $\quad\Rightarrow\quad$ $L_1^1 \subseteq L(\mathcal{A})$

Předpoklad: Dále budeme předpokládat, že $L_1^k \subseteq L(\mathcal{A})$ pro nějaké $k \geq 1$.

Krok indukce: *Dokazujeme*, že $L_1^{k+1} \subseteq L(\mathcal{A})$.
Vezměme slovo $w = w_1 \cdot w_2 \cdot \ldots \cdot w_{k+1}$, kde $|w_i| \geq 1,\ w_i = a_i \cdot w_i',\ w_i \in L_1$ pro $1 \leq i \leq k+1$.
$\Rightarrow$ pro každé $i \in \{1, \ldots, k+1\}$ existuje výpočet v $\mathcal{A}_1$:
$(q_0^1, w_i) \vdash (q_{x_i}, w_i') \vdash \ldots \vdash (q_{f_i}, \varepsilon),\ q_{x_i} \in Q_1,\ q_{f_i} \in F_1$
$\Rightarrow$ v $\mathcal{A}_1$ existují předpisy $\delta_1(q_0^1, a_i) \ni q_{x_i}$
$\Rightarrow$ v $\mathcal{A}$ existují předpisy $\delta(s_0, a_i) \ni q_{x_i},\ \delta(q_f, a_i) \ni q_{x_i}$ pro některé $q_f \in F$
$\Rightarrow$ pro každé $i \in \{1, \ldots, k+1\}$ v automatu $\mathcal{A}$ existují podvýpočty
$(s_0, w_i) \vdash (q_{x_i}, w_i') \vdash \ldots \vdash (q_{f_i}, \varepsilon),\ q_{x_i} \in Q,\ q_{f_i} \in F$
$(q_f, w_i) \vdash (q_{x_i}, w_i') \vdash \ldots \vdash (q_{f_i}, \varepsilon),\ q_{x_i} \in Q,\ q_{f_i} \in F$
$\Rightarrow$ první použijeme pro slovo w_1, druhý pro slova $w_i,\ i \in \{2, \ldots, k+1\}$:
$(s_0, w_1 \cdot w_2 \cdot \ldots \cdot w_{k+1}) \vdash (q_{x_1}, w_1' \cdot w_2 \cdot \ldots \cdot w_{k+1}) \vdash \ldots$
$\vdash (q_{f_1}, w_2 \cdot \ldots \cdot w_{k+1}) \vdash (q_{x_2}, w_2' \cdot \ldots \cdot w_{k+1}) \vdash \ldots$
$\vdash (q_{f_k}, w_{k+1}) \vdash (q_{x_{k+1}}, w_{k+1}') \vdash \ldots \vdash (q_{f_{k+1}}, \varepsilon)$
$\Rightarrow$ $w \in L(\mathcal{A}) \quad \Rightarrow \quad L_1^{k+1} \subseteq L(\mathcal{A})$

Zbývá dokázat opačný směr – pokud slovo $w \notin L_1^*$, pak také $w \notin L(\mathcal{A})$. Do funkce δ jsme přidávali pouze takové přechody, které zajistily „návrat“ na začátek po ukončení zpracování jednoho slova z jazyka L_1, a zároveň jsme vytvořením nového počátečního stavu odstranili případy „předčasného ukončení“ výpočtu, proto můžeme říct, že pokud $w \notin L_1^*$, pak $w \notin L(\mathcal{A})$.

Z toho vyplývá, že jazyk automatu $\mathcal{A}$ je iterací jazyka automatu $\mathcal{A}_1$. □

Poznámka:

V kapitole 2.3 na straně 29 jsme definovali množinu všech regulárních výrazů $RV(\Sigma)$ nad abecedou Σ s využitím operací sjednocení, zřetězení a iterace. V sekcích této kapitoly jsou uvedeny postupy, jak konstruovat konečný automat podle jiného, a to právě na základě těchto operací.

Tyto postupy lze použít i v trochu jiném typu úlohy – „podle zadaného regulárního výrazu sestrojte ekvivalentní konečný automat“ (resp. sestrojte konečný automat rozpoznávající jazyk určený daným regulárním výrazem). Postupujeme takto:

- složitý regulární výraz rozdělíme na jednoduché podvýrazy, ke kterým dokážeme bez problémů sestrojit konečný automat,
- sestrojíme tyto dílčí konečné automaty,
- podle předpisu v celkovém regulárním výrazu je pospojujeme – podle operace, která je má vázat (sjednocení, zřetězení, iterace).

Tímto způsobem dokážeme sestrojit konečný automat i pro dlouhý a komplikovaný regulární výraz, u kterého bychom to „najednou“ nezvládli.

3.6 Uzávěrové vlastnosti – některé další operace

Třída jazyků rozpoznávaných konečnými automaty je ve skutečnosti uzavřena i vzhledem k dalším operacím (přesněji – prakticky vzhledem ke každé operaci). Důkazy jako v předchozích případech lze tedy provést i pro operaci pozitivní iterace, zrcadlového obrazu, rozdílu, doplňku, průniku, homomorfismu, substituce. V tomto semestru již nebudeme probírat důkazy dalších uzávěrových vlastností, jen si ukážeme způsob konstrukce.

3.6.1 Pozitivní iterace

Operace pozitivní iterace je podobná operaci iterace (Kleeneho uzávěru, hvězdičce) s tím rozdílem, že pokud do původního jazyka nepatřilo prázdné slovo, nebude patřit ani do výsledného jazyka. Srovnejme definici operace iterace a pozitivní iterace (původní jazyk označme L):

$$\text{iterace:} \quad L^* = \bigcup_{i=0}^{\infty} L^i \qquad \text{pozitivní iterace:} \quad L^+ = \bigcup_{i=1}^{\infty} L^i$$

Věta 3.10

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci pozitivní iterace.

Postup konstrukce: Vezměme jakýkoliv konečný automat $\mathcal{A}_1 = (Q, \Sigma, \delta_1, q_0, F)$ rozpoznávající jazyk L_1. Sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ takový, že $L(\mathcal{A}) = L_1^+ = \bigcup_{i=1}^{\infty} L_1^i$. Tentokrát si nemusíme „hrát“ s počátečním stavem, protože do jazyka explicitně nepřidáváme prázdné slovo.

Postupujeme následovně:

- použijeme původní počáteční stav, množinu stavů, množinu koncových stavů,
- zajistíme iteraci – v koncových stavech přidáme reakce stejné, jaké jsou v počátečním stavu.

Zápis přechodové funkce:

$$\delta(p,a) = \begin{cases} \delta_1(p,a) \ ; \ p \in Q - F, \ a \in \Sigma \\ \delta_1(p,a) \cup \delta_1(q_0,a) \ ; \ p \in F, \ a \in \Sigma \end{cases}$$

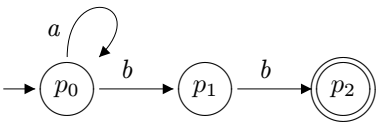
První řádek předpisu použijeme pro všechny stavy kromě koncových (jen přejmeme chování z původního automatu), druhý řádek platí pro koncové stavy. V nich necháme původní přechody a přidáme ty přechody, které vedou z počátečního stavu. □

Příklad 3.12

Postup si ukážeme na automatu rozpoznávajícím jazyk $L_1 = \{a^i bb \ ; \ i \geq 0\} = a^*bb$ (stejném jako u iterace). Tento jazyk je rozpoznáván automatem $\mathcal{A}_1 = (\{p_0, p_1, p_2\}, \{a, b\}, \delta_1, p_0, \{p_2\})$ určeným takto:

$\mathcal{A}_1$	a	b
$\rightarrow p_0$	p_0	p_1
p_1		p_2
$\leftarrow p_2$		

$$\delta_1(p_0, a) = p_0$$
$$\delta_1(p_0, b) = p_1$$
$$\delta_1(p_1, b) = p_2$$



Sestrojíme automat $\mathcal{A}$ rozpoznávající jazyk $L(\mathcal{A}) = L_1^+$: $\mathcal{A} = (\{p_0, p_1, p_2\},\ \{a, b\},\ \delta,\ p_0,\ \{p_2\})$.

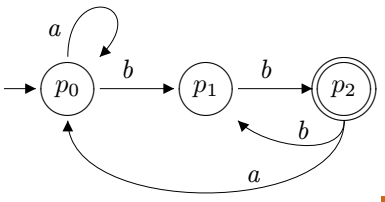
Přidáme nové přechody vedoucí z koncových stavů (v tomto případě ze stavu p_2) – „zkopírujeme“ přechody z počátečního stavu.

$\mathcal{A}$	a	b
$\rightarrow p_0$	p_0	p_1
p_1		p_2
$\leftarrow p_2$	p_0	p_1

$\delta(p_0, a) = p_0$ $\quad$ $\delta(p_2, a) = p_0$

$\delta(p_0, b) = p_1$ $\quad$ $\delta(p_2, b) = p_1$

$\delta(p_1, b) = p_2$



3.6.2 Zrcadlový obraz

Zrcadlový obraz slova provedeme tak, že „obrátíme pořadí symbolů“ v daném slově, zrcadlový obraz jazyka sestrojíme tak, že totéž provedeme se všemi slovy daného jazyka.

Věta 3.11

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci zrcadlového obrazu (reverze).

Jak tedy na konečném automatu provést tuto operaci? Jednoduše tak, že „obrátíme pořadí“ zpracovávání symbolů jazyka, jinými slovy – převrátíme všechny cesty v automatu (tam, kde jsme předtím začínali, budeme končit, tam, kde jsme končili, začneme). Počáteční stav se stane koncovým a naopak. Jediný problém, který musíme vyřešit, je případ, kdy původní automat má více než jeden koncový stav (všechny se nemohou stát počátečními, vždy musí být právě jeden počáteční stav). Níže naznačený postup je obecně platný pro jakýkoliv konečný automat (s různým počtem koncových stavů).

Postup konstrukce: Budeme předpokládat, že jazyk L_1 je rozpoznáván konečným automatem $\mathcal{A}_1$, kde $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$, $L_1 = L(\mathcal{A}_1)$. Vytváříme jazyk $L = L_1^R$ a konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, $L = L(\mathcal{A})$. Postupujeme takto:

- převrátíme všechny přechody,
- vytvoříme nový počáteční stav q_0, platí $q_0 \notin Q_1$,
- ze stavu q_0 budou vést přechody, které (po převrácení přechodů) vedou z bývalých koncových stavů (tj. z momentálních „virtuálních počátečních stavů“),
- množina koncových stavů bude obsahovat pouze původní počáteční stav, ale pokud do jazyka L_1 patřilo i slovo ε, pak tam zařadíme i stav q_0.

Takže platí:

- $Q = Q_1 \cup \{q_0\}$,
- $F = \begin{cases} \{q_0^1\}\ ;\ \varepsilon \notin L_1 \\ \{q_0^1, q_0\}\ ;\ \varepsilon \in L_1 \end{cases}$
- přechodová funkce: $\delta(p, a) = \begin{cases} \{r\ ;\ \delta_1(r, a) \ni p,\ a \in \Sigma\}\ ;\ p \in Q_1 \\ \{r\ ;\ \delta_1(r, a) \ni q_f,\ q_f \in F_1,\ a \in \Sigma\}\ ;\ p = q_0 \end{cases}$

Přechodová funkce vypadá trochu nepřehledně, ale ve skutečnosti není tak složitá – první řádek nám říká, že všechny existující přechody mají být převráceny (tj. $\delta(p,a) = r$, pokud $\delta_1(r,a) = p$ pro deterministický automat, nedeterministický by používal symboly $\in$ nebo $\ni$), druhý řádek řeší přechody v novém stavu q_0 (ze stavu q_0 vedou ty přechody, které dle δ_1 směřovaly *do* původních koncových stavů. □

Příklad 3.13

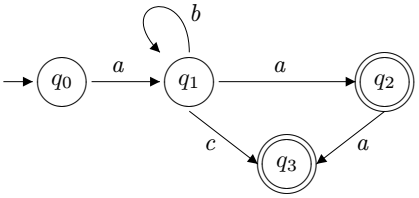
Je dán tento jazyk a konečný automat, který ho rozpoznává:

$L_1 = \left\{ab^i a^{\{1,2\}} \ ; \ i \geq 0\right\} \cup \{ab^i c \ ; \ i \geq 0\}$
$= \{ab^i \ ; \ i \geq 0\} \cdot \{a, aa, c\}$

$\mathcal{A}_1 = (\{q_0, q_1, q_2, q_3\},\ \{a, b, c\},\ \delta_1,\ q_0,\ \{q_2, q_3\})$

$\mathcal{A}_1$	a	b	c
$\rightarrow q_0$	q_1		
q_1	q_2	q_1	q_3
$\leftarrow q_2$	q_3		
$\leftarrow q_3$			

$\delta_1(q_0, a) = q_1$
$\delta_1(q_1, a) = q_2$
$\delta_1(q_1, b) = q_1$
$\delta_1(q_1, c) = q_3$
$\delta_1(q_2, a) = q_3$



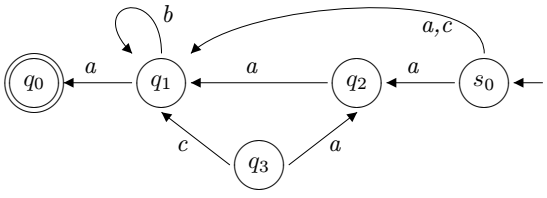
Sestrojíme automat $\mathcal{A}$ rozpoznávající jazyk $L(\mathcal{A}) = L_1^R$. První krok je jednoduchý – otočíme všechny existující přechody. V diagramu změníme orientaci hran, v předpisu přechodové funkce zaměníme stav v závorce se stavem za rovnítkem (zde je výsledkem nedeterministický automat, což při této úpravě musíme brát v úvahu), v tabulce zaměníme umístění stavu v označení řádku a stavu v buňce na tomto řádku.

Protože automat $\mathcal{A}_1$ má dva koncové stavy, nemůžeme za nový počáteční stav jednoduše prohlásit původní koncový, ale je třeba vytvořit nový stav s_0, který „přejme roli“ těchto stavů na začátku výpočtu každého slova v automatu $\mathcal{A}$.

$\mathcal{A} = (\{q_0, q_1, q_2, q_3, s_0\},\ \{a, b, c\},\ \delta,\ s_0,\ \{q_0\})$

$\mathcal{A}$	a	b	c
$\rightarrow s_0$	q_1, q_2		q_1
$\leftarrow q_0$			
q_1	q_0	q_1	
q_2	q_1		
q_3	q_2		q_1

$\delta(s_0, a) = \{q_1, q_2\}$
$\delta(s_0, c) = \{q_1\}$
$\delta(q_1, a) = \{q_0\}$
$\delta(q_1, b) = \{q_1\}$
$\delta(q_2, a) = \{q_1\}$
$\delta(q_3, a) = \{q_2\}$
$\delta(q_3, c) = \{q_1\}$



Sestrojený automat $\mathcal{A}$ rozpoznává jazyk $L = L_1^R = \{a, aa, c\} \cdot \{b^i a \ ; \ i \geq 0\}$.

Je zřejmé, že stav q_3 je nedosažitelný, tedy dalším krokem by mohlo být jeho odstranění. Výsledný automat by pak měl stejný počet stavů jako původní.

Pokud původní automat má pouze jeden koncový stav, pak samozřejmě není třeba do vytvářeného automatu přidávat nový počáteční stav, stačí použít původní koncový.

3.6.3 Průnik

Průnikem dvou jazyků je jazyk obsahující pouze ta slova, která jsou v obou původních jazycích.

Věta 3.12

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci průniku.

Postup konstrukce: Budeme předpokládat, že některé dva regulární jazyky L_1 a L_2 jsou rozpoznávány konečnými automaty
$\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1),\ L_1 = L(\mathcal{A}_1)$ a
$\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2),\ L_2 = L(\mathcal{A}_2),\ Q_1 \cap Q_2 = \emptyset$.

Vytváříme jazyk $L = L_1 \cap L_2$ a automat
$\mathcal{A} = (Q, \Sigma_1 \cap \Sigma_2, \delta, q_0, F),\ L = L(\mathcal{A})$.

Potřebujeme, aby automat $\mathcal{A}$ rozpoznával právě ta slova, která rozpoznávají automaty $\mathcal{A}_1$ a $\mathcal{A}_2$. Toho docílíme jednoduše tak, že totéž slovo necháme paralelně zpracovávat oba původní automaty, resp. v automatu $\mathcal{A}$ spustíme paralelní simulaci výpočtu původních automatů. Vyhledáme dvojice přechodů, jeden z automatu $\mathcal{A}_1$, druhý z automatu $\mathcal{A}_2$, takové, že oba tyto přechody lze použít ve stejné situaci (v našem případě při stejném vstupním signálu). Takže pokud v prvním automatu máme přechod $\delta_1(r, a) = s$ a v druhém $\delta_2(u, a) = v$, pak v automatu $\mathcal{A}$ bude přechod $\delta([r, u], a) = [s, v]$. Co z toho vyplývá?

- budeme chtít, aby na vstupu automatu byly *deterministické* automaty,
- množinu stavů výsledného automatu bude tvořit množina uspořádaných dvojic takových, že první prvek dvojice je stav z Q_1, druhý prvek je stav z Q_2.

Množinou všech takových uspořádaných dvojic by byl kartézský součin množin Q_1 a Q_2, ovšem je otázkou, zda opravdu budeme potřebovat všechny prvky tohoto kartézského součinu. Obecně ano, nicméně některé jeho prvky (nové stavy) budou v automatu $\mathcal{A}$ nedosažitelné či nadbytečné, tedy v reálu půjde pravděpodobně o podmnožinu kartézského součinu. Shrňme, jak bude výsledný automat vypadat:

- množina stavů je množinou uspořádaných dvojic:
  $Q = Q_1 \times Q_2 = \{[x, y]\ ;\ x \in Q_1,\ y \in Q_2\}$
  (simulujeme vlevo výpočet automatu $\mathcal{A}_1$, vpravo automatu $\mathcal{A}_2$, a to paralelně),
- počáteční stav je uspořádaná dvojice, jejíž oba prvky jsou počátečními stavy v původních automatech: $q_0 = [q_0^1, q_0^2]$ (v obou simulovaných automatech začínáme zároveň)
- množina koncových stavů bude podmnožinou množiny Q obsahující jen ty dvojice původních stavů, které jsou v automatech $\mathcal{A}_1$ a $\mathcal{A}_2$ koncové:
  $F = F_1 \times F_2 = \{[x, y]\ ;\ x \in F_1,\ y \in F_2\}$
  (v obou simulovaných automatech končíme zároveň)
- přechodová funkce δ vychází z přechodových funkcí δ_1 a δ_2, zde právě vidíme použitý paralelismus:
  $\delta([x, y], a) = [u, v],$ kde $\delta_1(x, a) = u,\ \delta_2(y, a) = v,\ a \in \Sigma_1 \cap \Sigma_2$

□

Příklad 3.14

Provedeme průnik následujících dvou jazyků, potřebujeme k nim ekvivalentní *deterministické* automaty:

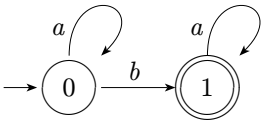
$L_1 = \{a^i b a^j \ ; \ i,j \geq 0\} = a^*ba^* \quad \mathcal{A}_1 = (\{0,1\},\ \{a,b\},\ \delta_1,\ 0,\ \{1\})$

$L_2 = \{(ab)^i aa^j \ ; \ i,j \geq 0\} = (ab)^*aa^* \quad \mathcal{A}_2 = (\{2,3,4\},\ \{a,b\},\ \delta_2,\ 2,\ \{3,4\})$

Aby se nám zápis zbytečně neprodlužoval, zvolíme co nejkratší pojmenování stavů, například podle čísel. Automaty pro jazyky L_1 a L_2:

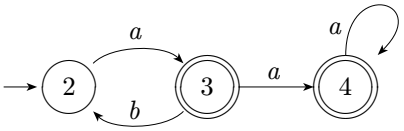
$\mathcal{A}_1$	a	b
$\to 0$	0	1
$\leftarrow 1$	1	

$\delta_1(0,a) = 0$
$\delta_1(0,b) = 1$
$\delta_1(1,a) = 1$



$\mathcal{A}_2$	a	b
$\to 2$	3	
$\leftarrow 3$	4	2
$\leftarrow 4$	4	

$\delta_2(2,a) = 3$
$\delta_2(3,a) = 4$
$\delta_2(3,b) = 2$
$\delta_2(4,a) = 4$

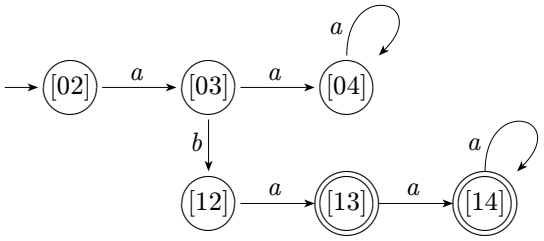


Množina stavů bude množinou uspořádaných dvojic původních stavů, ale z důvodu zkrácení a zjednodušení zápisu nebudeme psát čárku oddělující prvky uspořádané dvojice (zde si to můžeme dovolit, protože jako původní stavy máme čísla, která jsou všechna jednociferná). Sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, [02], F)$, kde

- $Q = Q_1 \times Q_2 = \{[02],[03],[04],[12],[13],[14]\}$
- $F = F_1 \times F_2 = \{[13],[14]\}$
- přechodová funkce δ:

$\mathcal{A}$	a	b
$\to [02]$	[03]	
[03]	[04]	[12]
[04]	[04]	
[12]	[13]	
$\leftarrow$ [13]	[14]	
$\leftarrow$ [14]	[14]	

$\delta([02],a) = [03]$
$\delta([03],a) = [04]$
$\delta([03],b) = [12]$
$\delta([04],a) = [04]$
$\delta([12],a) = [13]$
$\delta([13],a) = [14]$
$\delta([14],a) = [14]$



Stav [04] je, jak vidíme, nadbytečný, proto by nebyl problém ho odstranit. Jazyk automatu $\mathcal{A}$ je $L(\mathcal{A}) = \{abaa^j \ ; \ j \geq 0\} = abaa^*$

Poznámka:

Třída jazyků rozpoznávaných konečnými automaty je uzavřena i vzhledem k dalším operacím, ale těmi se v tomto semestru nebudeme zabývat.

Kapitola 4

Formální gramatiky

Až dosud jsme se zabývali možnostmi zjišťování, zda zadané slovo nebo posloupnost signálů vyhovuje daným podmínkám, tedy zda patří do určitého jazyka. V této kapitole se budeme zabývat možnostmi generování slov vyhovujících daným podmínkám, tedy patřících do určitého jazyka.

4.1 Gramatika a generování slov jazyka

Gramatika je algebraická struktura (i když tak možná nevypadá) popisující strukturu slov daného jazyka, tedy způsob, jakým jsou slova jazyka poskládána z prvků abecedy.

Jaký je rozdíl mezi automatem a gramatikou?

- Automat je *rozhodovací mechanismus* – pomocí automatu určujeme, zda slovo, které dostaneme na vstup, patří či nepatří do jazyka rozpoznávaného automatem.
- Gramatika je *generativní mechanismus* – nemá žádný vstup, sama generuje (vytváří) slova daného jazyka.

V první kapitole jsme si již gramatiky představili (od strany 1.1.2), tedy víme, že kromě *terminálních symbolů* (které odpovídají tomu, co u automatů máme jako abecedu) používáme neterminální symboly (obdobu proměnných).

Zatímco u automatů je „akční součástí definice“ přechodová funkce, u gramatik tuto roli plní množina pravidel. Pravidla jsou ve tvaru $\alpha \to \beta$, používáme je tak, že ve zpracovávaném řetězci hledáme podřetězec α a nahradíme jej podřetězcem β.

Vytvoření gramatiky si nejdřív ukážeme na příkladu, následují definice.

Příklad 4.1

Pro jazyk zadaný tímto regulárním výrazem sestrojíme gramatiku, která jej generuje:
$L = a^*(bc + c\,b^*c)$

Všimněme si nejdřív struktury tohoto výrazu. Jeho první část obsahuje sekvenci symbolů a, pak následuje druhá část, která může mít dvě varianty – buď se jedná o řetězec bc nebo o dva symboly c, mezi nimiž může být jakýkoliv počet symbolů b.

Gramatiku můžeme navrhnout několika různými způsoby – pro jeden jazyk může existovat i více odlišných gramatik, z nichž každá má trochu jiné vlastnosti. V našem případě můžeme

postupovat třeba takto:

- Vytvoříme neterminální symbol A, který použijeme pro generování první části slova obsahující symboly a:
  $A \to aA$
  $A \to \varepsilon$

  Tato dvojice pravidel určuje rekurzi (symbol A v řetězci opakovaně nahrazujeme dvojicí symbolů aA, čímž směrem doleva narůstá počet symbolů a) a zastavení rekurze (když jsme prvním pravidlem vytvořili dostatek symbolů a, odstraníme symbol A z řetězce a tím rekurzi zastavíme).
- Podobně pro symboly b:
  $B \to bB$
  $B \to \varepsilon$
- Pravidlo vytvářející první variantu druhé části slova:
  $H \to bc$
- Pravidlo vytvářející druhou variantu druhé části slova, napojíme na pravidla generující posloupnost symbolů b:
  $C \to cBc$
- A teď to všechno propojíme – neterminální symbol S bude představovat celé výsledné slovo, v pravidlech pro tento symbol určíme, že existují dvě varianty (obě začínají symboly a, ale v první následuje jeden podřetězec a v druhé variantě jiný):
  $S \to AH$
  $S \to AC$

Všechna vytvořená pravidla nyní shrneme do jediné gramatiky a očíslujeme:

$S \to AH$	①
$S \to AC$	②
$A \to aA$	③
$A \to \varepsilon$	④
$H \to bc$	⑤
$C \to cBc$	⑥
$B \to bB$	⑦
$B \to \varepsilon$	⑧

Úspornější způsob zápisu je takový, kde více pravidel se stejnou levou stranou (tj. přepisujících tentýž symbol) umístíme na jeden řádek:

$S \to AH \mid AC$	①, ②
$A \to aA \mid \varepsilon$	③, ④
$H \to bc$	⑤
$C \to cBc$	⑥
$B \to bB \mid \varepsilon$	⑦, ⑧

Generování slova probíhá tak, že začneme řetězcem obsahujícím jediný neterminální symbol (startovací symbol gramatiky), obvykle je označen S, a v každém kroku odvození tento řetězec měníme – vybereme některý symbol řetězce (neterminální, v prvním kroku nemáme na výběr,

tedy vybereme S) a zpracujeme ho podle některého pravidla gramatiky. Končíme tehdy, když „není co zpracovat“, tedy žádný neterminální symbol v řetězci nenajdeme. Generování podle výše uvedené gramatiky může vypadat například takto:
$S \Rightarrow AC \Rightarrow aAC \Rightarrow aaAC \Rightarrow aaC \Rightarrow aacBc \Rightarrow aacbBc \Rightarrow aacbc$

Jak jsme postupovali?

- V prvním kroku jsme použili pravidlo $S \to AC$ (vybrali jsme jedno ze dvou pravidel pro symbol S).
- V druhém kroku jsme v řetězci AC přepsali symbol A podle pravidla číslo ③ ($A \to aA$), tedy řetězec se změnil na aAC (to, co zrovna nezpracováváme, jen přepíšeme do další „generace“).
- Totéž pravidlo jsme použili ještě jednou (takže v řetězci máme dvakrát symbol a), ale pak jsme ukončili rekurzi pravidlem číslo ④ (epsilonovým), čímž jsme symbol A z řetězce odstranili.
- Následovalo zpracování symbolu C pravidlem číslo ⑥ (pro tento symbol nemáme na vybranou) a nakonec pravidly pro symbol B jsme vygenerovali jeden symbol b.

Pro tentýž jazyk je možné vytvořit i jiné gramatiky (vlastně i ta výše uvedená by se dala mírně upravit, přičemž by pořád generovala tentýž jazyk). Následující gramatika používá jednodušší tvar pravidel, ale generuje tentýž jazyk:

$S \to aS \mid bA \mid cB$	①,②,③
$A \to c$	④
$B \to bB \mid c$	⑤,⑥

V této gramatice vygenerujeme totéž slovo jako v předchozí:
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacB \Rightarrow aacbB \Rightarrow aacbc$

Obě gramatiky sice generují tentýž jazyk, ale každá má jiné vlastnosti. Ta první má přehlednější strukturu, snadněji se navrhuje. Druhá nemá tak přehlednou strukturu, ale zato má kratší pravidla, pravidel je méně, a při generování postupujeme striktně zleva doprava (tak, jak čteme text nebo načítáme soubor na počítači).

Definice 4.1 (Formální gramatika)

Formální gramatika je uspořádaná čtveřice (posloupnost) $G = (N, T, P, S)$, *kde*

- N *je neprázdná konečná množina neterminálních symbolů (neterminálů) – neterminální abeceda,*
- T *je neprázdná konečná množina terminálních symbolů (terminálů) – terminální abeceda, platí* $N \cap T = \emptyset$,
- P *je neprázdná konečná množina pravidel,* $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$
  jinak: $\alpha A \beta \to \gamma$, *kde* $A \in N$, $\alpha, \beta, \gamma \in (N \cup T)^*$
- S *je startovací symbol gramatiky,* $S \in N$.

V definici se může zdát trochu nesrozumitelné určení množiny pravidel. Ve výrazu, který je na tom místě uveden, označení $(N \cup T)^*$ znamená „jakkoliv dlouhá posloupnost jakýchkoliv symbolů“ (neterminálů i terminálů). Symbol pro kartézský součin $\times$ odděluje levou a pravou část pravidla. Celý výraz $(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ znamená:

- $(N \cup T)^* N (N \cup T)^*$ – na levé straně výrazu („před šipkou“) je alespoň jeden neterminál obklopený (z obou stran) jakkoliv dlouhými řetězci neterminálů a terminálů (v každém případě bude před šipkou alespoň ten jeden neterminál),
- $(N \cup T)^*$ – na pravé straně výrazu může být jakýkoliv řetězec (včetně prázdného řetězce, jak jsme viděli v předchozím příkladu).

V definici máme uveden obecný tvar pravidel, kde na levé straně pravidla může být jakýkoliv řetězec (obsahující alespoň jeden neterminál), nicméně v příkladu jsme použili pouze jednodušší tvar pravidel, kde na levé straně máme pouze ten jeden „vyžadovaný“ neterminál.

Příklad 4.2

V předchozím příkladu jsou dvě různé gramatiky. Jejich plný zápis je následující:

$G_1 = (\{S, A, H, C, B\}, \{a, b, c\}, P, S)$, kde P je
$S \to AH \mid AC$
$A \to aA \mid \varepsilon$
$H \to bc$
$C \to cBc$
$B \to bB \mid \varepsilon$

$G_2 = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P je
$S \to aS \mid bA \mid cB$
$A \to c$
$B \to bB \mid c$

Všimněte si, že tak, jako jsme u automatů při zápisu přechodové funkce museli uvést plnou specifikaci, aby bylo jasné, který stav je počáteční, které jsou koncové apod., u gramatiky je třeba také uvést plnou specifikaci (zde $G = \ldots$), abychom věděli, který symbol je startovací, případně které z použitých symbolů jsou terminální.

Další definice určují, jak mají být používána pravidla, která jsme definovali. Tyto definice jsou obdobou definic přechodu mezi konfiguracemi a výpočtu slova, které jsme viděli u konečných automatů.

Definice 4.2 (Relace kroku odvození)

Nechť $w_1, w_2 \in (N \cup T)^*$ jsou řetězce nad abecedou $N \cup T$. Tyto řetězce jsou v relaci $\Rightarrow_G$ v gramatice $G = (N, T, P, S)$, pokud existuje pravidlo $(\alpha \to \beta) \in P$ a zároveň $w_1 = x_1 \alpha x_2$, $w_2 = x_1 \beta x_2$. Zapisujeme: $w_1 \Rightarrow_G w_2$. Říkáme, že slovo w_2 lze přímo (v jednom kroku) odvodit ze slova w_1 použitím pravidla $\alpha \to \beta$.

Pokud je zřejmé, o kterou gramatiku se jedná, můžeme místo $\Rightarrow_G$ psát jen $\Rightarrow$.

Reflexivní a tranzitivní uzávěr relace kroku odvození $\Rightarrow$ zapisujeme $\Rightarrow^*$, proto například posloupnost $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow w_4$ můžeme zapsat zkráceně jako $w_1 \Rightarrow^* w_4$, případně s násobičem jako $w_1 \Rightarrow^3 w_4$ (tři kroky). Obdobně *tranzitivní uzávěr* této relace zapisujeme $\Rightarrow^+$.

Definice 4.3 (Derivace)

Derivace (odvození) délky n slova (věty) α v gramatice $G = (N, T, P, S)$ je posloupnost řetězců α_1, α_2, ... α_n nad abecedou $(N \cup T)$ taková, že

- $\alpha_1 = S$,
- $\alpha_n = \alpha,\ \alpha \in T^*$,
- $\alpha_i \Rightarrow \alpha_{i+1}\ \forall i:\ 1 \leq i \leq n-1$.

Jinak zapsáno: $S \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha$

Obdobou *konfigurace* u automatů je při odvození v gramatice vždy ten *řetězec*, který v daném kroku máme zpracovat některým pravidlem. Relace kroku odvození je pak relací mezi jedním takovým řetězcem a následujícím řetězcem, který získáme uplatněním některého pravidla gramatiky. Řetězce, které získáme postupným odvozováním ze startovacího symbolu, nazýváme větnými formami, poslední člen této posloupnosti se nazývá věta (případně slovo) a je složen z terminálů.

Definice 4.4 (Větná forma, věta)

Větná forma *gramatiky* $G = (N, T, P, S)$ *je kterékoliv slovo* $\alpha \in (N \cup T)^*$ *takové, že* $S \Rightarrow_G^* \alpha$, *je to kterékoliv slovo, které lze odvodit ze startovacího symbolu pomocí pravidel gramatiky.*

Věta *gramatiky* $G = (N, T, P, S)$ *je taková větná forma* w, *která se skládá pouze z terminálních symbolů, tedy* $S \Rightarrow_G^* w,\ w \in T^*$. *Můžeme říci, že jazyk generovaný gramatikou je množina všech vět této gramatiky.*

Příklad 4.3

V příkladu ze strany 67 máme dvě derivace téhož slova, každou v jiné gramatice:
$S \Rightarrow AC \Rightarrow aAC \Rightarrow aaAC \Rightarrow aaC \Rightarrow aacBc \Rightarrow aacbBc \Rightarrow aacbc$
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacB \Rightarrow aacbB \Rightarrow aacbc$

Poznámka:

Dávejte si pozor na správné značení – jednoduchá šipka „$\rightarrow$“ se používá při zápisu pravidel, dvojitá šipka „$\Rightarrow$“ je relace, používáme ji při zápisu derivace (odvození) slova (propojuje jednotlivé větné formy v derivaci).

Definice 4.5 (Jazyk gramatiky)

Jazyk generovaný gramatikou $G = (N, T, P, S)$ *je množina všech slov, která lze v gramatice vygenerovat ze startovacího symbolu:*

$$L(G) = \{w \in T^*\ ;\ S \Rightarrow_G^* w\} \tag{4.1}$$

Definice 4.6 (Ekvivalence gramatik, ekvivalence automatu a gramatiky)

Gramatiky G_1 a G_2 jsou (jazykově) ekvivalentní, pokud generují tentýž jazyk, tedy $L(G_1) = L(G_2)$.

Gramatika G je (jazykově) ekvivalentní automatu $\mathcal{A}$, pokud gramatika generuje jazyk rozpoznávaný automatem, tedy platí $L(G) = L(\mathrm{A})$.

Automatem zmíněným v předchozí definici může být míněn konečný automat, zásobníkový automat nebo třeba Turingův stroj. Ekvivalence může být i na vyšší úrovni – například třída jazyků rozpoznávaných konečnými automaty může být ekvivalentní s třídou jazyků generovaných gramatikami v určité formě. S pojmem „třída jazyků“ jsme se seznámili v předchozí kapitolo o konečných automatech, v sekci o uzávěrových operacích.

4.2 Chomského hierarchie

Po jazykovědci Noamu Chomském je pojmenována hierarchie základních typů formálních gramatik a tříd jazyků. Gramatiky v této hierarchii mají jedno společné – *sekvenční* způsob generování slova. To znamená, že v každém kroku odvození je použito právě jedno pravidlo na právě jednom místě v přepisovaném slově.

V hierarchii najdeme čtyři typy gramatik označených čísly 0, 1, 2, 3. *Třídy jazyků* generované těmito gramatikami označujeme $\mathscr{L}(0), \mathscr{L}(1), \mathscr{L}(2), \mathscr{L}(3)$, přičemž pro některé z nich se používá i jiné označení. Mimo hierarchii, ale v těsné vazbě na ni, existují další typy gramatik, na které se také podíváme.

4.2.1 Gramatiky v Chomského hierarchii

Chomského hierarchie zahrnuje tedy čtyři typy gramatik. U každého typu si uvedeme případný slovní název, tvar pravidel a ekvivalentní stroj, který rozpoznává stejnou třídu jazyků.

Definice 4.7 (Gramatiky typu 0 v Chomského hierarchii)

Gramatiky typu 0 (rekurzívně vyčíslitelné, RE – Recursively Enumerable) jsou gramatiky používající obecný tvar pravidel:

$$(N \cup T)^* N (N \cup T)^* \times (N \cup T)^* \tag{4.2}$$

Jiný zápis pravidel je následující: $\alpha A \beta \to \gamma, \quad A \in N,\ \alpha, \beta, \gamma \in (N \cup T)^*$

Strojem ekvivalentním ke gramatikám typu 0 je Turingův stroj (TS). Jazyky rozpoznávané gramatikami typu 0 nazýváme jazyky typu 0, třídu jazyků typu 0 značíme $\mathscr{L}(0)$, resp. $\mathscr{L}(RE)$.

Definice 4.8 (Gramatiky typu 1 v Chomského hierarchii)

Gramatiky typu 1 (nezkracující, monotónní, rekurzivní) jsou gramatiky, jejichž všechna pravidla zachovávají omezení pro pravidla gramatik typu 0 (min. neterminál vlevo) a zároveň jsou v tomto tvaru:

$$\alpha \to \beta, \quad |\alpha| \leq |\beta|, \quad \alpha, \beta \in (N \cup T)^* \tag{4.3}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Strojem ekvivalentním ke gramatikám typu 1 je lineárně ohraničený automat (LOA, Linearly Bounded Automaton – LBA). Jazyky rozpoznávané gramatikami typu 1 jsou jazyky typu 1, třídu jazyků typu 1 značíme $\mathscr{L}(1)$.

Gramatiky typu 1 nazýváme nezkracující, protože generované slovo se po jednotlivých krocích buď prodlužuje, nebo se jeho délka nemění, ale nezkracuje se – díky tomu, že jediné „zkracující“ (tedy epsilonové) pravidlo je možné použít pouze a jenom v prvním kroku výpočtu (pokud chceme vygenerovat prázdné slovo). Formulace „a zároveň se nenachází na pravé straně žádného pravidla“ totiž znamená právě to – pokud symbol S nemáme na pravé straně žádného pravidla, nemůže být v žádné větné formě kromě té první, ze které začíná výpočet.

Definice 4.9 (Gramatiky typu 2 v Chomského hierarchii)

Gramatiky typu 2 (bezkontextové, context-free – CF) jsou gramatiky, jejichž všechna pravidla jsou v tomto tvaru:

$$A \to \beta, \quad A \in N,\ \beta \in (N \cup T)^* \tag{4.4}$$

Strojem ekvivalentním ke gramatikám typu 2 je zásobníkový automat (Pushdown automaton). Jazyky rozpoznávané gramatikami typu 2 jsou jazyky typu 2, tedy bezkontextové jazyky, třídu jazyků typu 2 značíme $\mathscr{L}(2)$, resp. $\mathscr{L}(CF)$.

O gramatikách typu 2 budeme jednoduše hovořit jako o bezkontextových gramatikách, jazyky jimi rozpoznávané jsou bezkontextové jazyky.

Definice 4.10 (Gramatiky typu 3 v Chomského hierarchii)

Gramatiky typu 3 (regulární, REG) jsou gramatiky, jejichž všechna pravidla jsou v jednom z následujících tvarů:

$$\begin{aligned} A &\to aB, \qquad A, B \in N,\ a \in T \\ A &\to a, \qquad A \in N,\ a \in T \end{aligned} \tag{4.5}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Strojem ekvivalentním ke gramatikám typu 3 je konečný automat (Finite automaton). Jazyky rozpoznávané gramatikami typu 3 jsou jazyky typu 3, tedy regulární jazyky, třídu jazyků typu 3 značíme $\mathscr{L}(3)$, resp. $\mathscr{L}(REG)$.

O gramatikách typu 3 budeme jednoduše hovořit jako o regulárních gramatikách, jazyky jimi rozpoznávané jsou regulární jazyky.

Poznámka:

V prvním příkladu této kapitoly máme dvě gramatiky – teď už víme, že první z nich je bezkontextová, druhá regulární.

Regulární gramatiky striktně zachovávají způsob generování slova zleva doprava, což také odpovídá činnosti konečného automatu (ten taky zpracovává slovo ze vstupu výhradně zleva doprava, v každém kroku zpracuje jeden symbol, podobně regulární pravidla generují v každém kroku jeden symbol). Naproti tomu bezkontextové gramatiky popisují i složitější vnitřní strukturu generovaného výrazu.

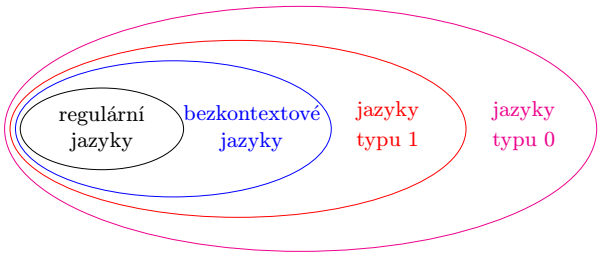
Regulárními gramatikami a jejich vztahem ke konečným automatům se budeme zabývat v následující kapitole. V tomto semestru se také podíváme na bezkontextové gramatiky a zásobníkové automaty, nicméně jen v základu. Další související témata budeme probírat v navazujícím předmětu (Teorie jazyků a automatů II nebo Základy teoretické informatiky II).

Věta 4.1 (Vztah mezi jazyky gramatik v Chomského hierarchii)

Mezi třídami jazyků generovaných gramatikami v Chomského hierarchii jsou tyto vztahy (určující generativní sílu jednotlivých gramatik):

$$\mathscr{L}(3) \subset \mathscr{L}(2) \subset \mathscr{L}(1) \subset \mathscr{L}(0) \tag{4.6}$$

Předchozí věta nám říká, že každý regulární jazyk je zároveň bezkontextovým (a taky typu 1 a typu 0), každý bezkontextový je zároveň typu 1 (a taky typu 0), každý jazyk typu 1 je také typu 0. Vztahy mezi těmito třídami jazyků můžeme vyjádřit i množinově:



Obrázek 4.1: Vztahy mezi jazyky v Chomského hierarchii

Poznámka:

Pozor, tento vztah platí pro jazyky, nikoliv pro gramatiky. Uvědomte si, že pro jeden jazyk je možné sestrojit několik různých gramatik (které každá mohou být jiného typu), a například pravidla bezkontextových gramatik nejsou obecně nezkracující, tedy ne všechny bezkontextové gramatiky by mohly být zároveň gramatikami typu 1.

Náznak důkazu (Věta 4.1): Pro korektní důkaz tohoto tvrzení ještě nemáme dostatek znalostí, jen si naznačíme, jak by takový důkaz vypadal, a jednodušší části důkazu provedeme. U každé vlastní inkluze $\subset$ je třeba nejdřív dokázat samotnou inkluzi $\subseteq$ a potom najít jazyk, který dokazuje, že jde o vlastní inkluzi. Postupně:

- V případě $\mathscr{L}(3) \subset \mathscr{L}(2)$

  - $\mathscr{L}(3) \subseteq \mathscr{L}(2)$ plyne z tvaru pravidel, protože každé regulární pravidlo zároveň splňuje podmínky pro bezkontextová pravidla, tedy pokud pro jazyk lze sestrojit regulární gramatiku, je možné sestrojit i bezkontextovou,
  - například jazyk $L_1 = \{a^n b^n \ ; \ n \geq 0\}$ je bezkontextový, ale není regulární (což zatím neumíme dokázat): $L_1 \in \mathscr{L}(2) - \mathscr{L}(3)$,

  $\Rightarrow \mathscr{L}(3) \subset \mathscr{L}(2)$.
- V případě $\mathscr{L}(2) \subset \mathscr{L}(1)$
  - $\mathscr{L}(3) \subseteq \mathscr{L}(2)$ – každou bezkontextovou gramatiku lze převést do tvaru nezkracující bezkontextové gramatiky (což ještě neumíme), pak platí obdobné tvrzení jako v předchozím případě (plynulo by z tvaru pravidel),
  - například jazyk $L_2 = \{a^n b^n c^n \ ; \ n \geq 0\}$ je typu 1, ale není bezkontextový (což zatím neumíme dokázat): $L_2 \in \mathscr{L}(1) - \mathscr{L}(2)$,

  $\Rightarrow \mathscr{L}(2) \subset \mathscr{L}(1)$.
- V případě $\mathscr{L}(1) \subset \mathscr{L}(0)$
  - $\mathscr{L}(1) \subseteq \mathscr{L}(0)$ je zřejmé, protože $\mathscr{L}(0)$ obsahuje všechny rekurzívně vyčíslitelné jazyky (tj. všechny jazyky, ke kterým dokážeme sestrojit jakoukoliv funkční gramatiku), tedy rozhodně obsahuje i jazyky, pro které lze sestrojit nezkracující gramatiku,
  - to je již složitější, ale důkaz můžeme provést pomocí automatů – pokud dokážeme ekvivalenci mezi gramatikami typu 1 a LBA, pak stačí dokázat, že existují jazyky, které LBA nerozpoznává,

  $\Rightarrow \mathscr{L}(1) \subset \mathscr{L}(0)$. □

Definice 4.11 (Ekvivalence gramatik a tříd jazyků)

Gramatiky G_1 a G_2 jsou ekvivalentní, pokud generují tentýž jazyk: $L(G_1) = L(G_2)$.

Třídy jazyků $\mathcal{T}_1$ a $\mathcal{T}_2$ jsou ekvivalentní, jestliže jsou množinově ekvivalentní (protože se jedná o množiny jazyků). Zapisujeme $\mathcal{T}_1 \cong \mathcal{T}_2$.

4.2.2 Související typy gramatik

Rozlišujeme více různých druhů gramatik, které buď odpovídají některému z typů gramatik Chomského hierarchie, nebo jsou někde „mezi“ různými typy.

Definice 4.12 (Kontextové gramatiky)

Kontextové gramatiky (Context-sensitive, CS) jsou gramatiky, jejichž všechna pravidla odpovídají předpisu

$$\alpha A \beta \to \alpha \gamma \beta, \quad A \in N, \ \alpha, \beta, \gamma \in (N \cup T)^*, \ \gamma \neq \varepsilon \tag{4.7}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Jazyky rozpoznávané kontextovými gramatikami nazýváme kontextovými jazyky, třídu těchto jazyků značíme $\mathscr{L}(CS)$.

Pravidla $\alpha A\beta \to \alpha\gamma\beta$ uvedená v předchozí definici *zachovávají kontext*. To znamená, že sice ve skutečnosti přepisujeme symbol A na řetězec γ (provádíme přepis $A \to \gamma$), ale jen tehdy, pokud je tento symbol A obklopen stanoveným kontextem – okolím (levým kontextem α, pravým kontextem β), a tento kontext zůstane tak jak je i po přepisu.

Věta 4.2

Výpočetní síla třídy kontextových jazyků Třída jazyků $\mathscr{L}(CS)$ je ekvivalentní třídě jazyků typu 1, tedy platí $\mathscr{L}(CS) \cong \mathscr{L}(1)$.

Důkaz tohoto tvrzení necháme na příští semestr.

Poznámka:

V některých zdrojích jsou kontextové gramatiky přímo ztotožňovány s gramatikami typu 1 místo gramatik nezkracujících.

Definice 4.13 (Lineární gramatiky)

Lineární gramatiky LIN jsou gramatiky, jejichž všechna pravidla odpovídají předpisu

$$\begin{aligned} &A \to \alpha B\beta, \\ &A \to \alpha, \qquad A, B \in N,\ \alpha, \beta \in T^* \end{aligned} \tag{4.8}$$

Pokud jsou všechna pravidla gramatiky ve tvaru $A \to \alpha B$ nebo $A \to \alpha$, $A, B \in N,\ \alpha \in T^*$, pak je tato gramatika pravolineární.

Pokud jsou všechna pravidla gramatiky ve tvaru $A \to B\beta$ nebo $A \to \beta$, $A, B \in N,\ \beta \in T^*$, pak je tato gramatika levolineární.

Třídu jazyků generovaných lineárními gramatikami označíme $\mathscr{L}(LIN)$, pravolineárními gramatikami $\mathscr{L}(RLIN)$, levolineárními gramatikami $\mathscr{L}(LLIN)$.

Lineární gramatika je tedy taková gramatika, která má na levé straně pravidla pouze neterminál (a nic jiného, tak jako u bezkontextových nebo regulárních gramatik) a na pravé straně pravidla nejvýše jeden neterminál a jakýkoliv počet terminálů.

Tvar pravidel typu pravolineární velmi připomíná tvar regulárních pravidel, až na to, že zatímco u regulárních pravidel lze mít na pravé straně jen jeden terminál, u pravolineárních zde můžeme mít řetězec terminálů.

Věta 4.3 (Vztah mezi pravo- a levolineárními a regulárními gramatikami)

Třídy jazyků generovaných pravolineárními, levolineárními a regulárními gramatikami jsou navzájem ekvivalentní:

$$\mathscr{L}(RLIN) \cong \mathscr{L}(LLIN) \cong \mathscr{L}(REG) \tag{4.9}$$

Náznak důkazu: Nejdřív se zaměříme na vztah mezi pravolineárními a levolineárními gramatikami. Stačí si uvědomit, že pravolineární gramatiky generují slovo zleva doprava (tak jako regulární gramatiky), kdežto levolineární v opačném směru.

V této části důkazu nepůjdeme do podrobností (sestrojení pravolineární gramatiky podle levolineární nebo opačně by znamenalo „překopání“ celé gramatiky); z konstrukčního hlediska by bylo jednodušší sestrojit výslednou gramatiku od začátku, nevycházet z původní. Nicméně je možné dokázat, že $\mathscr{L}(RLIN) \cong \mathscr{L}(LLIN)$.

Nyní ke vztahu mezi pravolineárními a regulárními gramatikami. Je zřejmé, že regulární gramatiky můžeme chápat jako speciální případ pravolineárních, kdy jen „zostříme“ požadavky na tvar pravidel, tedy vztah $\mathscr{L}(REG) \subseteq \mathscr{L}(RLIN)$ je triviální.

Platí i opačný vztah? Pravolineární pravidlo může být v jednom z těchto tvarů:

1. $A \to \alpha B$ nebo $A \to \alpha,\ A, B \in N,\ \alpha \in T^*,\ |\alpha| = 1$ (právě jeden terminál)
2. $A \to \alpha B,\ A, B \in N,\ \alpha \in T^*,\ |\alpha| \geq 2$
3. $A \to \alpha,\ A, \in N,\ \alpha \in T^*,\ |\alpha| \geq 2$
4. $A \to B,\ A, B \in N$ (tj. $|\alpha| = 0$)
5. $A \to \varepsilon,\ A \in N$

ad. 1. Pravidla podle prvního bodu odpovídají regulárním pravidlům, nemusíme řešit.

ad. 2. Pravidla podle druhého bodu sice ne, ale není problém takové pravidlo upravit, resp. nahradit množinou jiných pravidel tak, aby byl generován tentýž řetězec:

Původní pravidlo: $A \to a_1 a_2 \dots a_n B, \quad A, B \in N,\ a_i \in T,\ 1 \leq i \leq n$
Nahradíme těmito pravidly:
$A \to a_1 X_1$
$X_1 \to a_2 X_2$
$\dots$
$X_{n-2} \to a_{n-1} X_{n-1}$
$X_{n-1} \to a_n B$
kde symboly $X_1, \dots, X_{n-1}$ jsou nové, dosud nepoužité.

Například pravidlo $A \to abcM$ bychom nahradili sadou regulárních pravidel
$A \to aX_1$
$X_1 \to bX_2$
$X_2 \to cM$

Můžeme si ověřit, že ekvivalence výstupu zůstává zachována. Odvození $A \Rightarrow abcM$ podle původního pravidla by se podle nové sady pravidel protáhlo na $A \Rightarrow aX_1 \Rightarrow abX_2 \Rightarrow abcM$ (tři kroky místo jednoho), nicméně výsledek je tentýž.

ad. 3. Tento případ je podobný předchozímu – místo přímého vygenerování celého terminálního řetězce generujeme postupně jednotlivé terminály.

Pravidlo $A \to a_1 a_2 \dots a_n, \quad A \in N,\ a_i \in T,\ 1 \leq i \leq n$

Nahradíme těmito pravidly:
$A \to a_1 X_1$
$X_1 \to a_2 X_2$
$\dots$

$X_{n-2} \to a_{n-1}X_{n-1}$
$X_{n-1} \to a_n$
kde symboly $X_1, \ldots, X_{n-1}$ jsou nové, dosud nepoužité (pozor, nesmí být použity ani při podobné úpravě jiného pravidla).

ad. 4. Pravidlo ve tvaru $A \to B,\ A, B \in N$ nazýváme jednoduchým pravidlem (později se těmto pravidlům budeme věnovat více). Je zřejmé, že tato pravidla ve skutečnosti nic terminálního negenerují, tedy je možné se jich jednoduchou úpravou pravidel zbavit. Předpokládejme, že máme (kromě jiných) tato pravidla:

$A \to \alpha_1 \mid \ldots \mid \alpha_k \mid B$
$B \to \beta_1 \mid \ldots \mid \beta_m$

(tj. pro symbol A kromě toho pravidla, které nám „vadí“, ještě k jiných pravidel, a pak m pravidel pro symbol B). Pak je nahradíme těmito pravidly:

$A \to \alpha_1 \mid \ldots \mid \alpha_k \mid \beta_1 \mid \ldots \mid \beta_m$
$B \to \beta_1 \mid \ldots \mid \beta_m$

S pravidly pro symbol B jsme nehýbali, jen pravidlo $A \to B$ jsme nahradili pravidly pro symbol B. Pak bychom například místo odvození $A \Rightarrow B \Rightarrow \beta_2$ měli (po zmíněné úpravě) odvození $A \Rightarrow \beta_2$. Tedy odstranění jednoduchých pravidel znamená zkrácení odvozování.

ad. 5. Tento případ je podobný jako předchozí. Nejdřív předpokládejme, že přepisovaný symbol A není startovacím symbolem gramatiky. Pak bychom pravidla ve tvaru

$A \to \alpha_1 \mid \ldots \mid \alpha_k \mid \varepsilon$
$B \to \beta A \mid \ldots$

nahradili těmito pravidly:

$A \to \alpha_1 \mid \ldots \mid \alpha_k$
$B \to \beta A \mid \beta \mid \ldots$

Odstranili jsme epsilonové pravidlo $A \to \varepsilon$, ale abychom nezasáhli do jazyka generovaného gramatikou, přidali jsme pravidla „simulující“ použití tohoto pravidla. A se vyskytuje na pravé straně pravidla $B \to \beta A$, proto přidáme pravidlo $B \to \beta$ nahrazující případ, kdy bychom vygenerovaný symbol A přepsali epsilonovým pravidlem. Tedy místo odvození $B \Rightarrow \beta A \Rightarrow \beta$ by bylo odvození $B \Rightarrow \beta$.

Úpravu by samozřejmě bylo nutné provést pro všechna pravidla obsahující neterminál A.

Pokud by symbol A byl startovacím symbolem gramatiky, museli bychom zkontrolovat, jestli se nachází na pravé straně některého pravidla. Pokud ne, nebudeme provádět žádnou úpravu (necháme pravidlo $A \to \varepsilon$ tak jak je), pokud ano, provedeme výše naznačenou úpravu, ale navíc přidáme nový startovací symbol (označíme ho třeba S') takto:

$S' \to A \mid \varepsilon$
$A \to \alpha_1 \mid \ldots \mid \alpha_k$
$B \to \beta A \mid \beta \mid \ldots$

A následně provedeme úpravu podle bodu 4.

Po úpravě všech pravidel (původně pravolineární) gramatiky již výsledná gramatika obsahuje pouze pravidla v regulárním tvaru, tedy $\mathscr{L}(REG) \supseteq \mathscr{L}(RLIN)$, proto $\mathscr{L}(REG) \cong \mathscr{L}(RLIN)$. Protože na začátku postupu jsme ukázali, že pravolineární a levolineární gramatiky jsou ekvivalentní, platí tvrzení z uvedené věty. □

Věta 4.4 (Výpočetní síla lineárních gramatik)

Pro třídu jazyků generovaných lineárními gramatikami platí následující:

$$\mathscr{L}(REG) \subset \mathscr{L}(LIN) \subset \mathscr{L}(CF) \tag{4.10}$$

Náznak důkazu: Podobně jako u vztahů v Chomského hierarchii, i zde si jen ukážeme, jak by korektní důkaz vypadal, a jednodušší části důkazu provedeme. Účelem je ukázat, že výpočetní síla lineárních gramatik je mezi výpočetní silou regulárních a bezkontextových gramatik.

U každé vlastní inkluze $\subset$ je třeba nejdřív dokázat samotnou inkluzi $\subseteq$ a potom najít jazyk, který dokazuje, že jde o vlastní inkluzi.

- V případě $\mathscr{L}(REG) \subset \mathscr{L}(LIN)$
  - důkaz, že platí $\mathscr{L}(REG) \subseteq \mathscr{L}(LIN)$ je jednoduchý – u předchozí věty jsme si ukázali, že $\mathscr{L}(REG) \cong \mathscr{L}(RLIN)$ a zároveň víme, že pravolineární gramatiky jsou speciálním případem lineárních gramatik;

    z toho vyplývá, že pro jakýkoliv jazyk generovaný regulární gramatikou dokážeme sestrojit lineární gramatiku (a dokonce každé regulární pravidlo je vlastně lineárním pravidlem, nemusíme konstruovat nic nového),
  - například jazyk $L_1 = \{a^n b^n \ ; \ n \geq 0\}$ není regulární (bylo zmíněno už na straně 75), ale lze jej generovat touto lineární gramatikou:

    $G_{LIN} = (\{S\},\ \{a, b\},\ P,\ S)$ s pravidly $S \to aSb \mid \varepsilon$

  $\Rightarrow\ \mathscr{L}(REG) \subset \mathscr{L}(LIN)$
- V případě $\mathscr{L}(LIN) \subset \mathscr{L}(CF)$
  - důkaz, že platí $\mathscr{L}(LIN) \subseteq \mathscr{L}(CF)$, je opět triviální – je zřejmé podle tvaru pravidel:
    - pravidla v lineárním tvaru: $A \to \alpha B \beta \mid \alpha,\ A, B \in N,\ \alpha, \beta \in (N \cup T)^*$
    - pravidla bezkontextová: $A \to \gamma,\ \gamma \in (N \cup T)^*$

    první uvedená splňují podmínky pro druhá uvedená, tedy každá lineární gramatika je zároveň bezkontextovou gramatikou,
  - například jazyk $L_2 = \{a^i b^i a^j b^j \ ; \ i, j \geq 1\}$ není lineární, ale lze jej generovat touto bezkontextovou gramatikou:

    $G_{CF} = (\{S, A\},\ \{a, b\},\ P,\ S)$ s pravidly $S \to AA,\ A \to aAb \mid ab$

  $\Rightarrow\ \mathscr{L}(LIN) \subset \mathscr{L}(CF)$ □

Definice 4.14 (Konečné jazyky)

Konečný jazyk je jazyk s konečným počtem slov. Třídu konečných jazyků označujeme $\mathscr{L}(FIN)$ (finite languages).

Pokud potřebujeme vytvořit gramatiku generující konečný jazyk, obvykle volíme regulární gramatiku. Co se automatu týče, vhodný je konečný automat (samozřejmě bez smyček – jakákoliv smyčka by znamenala nekonečně mnoho variant rozpoznávaných slov a tedy nekonečný jazyk).

Příklad 4.4

Ukážeme si postup vytvoření gramatiky generující konečný jazyk, použijeme výhradně regulární pravidla. Bude to gramatika pro tento jazyk:
$L = \{if, then, this\}$

Je dobré seřadit si slova jazyka tak, abychom viděli, která slova začínají stejným podřetězcem. V našem případě druhé a třetí slovo začínají podřetězcem *th*. Na straně 42 je pro tento jazyk sestrojen konečný automat, v případě gramatiky budeme postupovat stejně – pravidla budou taková, aby pro jednotlivá slova existovaly různé „větve“, zde větve v odvozování, a u slov začínajících stejným podřetězcem se tyto větve budou v průběhu generování postupně dělit.

Pravidla pro slovo *if*:	Pravidla pro slovo *then*:	Pravidla pro slovo *this*:
$S \to iA$	$S \to tB$	$S \to tB$
$A \to f$	$B \to hC$	$B \to hC$
	$C \to eD$	$C \to iE$
	$D \to n$	$E \to s$

Celá gramatika (každé pravidlo uvedeme jen jednou):
$G = (\{S, A, B, C, D, E\},\ \{i, f, t, h, e, n, s\},\ P,\ S)$, kde v množině P jsou pravidla
$S \to iA \mid tB$
$A \to f$
$B \to hC$
$C \to eD \mid iE$
$D \to n\ E \to s$

Všimněte si, že v pravidlech gramatiky generující konečný jazyk není žádná rekurze – ani být nemůže, jakákoliv rekurze by znamenala generování nekonečného jazyka.

Věta 4.5 (Vztah konečných jazyků k třídám jazyků v Chomského hierarchii)

Pro třídu konečných jazyků platí následující:

$$\mathscr{L}(FIN) \subset \mathscr{L}(REG) \tag{4.11}$$

Důkaz: Tento důkaz je jednoduchý – platnost vztahu $\mathscr{L}(FIN) \subseteq \mathscr{L}(REG)$ plyne z toho, že pro každý konečný jazyk dokážeme sestrojit regulární gramatiku (postup viz výše).

Vlastní inkluzi dokážeme tak, že určíme alespoň jeden jazyk patřící do $\mathscr{L}(REG) - \mathscr{L}(FIN)$ (tj. regulární jazyk, který není konečný). Takovým jazykem je například $L = \{a^n\ ;\ n \geq 1\}$. Regulární gramatika generující tento jazyk je $G_{REG} = (\{S\},\ \{a\},\ P,\ S)$ s pravidly $S \to aS \mid a$. Zde stačilo zvolit jakýkoliv nekonečný regulární jazyk. □

Poznámka:

Z výše uvedeného vyplývá, že vztahy v Chomského hierarchii můžeme doplnit následovně:

$$\mathscr{L}(FIN) \subset \mathscr{L}(REG) \subset \mathscr{L}(LIN) \subset \mathscr{L}(CF) \subset \mathscr{L}(1) \subset \mathscr{L}(RE) \tag{4.12}$$

4.3 Další možnosti generování výstupu

Gramatiky Chomského hierarchie mají tyto společné vlastnosti:

- odvození začíná větnou formou obsahující jeden symbol (startovací symbol),
- neterminální a terminální abeceda, které jsou navzájem disjunktní,
- sekvenční způsob práce – v každém kroku je zpracován jeden symbol jedním pravidlem,
- všechna pravidla jsou „rovnoprávná“, nejsou rozdělena do žádných skupin, žádné z nich nemá vyšší prioritu než ostatní.

Existují jiné typy gramatik a gramatických systémů (gramatický systém je systém spolupracujících gramatik), které minimálně jednu z těchto vlastností nemají. Pro představu si ukážeme jeden takový systém, který „porušuje“ první tři pravidla.

0L systém $G = (\Sigma, P, \omega_0)$ je druh L systému (viz první kapitolu, str. 5), ve kterém začíná odvození nikoliv symbolem, ale obecně axiomem – startovacím řetězcem. Odvození probíhá paralelně, v každém kroku jsou přepsány vždy všechny symboly slova. Máme jedinou abecedu Σ a každá větná forma v odvození je větou (řadí se do jazyka generovaného systémem):
$L(G) = \{w \in \Sigma^* \ ; \ \omega_0 \Rightarrow^* w\}$

Příklad 4.5

Vezměme 0L systém $G_1 = (\Sigma, P, \omega_0)$, kde je abeceda $\Sigma = \{a\}$, množina pravidel $P = \{a \to aa\}$, axiom (startovací řetězec) má pouze délku 1, je $\omega_0 = a$.

Odvození v tomto systému je plně paralelní – v každém kroku se přepisuje každý symbol (každé a v řetězci se přepíše na aa, takže se v každém kroku zdvojnásobí počet symbolů a):
$a \Rightarrow a^2 \Rightarrow a^4 \Rightarrow a^8 \Rightarrow a^{16} \Rightarrow a^{32} \Rightarrow \ldots$

Jazyk tohoto 0L systému obsahuje všechna slova, která lze odvodit z axiomu v některém počtu kroků, také axiom samotný do tohoto jazyka patří. V každém kroku se počet symbolů a ve slově zdvojuje, délka slova tedy roste exponenciálně:
$L(G_1) = \{a^{2^n} \ ; \ n \geq 0\}$

Jazyk 0L systému z předchozího příkladu řadíme v Chomského hierarchii k jazykům typu 1 (nelze jej generovat bezkontextovou gramatikou). Jak vidíme, i velmi jednoduchý 0L systém generuje jazyk, pro který bychom v Chomského hierarchii museli vytvořit poměrně složitou gramatiku.

Příklad 4.6

Ukažme si jiný 0L systém $G_2 = (\Sigma, P, \omega_0)$, kde je abeceda $\Sigma = \{a, b\}$, množina pravidel je dvouprvková $P = \{a \to bb, \ b \to aa\}$, axiom je $\omega_0 = ab$.

Odvození v tomto systému vypadá takto:
$ab \Rightarrow b^2a^2 \Rightarrow a^4b^4 \Rightarrow b^8a^8 \Rightarrow a^{16}b^{16} \Rightarrow b^{32}a^{32} \Rightarrow \ldots$

Nezáleží na tom, jestli slovo vzniklo v lichém nebo sudém kroku výpočtu, všechna se řadí do jazyka – v lichém kroku jsou slova začínající symboly b, v sudém slova začínající symboly a.
$L(G_2) = \{a^ib^i \ ; \ i = 2^{2\cdot n}, \ n \geq 0\} \cup \{b^ia^i \ ; \ i = 2^{2\cdot n+1}, \ n \geq 0\}$

Opět se jedná o jazyk typu 1, způsob generování je podobný jako v předchozím případě. 0L systémy mají naopak problémy s některými konečnými jazyky, některé z nich nedokážou generovat.

Příklad 4.7

Další 0L systém $G_3 = (\Sigma, P, \omega_0)$, kde je abeceda $\Sigma = \{a, b\}$, množina pravidel je taktéž dvouprvková $P = \{a \to ab, \ b \to a\}$, axiom je $\omega_0 = b$. Ukázka odvození:
$b \Rightarrow a \Rightarrow ab \Rightarrow aba \Rightarrow abaab \Rightarrow abaababa \Rightarrow abaababaabaab \Rightarrow \ldots$

V tomto případě nás nebude ani tak zajímat, které symboly jsou ve slově a na kterých místech, budou nás zajímat délky generovaných slov. Axiom má délku 1, druhé slovo také, třetí slovo má délku 2, čtvrté délku 3, následují délky 5, 8, 13, atd. Dostáváme řadu
$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$
Při bližším pohledu zjistíme, že každé číslo posloupnosti (kromě prvních dvou) je součtem předchozích dvou čísel – je to Fibbonacciho posloupnost.

Maticové gramatiky také nejsou přímo součástí Chomského hierarchie gramatik. Je to jeden z mechanismů řízeného odvozování. Maticová gramatika pracuje podobně jako běžná (máme neterminální a terminální abecedu a jeden startovací symbol, odvozujeme víceméně sekvenčně), ale pravidla jsou rozdělena do skupin, kterým říkáme matice, a navíc je v těchto maticích dáno pořadí těchto pravidel (tj. porušujeme poslední odrážku v textu na začátku této sekce).

Příklad 4.8

Představíme maticovou gramatiku $G_M = (N, T, \mathcal{M}, S)$, kde $N = \{S, A, B, C\}$, $T = \{a, b, c\}$, množina matic $\mathcal{M} = \{m_1, m_2, m_3\}$, jednotlivé matice jsou
$m_1 = \{S \to ABC\}$
$m_2 = \{A \to aA, \ B \to bB, \ C \to cC\}$
$m_3 = \{A \to a, \ B \to b, \ C \to c\}$

Krok odvození probíhá takto:

- vybereme tu matici, jejíž všechna pravidla jsou aplikovatelná (přepisované symboly jsou ve zpracovávaném slově, případně symbol přepisovaný určitým pravidlem matice může být vytvořen některým z předchozích pravidel matice),
- pravidla v matici provedeme sekvenčně jedno po druhém (všechna, postupně zleva doprava).

V prvním kroku je aplikovatelná pouze první matice, protože ve zpracovávaném slově máme pouze symbol S. V dalších krocích používáme opakovaně matici m_2, matice m_3 bude použita až v posledním kroku (protože všechny neterminály přepíše na terminály). Pro přehlednost si u každého kroku poznačíme použitou matici (jinak toto značení nepoužíváme).
$S \overset{m_1}{\Rightarrow} ABC \overset{m_2}{\Rightarrow} aAbBcC \overset{m_2}{\Rightarrow} aaAbbBccC \overset{m_2}{\Rightarrow} aaaAbbbBcccC \overset{m_3}{\Rightarrow} aaaabbbbcccc$

Tato maticová gramatika generuje jazyk
$L(G_M) = \{a^n b^n c^n \ ; \ n \geq 1\}$

Maticové gramatiky opět nelze přesně napasovat do Chomského hierarchie, nicméně (jak vidíme na předchozím příkladu) jejich generativní síla je vysoká – zde jsme vcelku jednoduchou maticovou gramatikou vygenerovali jazyk patřící do $\mathscr{L}(1)$.

Kapitola 5

Regulární gramatiky

V této kapitole se budeme zabývat gramatikami typu 3 v Chomského hierarchii, tedy regulárními gramatikami, a jejich vztahem ke konečným automatům.

5.1 Definice regulární gramatiky

Definici regulární gramatiky jsme si uvedli v předchozí kapitole, zde si ji zopakujeme:

Definice 5.1 (Regulární gramatika)

Regulární gramatika gramatika typu 3 je gramatika, jejíž všechna pravidla jsou v jednom z následujících tvarů:

$$\begin{aligned} A &\to aB, \qquad A, B \in N,\ a \in T \\ A &\to a, \qquad A \in N,\ a \in T \end{aligned} \tag{5.1}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Regulární typ pravidel je v principu nezkracující (tj. na pravé straně pravidla máme řetězec stejně dlouhý nebo delší než na levé straně pravidla).

Poznámka:

To je důvodem, proč potřebujeme „výjimku“ – pravidlo $S \to \varepsilon$ pro startovací symbol, pokud potřebujeme do jazyka generovaného gramatikou zařadit i prázdné slovo.

Podmínka „S se pak nesmí nacházet na pravé straně žádného pravidla“ je také důležitá – kdyby v definici nebyla, pak by vlastnost „nezkracující“ přestala fungovat, protože bychom mohli v průběhu výpočtu zpracovávané slovo krátit (některý symbol přepsat na ε).

Jak bylo dříve uvedeno, regulární gramatiky fungují podobně jako konečné automaty (ne zcela – gramatika je generativní mechanismus, kdežto automat je rozpoznávací mechanismus). Stejně jako konečný automat postupuje striktně zleva doprava, v každém kroku vygeneruje (kde automat zpracuje) právě jeden terminální symbol (tj. prvek abecedy).

Příklad 5.1

Sestrojíme regulární gramatiku pro jazyk zadaný regulárním výrazem $R = caa^* + d(ab)^* + \varepsilon$.

Postupujeme následovně:

- Pro sekvenci aa^* potřebujeme pravidla
  $A \to aA \mid a$
- Pro sekvenci $(ab)^*$ potřebujeme pravidla
  $B \to aD$
  $D \to bB \mid b$
  (a zároveň je třeba zajistit, aby dvojic ab mohlo být i nula).
- Ze startovacího symbolu S této gramatiky povedou tři možné „cesty“ – možnosti pokračování výpočtu – odpovídající třem různým tvarům slov jazyka (v regulárním výrazu spojeným operátorem $+$.
  $S \to cA$ pro první tvar slov, napojíme pravidla generující symboly a
  $S \to dB \mid d$ pro druhý tvar slov, musíme zajistit i počet $(ab)^0$
  $S \to \varepsilon$ třetí tvar reprezentuje prázdné slovo

Celá gramatika $G = (N, T, P, S)$ je určena takto: $N = \{S, A, B, D\}$, $T = \{a, b, c, d\}$, množina pravidel P obsahuje tato pravidla:

$S \to cA \mid dB \mid d \mid \varepsilon$
$A \to aA \mid a$
$B \to aD$
$D \to bB \mid b$

Některé možné derivace v této gramatice jsou:

$S \Rightarrow cA \Rightarrow caA \Rightarrow caaA \Rightarrow caaa$
$S \Rightarrow d$
$S \Rightarrow dB \Rightarrow daD \Rightarrow dabB \Rightarrow dabaD \Rightarrow dababB \Rightarrow dababaD \Rightarrow dababab$
$S \Rightarrow \varepsilon$

Poznámka:

Není jednoduché vytvořit regulární gramatiku přímo podle regulárního výrazu nebo předpisu jazyka. Často bývá jednodušší vytvořit nejdřív konečný automat (postupem, který jsme si ukazovali v sekci o uzávěrových vlastnostech (kapitola 3.5, strana 52) a následně podle konečného automatu vytvořit ekvivalentní regulární gramatiku (postup si ukážeme na následujících stránkách).

5.2 Vytvoření konečného automatu podle regulární gramatiky

V předchozí kapitole je uvedeno, že regulárním gramatikám (coby generativnímu mechanismu) odpovídají konečné automaty (coby rozpoznávací mechanismus). V této sekci si ukážeme, jak podle regulární gramatiky sestrojit ekvivalentní konečný automat (tj. takový automat, který rozpoznává tentýž jazyk, který gramatika generuje).

Věta 5.1 (Regulární gramatika ⟶ konečný automat)

Pro třídy jazyků generovaných regulárními gramatikami a rozpoznávaných konečnými automaty platí vztah

$$\mathscr{L}(REG) \subseteq \mathscr{L}(KA) \tag{5.2}$$

Postup konstrukce: Je dána regulární gramatika $G = (N, T, P, S)$ a naším úkolem je sestrojit konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ takový, že $L(G) = L(\mathcal{A})$. Postupujeme takto:

- abecedu přejmeme, použijeme terminální abecedu gramatiky: $\Sigma = T$,
- neterminály použijeme jako stavy, a protože musíme v automatu začínat na stejném místě jako v gramatice, jako počáteční stav použijeme startovací symbol gramatiky,
- obdobu konečných stavů u neterminálů nenajdeme (v odvozování v gramatice všechny neterminály „zmizí“ před koncem derivace, ve finální větě už nejsou), proto vytvoříme nový stav, jeho označení zvolíme tak, aby nekolidovalo s označením neterminálů gramatiky, často se používá X,
- pokud je v jazyce gramatiky prázdné slovo ε, pak bude do množiny koncových stavů patřit i počáteční stav S, jinak to bude pouze stav X,
- přechodovou funkci δ zkonstruujeme podle množiny pravidel P:
  pro $(A \to aB) \in P \quad \delta(A, a) \ni B$
  pro $(A \to a) \in P \quad \delta(A, a) \ni X$

Množina stavů Q tedy bude obsahovat tyto prvky: $Q = N \cup \{X\}$, kde $X \notin N$ (tj. nově přidaný). Množina koncových stavů je určena takto:

$$F = \begin{cases} \{X\} \ ; \ \varepsilon \notin L(G) \\ \{X, S\} \ ; \ \varepsilon \in L(G) \end{cases}$$

□

Příklad 5.2

Je dána gramatika $G = (\{S, A\},\ \{a, b, c, d\},\ P,\ S)$, kde v P jsou pravidla
$S \to aS \mid aA \mid c$
$A \to bA \mid cS \mid d$

Vytvoříme konečný automat $\mathcal{A} = (Q,\ \{a, b, c, d\},\ \delta,\ S,\ F)$:

- jako počáteční stav použijeme startovací symbol gramatiky S,
- $Q = \{S, A, X\}$ (použili jsme množinu neterminálů, přidali jsme stav pro ukončení výpočtu),
- $F = \{X\}$ (v jazyce gramatiky zjevně není prázdné slovo, protože tam nemáme epsilonové pravidlo).

Přechodovou funkci sestrojíme podle množiny pravidel P, budeme postupovat po jednotlivých pravidlech takto:

- podle pravidel $S \to aS \mid aA$ vytvoříme předpis $\delta(S, a) = \{S, A\}$,
- podle pravidla $S \to c$ vytvoříme předpis $\delta(S, c) = \{X\}$,
- atd. podle dalších pravidel.

Mnemotechnická pomůcka: v přechodové funkci budou jednotlivé prvky ve stejném pořadí jako v pravidle, podle kterého předpis tvoříme:

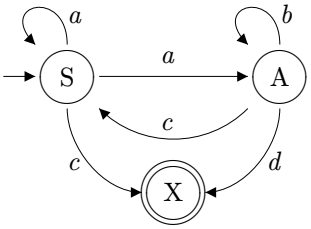
pravidlo:		S	$\to$	a		A
přechod:	$\delta($	S	,	a	$) =$	$\{\ldots, A\}$

Tabulka 5.1: Vytvoření předpisu přechodové funkce podle pravidla gramatiky

Ukážeme si všechny tři reprezentace výsledného konečného automatu:

$\delta(S, a) = \{S, A\}$
$\delta(S, c) = \{X\}$
$\delta(A, b) = \{A\}$
$\delta(A, c) = \{S\}$
$\delta(A, d) = \{X\}$

$\mathcal{A}$	a	b	c	d
$\to S$	S, A		X	
A		A	S	X
$\leftarrow X$				



Jazyk gramatiky i automatu je
$L(G) = L(\mathcal{A}) = a^*(ab^*ca^*)^*c + a^*ab^*(ca^*ab^*)^*d$

Příklad 5.3

V předchozím příkladu byla gramatika, v jejímž jazyce není prázdné slovo. V tomto příkladu si ukážeme postup pro gramatiku, v jejímž jazyce prázdné slovo je.

Je dána gramatika $G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$, kde v P jsou pravidla
$S \to aA \mid b \mid \varepsilon$
$A \to bA \mid bB$
$B \to aA \mid b$

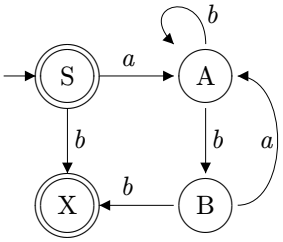
Vytvoříme konečný automat $\mathcal{A} = (Q,\ \{a, b\},\ \delta,\ S,\ F)$. Pravidlo gramatiky $S \to \varepsilon$ se při převodu na konečný automat projeví pouze tím, že stav S zařadíme do množiny koncových stavů. Všechna ostatní pravidla využijeme při konstrukci přechodové funkce, a to stejným způsobem jako v předchozím příkladu:

- $Q = \{S, A, B, X\}$,
- $F = \{X, S\}$,

přechodová funkce bude následující (opět uvedeme všechny tři reprezentace):

$\delta(S, a) = \{A\}$
$\delta(S, b) = \{X\}$
$\delta(A, b) = \{A, B\}$
$\delta(B, a) = \{A\}$
$\delta(B, b) = \{X\}$

$\mathcal{A}$	a	b
$\leftrightarrow S$	A	X
A		A, B
B	A	X
$\leftarrow X$		



Jazyk gramatiky i automatu je
$L(G) = L(\mathcal{A}) = \varepsilon + b + ab^*b(ab^*b)^*b$

Pozor, pravidlo $S \to \varepsilon$ se u automatu v přechodové funkci nijak neprojevuje – projeví se pouze a jenom tím, že množina koncových stavů bude dvouprvková: $F = \{X, S\}$.

Důkaz (Věta 5.1): Dokážeme, že výše popsaný algoritmus vytvoří automat rozpoznávající jazyk, který je generován danou gramatikou, tedy že $L(\mathcal{A}) = L(G)$.
Dokazujeme $L(G) \subseteq L(\mathcal{A})$:
Vezměme jakékoliv slovo $w \in L(G),\ |w| \geq 1,$ označme $w = a_1 a_2 \dots a_n,$ kde $a_i \in T,\ 1 \leq i \leq n$ (tj. rozložíme slovo na jednotlivé symboly).

$\Rightarrow$ $\exists$ derivace slova w v gramatice G
$S = A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \dots \Rightarrow a_1 a_2 \dots a_n = w$ (o délce $n,\ n \geq 1$)
(slovo generujeme po jednotlivých symbolech, v každém kroku jeden)

$\Rightarrow$ tomu odpovídá posloupnost použitých pravidel:
pro každý krok $i,\ 1 \leq i \leq (n-1)$:
$a_1 \dots a_{i-1} A_{i-1} \Rightarrow a_1 \dots a_{i-1} a_i A_i$ pravidlo $A_{i-1} \to a_i A_i,$ kde $A_{i-1}, A_i \in N,\ a_1, \dots, a_i \in T$
(neznamená to, že v každém kroku použijeme jiný neterminál, tolik jich nemáme; jen jsou takto označeny)
pro poslední krok (n) :
$a_1 \dots a_{n-1} A_{n-1} \Rightarrow a_1 \dots a_{n-1} a_n$ pravidlo $A_{n-1} \to a_n,$ kde $A_{n-1} \in N,\ a_n \in T$

$\Rightarrow$ v přechodové funkci automatu máme podle těchto pravidel vytvořeny předpisy:
pro každý krok $i,\ 1 \leq i \leq (n-1)$:
předpisy $\delta(A_{i-1}, a_i) \ni A_i,\ A_{i-1}, A_i \in Q,\ a_i \in \Sigma$
pro poslední krok (n) :
předpis $\delta(A_{n-1}, a_n) \ni X,\ A_{n-1} \in Q,\ a_n \in \Sigma$

$\Rightarrow$ v $\mathcal{A}$ lze sestrojit posloupnost konfigurací
$(S, a_1 a_2 \dots a_n) \vdash (A_1, a_2 \dots a_n) \vdash \dots \vdash (A_{n-1}, a_n) \vdash (X, \varepsilon)$
(na konci výpočtu je koncová konfigurace)

$\Rightarrow$ $w \in L(\mathcal{A})$

V případě, že $\varepsilon \in L(G)$ a $w = \varepsilon$ (tj. $|w| = 0$):

$\Rightarrow$ v gramatice G máme odvození $S \Rightarrow \varepsilon$

$\Rightarrow$ v automatu $\mathcal{A}$ platí $S \in F$

$\Rightarrow$ automat $\mathcal{A}$ přijímá prázdné slovo

$\Rightarrow$ $\varepsilon \in L(\mathcal{A})$

Dokazujeme $L(G) \supseteq L(\mathcal{A})$:
Vezměme jakékoliv slovo $w \in L(\mathcal{A}),\ |w| \geq 1,$ označme $w = a_1 a_2 \dots a_n,$ kde $a_i \in \Sigma,\ 1 \leq i \leq n.$

$\Rightarrow$ $\exists$ výpočet slova w v automatu $\mathcal{A}$
$(S, a_1 a_2 \dots a_n) \vdash (A_1, a_2 \dots a_n) \vdash \dots \vdash (A_{n-1}, a_n) \vdash (X, \varepsilon)$

$\Rightarrow$ tomu odpovídají předpisy v přechodové funkci:
pro každý krok $i,\ 1 \leq i \leq (n-1)$:
předpisy $\delta(A_{i-1}, a_i) \ni A_i,\ A_{i-1}, A_i \in Q,\ a_i \in \Sigma$
pro poslední krok (n) :
předpis $\delta(A_{n-1}, a_n) \ni X,\ A_{n-1} \in Q,\ a_n \in \Sigma$

$\Rightarrow$ tyto předpisy byly vytvořeny podle těchto pravidel gramatiky:
pro každý krok $i,\ 1 \leq i \leq (n-1)$:
$a_1 \ldots a_{i-1}A_{i-1} \Rightarrow a_1 \ldots a_{i-1}a_iA_i$ pravidlo $A_{i-1} \to a_iA_i,$ kde $A_{i-1}, A_i \in N,\ a_1, \ldots, a_i \in T$
pro poslední krok (n) :
$a_1 \ldots a_{n-1}A_{n-1} \Rightarrow a_1 \ldots a_{n-1}a_n$ pravidlo $A_{n-1} \to a_n,$ kde $A_{n-1} \in N,\ a_n \in T$
$\Rightarrow$ v G existuje odvození
$S = A_0 \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow \ldots \Rightarrow a_1a_2 \ldots a_n = w$
$\Rightarrow$ $w \in L(G)$
V případě, že $\varepsilon \in L(\mathcal{A})$ a $w = \varepsilon$ (tj. $|w| = 0$):
$\Rightarrow$ v automatu $\mathcal{A}$ platí $S \in F$
$\Rightarrow$ v gramatice G máme odvození $S \Rightarrow \varepsilon$
$\Rightarrow$ $\varepsilon \in L(G)$

Z toho vyplývá, že jazyk gramatiky G a automatu $\mathcal{A}$ sestrojeného dle výše uvedeného postupu jsou ekvivalentní. □

5.3 Vytvoření regulární gramatiky podle konečného automatu

Podíváme se na opačný směr – je dán konečný automat a máme sestrojit ekvivalentní regulární gramatiku.

Věta 5.2 (Konečný automat $\longrightarrow$ regulární gramatika)

Pro třídy jazyků generovaných regulárními gramatikami a rozpoznávaných konečnými automaty platí vztah

$$\mathscr{L}(REG) \cong \mathscr{L}(KA) \tag{5.3}$$

V předchozí sekci jsme ukázali na příkladu a následně dokázali, že $\mathscr{L}(REG) \subseteq \mathscr{L}(KA)$, zbývá dokázat opačný směr: $\mathscr{L}(REG) \supseteq \mathscr{L}(KA)$. Tím dokážeme ekvivalenci obou tříd jazyků.

Postup konstrukce: Je dán konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, naším úkolem je sestrojit regulární gramatiku $G = (N, T, P, S)$ takovou, že $L(G) = L(\mathcal{A})$.

Nejdřív si musíme uvědomit, že na regulární gramatiku jsou kladeny poněkud jiné požadavky než na konečný automat:

- Pokud do jazyka patří slovo ε, v gramatice potřebujeme pravidlo $S \to \varepsilon$, ale potom se S nesmí nacházet na pravé straně žádného pravidla; jenže v automatu je to „volnější“ – v tomto případě bude počáteční stav patřit do množiny koncových stavů, ale není vyžadováno, aby pak do něj nevedly žádné přechody (tj. může se vyskytovat „na pravé straně“ předpisu přechodové funkce).

$\Rightarrow$ Budeme požadovat, aby v případě, že $\varepsilon \in L(\mathcal{A})$, do počátečního stavu nevedly žádné přechody.

- Zatímco u gramatiky končíme defacto jednotně (ze zpracovávaného slova „zmizí“ neterminál), u konečného automatu může dojít k situaci, že jsme sice v koncovém stavu, ale vstup ještě

není zpracovaný – pak prostě pokračujeme dál (jinými slovy – i z koncového stavu mohou vést přechody).

⇒ Pro zjednodušení můžeme požadovat, aby po úpravě měl automat pouze jeden koncový stav (s případnou výjimkou počátečního stavu, jestliže do jazyka patří prázdné slovo) a z něj nesmí vést žádné přechody.

Můžeme sice prostě použít přesně opačný postup k tomu, který jsme si ukázali na předchozích stránkách, ale nejdřív musíme vyřešit výše jmenované odlišnosti a automat upravit takto:

1. Pokud $\varepsilon \in L(\mathcal{A})$ a zároveň do počátečního stavu vede nejméně jeden přechod, vytvoříme nový počáteční stav, který nahradí ten původní *na začátku výpočtu* (původní necháme, jen už nebude počáteční). Z nového počátečního stavu povedou stejné přechody jako z původního.
2. Pokud existuje víc koncových stavů (nepočítaje v to počáteční stav) nebo sice jeden, ale vedou do něj přechody, vytvoříme nový stav, který přejme funkci původních koncových stavů *na konci výpočtu* (původní necháme, jen už nebudou koncovými stavy). Do nového koncového stavu povedou stejné přechody jako do původních.

Pořadí kroků je důležité, protože po provedení prvního kroku může vzniknout další koncový stav, který není počáteční a přitom do něj vedou přechody. Všechny tyto úpravy jsou *ekvivalentní*, tj. po jejich provedení se nijak nezmění rozpoznávaný jazyk (jen pozměníme cesty v grafu automatu, jejichž procházením jsou slova rozpoznávána).

Předpokládejme, že předběžná úprava je již provedena. Dále postupujeme takto:

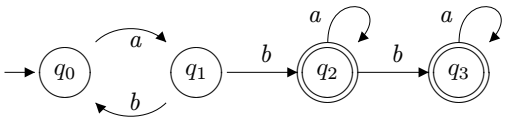
- jako terminální abecedu použijeme abecedu automatu: $T = \Sigma$,
- stavy použijeme jako neterminály, kromě koncového stavu (označme jej například X), jako startovací symbol použijeme počáteční stav automatu,
- množinu pravidel P zkonstruujeme podle přechodové funkce automatu:
  pro $\delta(A, a) \ni B \quad (A \to aB) \in P$
  pro $\delta(A, a) \ni X \quad (A \to a) \in P$
- pokud je v jazyce automatu $\mathcal{A}$ prázdné slovo ε, pak v množině pravidel gramatiky G bude pravidlo $S \to \varepsilon$,
- označme koncový stav automatu X; množina neterminálů gramatiky je $N = Q - \{X\}$. □

Příklad 5.4

Je dán konečný automat $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_2, q_3\})$

$\delta(q_0, a) = \{q_1\}$
$\delta(q_1, b) = \{q_0, q_2\}$
$\delta(q_2, a) = \{q_2\}$
$\delta(q_2, b) = \{q_3\}$
$\delta(q_3, a) = \{q_3\}$

$\mathcal{A}$	a	b
$\to q_0$	q_1	
q_1		q_0, q_2
$\leftarrow q_2$	q_2	q_3
$\leftarrow q_3$	q_3	

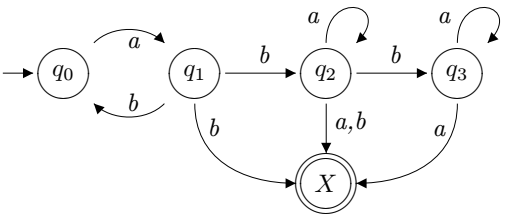


Počáteční stav není v množině koncových stavů, tedy první bod předběžné úpravy nemusíme provádět, ale druhý bod ano – máme víc než jeden koncový stav a navíc do obou těchto stavů vedou přechody.

Nejdřív automat upravíme, abychom mohli použít postup přesně opačný postupu převodu gramatiky na automat. Vytvoříme nový stav X, který bude plnit roli původních koncových stavů q_2 a q_3, a to pouze na konci zpracování slov (objeví se v poslední konfiguraci v posloupnosti výpočtu).

$\delta(q_0, a) = \{q_1\}$
$\delta(q_1, b) = \{q_0, q_2, X\}$
$\delta(q_2, a) = \{q_2, X\}$
$\delta(q_2, b) = \{q_3, X\}$
$\delta(q_3, a) = \{q_3, X\}$

$\mathcal{A}'$	a	b
$\rightarrow q_0$	q_1	
q_1		q_0, q_2, X
q_2	q_2, X	q_3, X
q_3	q_3, X	
$\leftarrow X$		



Výsledná gramatika a její úprava (jen nahradíme neterminály velkými písmeny, aby byla množina pravidel přehlednější):

$G = (\{q_0, q_1, q_2, q_3\},\ \{a, b\},\ P,\ q_0)$
$q_0 \rightarrow aq_1$
$q_1 \rightarrow bq_0 \mid bq_2 \mid b$
$q_2 \rightarrow aq_2 \mid bq_3 \mid a \mid b$
$q_3 \rightarrow aq_3 \mid a$

$G' = (\{S, A, B, C\},\ \{a, b\},\ P,\ S)$
$S \rightarrow aA$
$A \rightarrow bS \mid bB \mid b$
$B \rightarrow aB \mid bC \mid a \mid b$
$C \rightarrow aC \mid a$

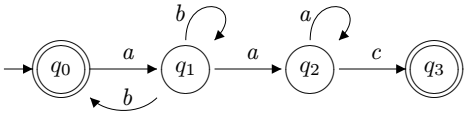
Jazyk gramatiky i automatu je
$L(G) = L(\mathcal{A}) = a(ba)^*b \ + \ a(ba)^*ba^*(a + b + ba^*a)$

Příklad 5.5

Je dán konečný automat $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta, q_0, \{q_0, q_3\})$

$\delta(q_0, a) = \{q_1\}$
$\delta(q_1, a) = \{q_2\}$
$\delta(q_1, b) = \{q_0, q_1\}$
$\delta(q_2, a) = \{q_2\}$
$\delta(q_2, c) = \{q_3\}$

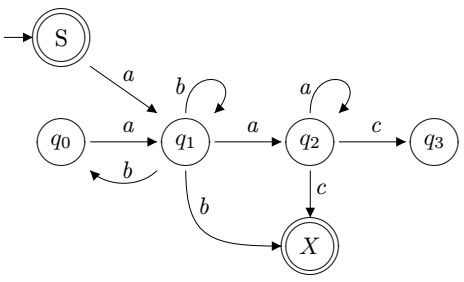
$\mathcal{A}$	a	b	c
$\leftrightarrow q_0$	q_1		
q_1	q_2	q_0, q_1	
q_2	q_2		q_3
$\leftarrow q_3$			



Upravíme automat, tentokrát musíme řešit i počáteční stav:

$\delta(S, a) = \{q_1\}$
$\delta(q_0, a) = \{q_1\}$
$\delta(q_1, a) = \{q_2\}$
$\delta(q_1, b) = \{q_0, q_1, X\}$
$\delta(q_2, a) = \{q_2\}$
$\delta(q_2, c) = \{q_3, X\}$

	a	b	c
$\leftrightarrow S$	q_1		
q_0	q_1		
q_1	q_2	q_0, q_1, X	
q_2	q_2		q_3, X
q_3			
$\leftarrow X$			



Stav q_3 se stává nadbytečným, neexistuje žádná cesta vedoucí od něho do koncového stavu, proto jej můžeme odstranit a ve výsledné gramatice se nijak neprojeví.

Výsledná gramatika a převedení neterminálů na velká písmena:

$G = (\{S, q_0, q_1, q_2\},\ \{a, b, c\},\ P,\ S)$

$S \to aq_1 \mid \varepsilon$

$q_0 \to aq_1$

$q_1 \to aq_2 \mid bq_0 \mid bq_1 \mid b$

$q_2 \to aq_2 \mid c$

$G' = (\{S, A, B, C\},\ \{a, b, c\},\ P,\ S)$

$S \to aB \mid \varepsilon$

$A \to aB$

$B \to aC \mid bA \mid bB \mid b$

$C \to aC \mid c$

Jazyk gramatiky i automatu je

$L(G) = L(\mathcal{A}) = (ab^*b)^* \cdot (\varepsilon + ab^*aa^*c)$

Důkaz (Věta 5.2): Navazujeme na předchozí důkaz – jeden směr ekvivalence máme dokázán, je třeba dokázat $\mathscr{L}(REG) \supseteq \mathscr{L}(KA)$.

Dokážeme, že výše popsaný algoritmus vytvoří gramatiku generující jazyk, který je rozpoznáván daným automatem, tedy že $L(G) = L(\mathcal{A})$. Předpokládejme, že automat je upraven, je v tom tvaru, jaký byl naznačen v postupu konstrukce.

Dokazujeme $L(\mathcal{A}) \subseteq L(G)$:

Vezměme jakékoliv slovo $w \in L(\mathcal{A}),\ |w| \geq 1,$ označme $w = a_1a_2 \ldots a_n,$ kde $a_i \in \Sigma,\ 1 \leq i \leq n.$

$\Rightarrow$ $\exists$ výpočet slova w v automatu $\mathcal{A}$

$(q_0, a_1a_2 \ldots a_n) \vdash (q_1, a_2 \ldots a_n) \vdash \ldots \vdash (q_{n-1}, a_n) \vdash (X, \varepsilon)$

$\Rightarrow$ tomu odpovídají předpisy v přechodové funkci:

pro každý krok $i,\ 1 \leq i \leq (n-1)$:

předpisy $\delta(q_{i-1}, a_i) \ni q_i,\ q_{i-1}, q_i \in Q,\ a_i \in \Sigma$

pro poslední krok (n) :

předpis $\delta(q_{n-1}, a_n) \ni X,\ q_{n-1} \in Q,\ a_n \in \Sigma$

$\Rightarrow$ pravidla gramatiky jsme tvořili podle přechodové funkce, tedy existují následující:

pro každý krok $i,\ 1 \leq i \leq (n-1)$:

$a_1 \ldots a_{i-1}q_{i-1} \Rightarrow a_1 \ldots a_{i-1}a_iq_i$ pravidlo $q_{i-1} \to a_iq_i,$ kde $q_{i-1}, q_i \in N,\ a_1, \ldots, a_i \in T$

pro poslední krok (n) :

$a_1 \ldots a_{n-1}q_{n-1} \Rightarrow a_1 \ldots a_{n-1}a_n$ pravidlo $q_{n-1} \to a_n,$ kde $q_{n-1} \in N,\ a_n \in T$

$\Rightarrow$ v G existuje odvození

$q_0 \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \ldots \Rightarrow a_1a_2 \ldots a_n = w$

$\Rightarrow$ $w \in L(G)$

V případě, že $\varepsilon \in L(\mathcal{A})$ a $w = \varepsilon$ (tj. $|w| = 0$):

$\Rightarrow$ v automatu $\mathcal{A}$ platí $q_0 \in F$

$\Rightarrow$ v gramatice G máme odvození $q_0 \Rightarrow \varepsilon$

$\Rightarrow$ $\varepsilon \in L(G)$

Dokazujeme $L(\mathcal{A}) \supseteq L(G)$:

Vezměme jakékoliv slovo $w \in L(G),\ |w| \geq 1,$ označme $w = a_1a_2 \ldots a_n,$ kde $a_i \in T,\ 1 \leq i \leq n.$

$\Rightarrow$ $\exists$ derivace slova w v gramatice G

$q_0 \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \ldots \Rightarrow a_1a_2 \ldots a_n = w$ (o délce $n,\ n \geq 1$)

$\Rightarrow$ tomu odpovídá posloupnost použitých pravidel:
pro každý krok $i,\ 1 \leq i \leq (n-1)$:
$a_1 \ldots a_{i-1}q_{i-1} \Rightarrow a_1 \ldots a_{i-1}a_iq_i$ pravidlo $q_{i-1} \to a_iq_i,$ kde $q_{i-1}, q_i \in N,\ a_1, \ldots, a_i \in T$
pro poslední krok (n) :
$a_1 \ldots a_{n-1}q_{n-1} \Rightarrow a_1 \ldots a_{n-1}a_n$ pravidlo $q_{n-1} \to a_n,$ kde $q_{n-1} \in N,\ a_n \in T$

$\Rightarrow$ tato pravidla jsme vytvořili podle těchto předpisů v přechodové funkci:
pro každý krok $i,\ 1 \leq i \leq (n-1)$:
předpisy $\delta(q_{i-1}, a_i) \ni q_i,\ q_{i-1}, q_i \in Q,\ a_i \in \Sigma$
pro poslední krok (n) :
předpis $\delta(q_{n-1}, a_n) \ni X,\ q_{n-1} \in Q,\ a_n \in \Sigma$

$\Rightarrow$ v $\mathcal{A}$ existuje posloupnost konfigurací
$(q_0, a_1a_2 \ldots a_n) \vdash (q_1, a_2 \ldots a_n) \vdash \ldots \vdash (q_{n-1}, a_n) \vdash (X, \varepsilon)$

$\Rightarrow$ $w \in L(\mathcal{A})$

V případě, že $\varepsilon \in L(G)$ a $w = \varepsilon$ (tj. $|w| = 0$):

$\Rightarrow$ v gramatice G musí existovat odvození $S \Rightarrow \varepsilon$ podle pravidla $S \to \varepsilon$

$\Rightarrow$ v automatu $\mathcal{A}$ platí $q_0 \in F$

$\Rightarrow$ automat $\mathcal{A}$ přijímá prázdné slovo

$\Rightarrow$ $\varepsilon \in L(\mathcal{A})$

Z toho vyplývá, že jazyk automatu $\mathcal{A}$ a gramatiky G sestrojené dle výše uvedeného postupu jsou ekvivalentní. □

Kapitola 6

Bezkontextové gramatiky a jazyky

V této kapitole se budeme zabývat jazyky patřícími do třídy jazyků $\mathcal{L}(2)$ (resp. CF) v Chomského hierarchii, tedy bezkontextovými jazyky, a k nim ekvivalentními bezkontextovými gramatikami.

6.1 Definice bezkontextové gramatiky

S bezkontextovými gramatikami jsme se již seznámili v kapitole o formálních gramatikách a Chomského hierarchii, nicméně si definici připomeneme:

Definice 6.1 (Bezkontextová gramatika)

Gramatika typu 2 (bezkontextová, context-free – CF) je gramatika, jejíž všechna pravidla jsou v tomto tvaru:

$$A \to \beta, \quad A \in N,\ \beta \in (N \cup T)^* \tag{6.1}$$

Poznámka:

V mnoha zdrojích se setkáme s mírně odlišnou definicí bezkontextových pravidel:

$$A \to \beta, \quad A \in N,\ \beta \in (N \cup T)^*,\ |\beta| \geq 1, \tag{6.2}$$

dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Tato definice striktně zachovává dědění vlastností v rámci Chomského hierarchie (všimněte si – je to defacto tatáž definice jako u kontextových gramatik, o kterých víme, že jsou ekvivalentní nezkracujícím gramatikám typu 1, až na to, že zde odstraňujeme kontext).

Později si ukážeme, že obě tyto definice jsou navzájem ekvivalentní. K jakékoliv bezkontextové gramatice lze sestrojit ekvivalentní nezkracující bezkontextovou gramatiku (tedy bez ε-pravidel kromě případného pro startovací symbol).

My se zde přidržíme volnější verze, ve které lze používat ε-pravidla pro kterýkoliv neterminální symbol, protože právě tento tvar je prakticky použitelnější (využívá se například při programování).

Poznámka:

Je třeba si uvědomit, že k definici bezkontextové gramatiky ve skutečnosti patří také související definice, které platí pro gramatiky obecně – relace kroku odvození, reflexivní a tranzitivní uzávěr této relace, jazyk generovaný gramatikou, větná forma, věta.

Tyto definice jsou uvedeny v kapitole o formálních gramatikách od strany 70 (stačí jen v probíraných pravidlech uvést na levé straně jediný neterminál místo řetězce).

Příklad 6.1

Ukážeme si gramatiku generující tento jazyk:
$L_1 = \left\{ ww^R \ ; \ w \in \{a,b\}^* \right\}$

Jak vidíme, slova tohoto jazyka jsou všechny možné palindromy nad abecedou $\{a,b\}$ (palindrom je řetězec, který se čte zleva doprava stejně jako zprava doleva). Gramatika generující tento jazyk je jednoduchá, vystačíme si s jediným neterminálem:
$G_1 = (\{S\}, \{a,b\}, P, S)$, kde množina P obsahuje pravidla $S \to aSa \mid bSb \mid \varepsilon$

Ukázka derivace:
$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbbba$

Tato gramatika má dokonce pouze lineární tvar pravidel (to však nevadí, víme, že každá lineární gramatika je zároveň gramatikou bezkontextovou).

Příklad 6.2

Vytvoříme bezkontextovou gramatiku generující tento jazyk:
$L_2 = \{a^n b^m c^n \ ; \ m,n > 0\}$

Tento jazyk má s předchozím jedno společné – jeho slova sice nejsou palindromy, ale první a poslední část slova mají shodný počet prvků (i když odlišných). Také se bude jednat o gramatiku s lineárními pravidly.

Při návrhu gramatiky se vždy zamýšlíme nad tím, co má být „synchronizováno“ – zde je třeba zajistit, abychom měli stejný počet symbolů a na začátku jako symbolů c na konci. Synchronizaci zajistíme tak, že v rámci jediného pravidla generujeme jeden symbol a a zároveň jeden symbol c.
$G_2 = (\{S,A\}, \{a,b,c\}, P, S)$, kde množina P obsahuje pravidla
$S \to aSc \mid aAc$
$A \to bA \mid b$

Ukázka derivace:
$S \Rightarrow aSc \Rightarrow aaAcc \Rightarrow aabAcc \Rightarrow aabbAcc \Rightarrow aabbbcc$

Příklad 6.3

Třetí jazyk je následující:
$L_3 = \{0^n 1^m \ ; \ 1 \leq n \leq m\}$

Slova jazyka vypadají zdánlivě jednoduše – nejdřív symboly 0 a pak symboly 1. Důležitá je však podmínka, která říká, že počet nul má být menší nebo roven počtu jedniček. Splnění podmínky zajistíme tak, že nejdřív v rekurzivním pravidle generujeme nuly a jedničky tak, že jich je stejný počet, a pak v prostřední části můžeme přidat další jedničky.
$G_3 = (\{A\}, \{0, 1\}, P, A)$, kde množina P obsahuje pravidla
$A \to 0A1 \mid A1 \mid 01$

Ukázka derivace:
$A \Rightarrow 0A1 \Rightarrow 0A11 \Rightarrow 00111$

Příklad 6.4

L_4 je jazyk matematických výrazů obsahujících

- celá čísla,
- operátory $+$, $*$ (bez ohledu na prioritu),
- závorky.

Gramatika generující tento jazyk je
$G_4 = (\{E\}, \{n, +, *, (,)\}, P, E)$, kde množina P obsahuje pravidla
$E \to E + E \mid E * E \mid (E) \mid n$

Ukázka derivace:
$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow n + E * E \Rightarrow n + (E) * E \Rightarrow n + (E) * n \Rightarrow n + (n) * n$

Příklad 6.5

L_5 je jazyk matematických výrazů obsahujících

- celá čísla,
- operátory $+$, $-$, $*$, $/$, tentokrát s ohledem na priority operátorů,
- závorky.

Gramatika generující tento jazyk je
$G_5 = (\{E, F, G\}, \{n, +, -, *, /, (,)\}, P, E)$, kde množina P obsahuje pravidla
$E \to E + F \mid E - F \mid F$
$F \to F * G \mid F/G \mid G$
$G \to (E) \mid n$

Ukázka derivace:
$E \Rightarrow F \Rightarrow F * G \Rightarrow G * G \Rightarrow (E) * G \Rightarrow (E + F) * G \Rightarrow (F + F) * G \Rightarrow (G + F) * G \Rightarrow$
$\Rightarrow (n + F) * G \Rightarrow (n + G) * G \Rightarrow (n + n) * G \Rightarrow (n + n) * n$

Tento jazyk se po určité úpravě používá jako základ pro syntaktickou analýzu běžných programovacích jazyků.

6.2 Derivační strom

Derivační strom slova generovaného v dané gramatice je grafickým znázorněním derivace tohoto slova.

Definice 6.2 (Derivační strom)

Derivační strom dané derivace slova v gramatice $G = (N, T, P, S)$ je uspořádaný kořenový strom (tj. souvislý acyklický graf), jehož uzly jsou ohodnoceny symboly z množiny $N \cup T \cup \{\varepsilon\}$ a platí:

- *všechny uzly kromě kořene mají jednoho předchůdce, kořen nemá žádného,*
- *pokud je některý uzel ohodnocen symbolem A, jeho následníci po řadě symboly a_1, a_2, ... a_n, pak v množině pravidel P gramatiky existuje pravidlo $A \to a_1 a_2 \dots a_n$,*
- *jestliže uzel ohodnocený symbolem A má jediného následníka ohodnoceného symbolem ε, pak v P existuje pravidlo $A \to \varepsilon$,*
- *kořen stromu je ohodnocen S, listy jsou ohodnoceny symboly z množiny $T \cup \{\varepsilon\}$, ostatní uzly jsou ohodnoceny symboly z množiny N.*

V každém kroku vytváření derivačního stromu tvoří listy větnou formu v gramatice.

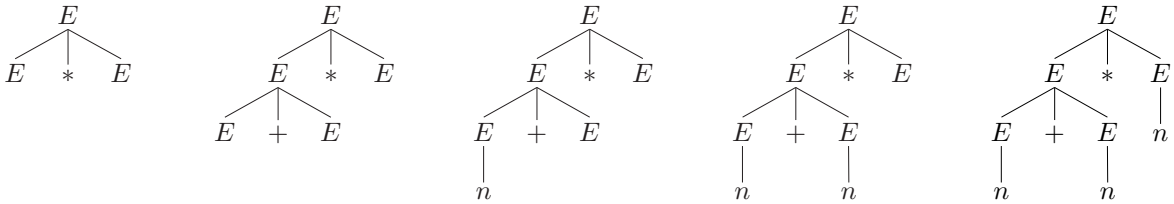
Příklad 6.6

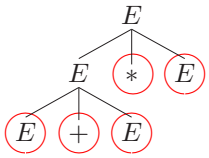
Vytvoříme derivační strom k jedné z derivací slova $n + n * n$ v gramatice, se kterou jsme se setkali v jednom z předchozích příkladů:
$G_4 = (\{E\}, \{n, +, *, (,)\}, P, E)$, kde množina P obsahuje pravidla
$E \to E + E \mid E * E \mid (E) \mid n$

Jedna z možných derivací:
$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow n + E * E \Rightarrow n + n * E \Rightarrow n + n * n$

Budeme postupně vytvářet derivační strom této derivace:

po 1. kroku: po 2. kroku: po 3. kroku: po 4. kroku: po 5. kroku:





Pokud v jakékoliv fázi konstrukce derivačního stromu přečteme obsah (momentálních) listů stromu zleva doprava, získáme větnou formu, která je v této fázi konstrukce v derivaci. Například po druhém kroku (díváme se na druhý obrázek výše, případně vlevo) vyčteme z listů tohoto „ministromu“ řetězec $E + E * E$, což je také větná forma vytvořená po druhém kroku derivace.

6.3 Nezkracující bezkontextová gramatika

Nezkracující bezkontextová gramatika je taková bezkontextová gramatika $G = (N, T, P, S)$, kde množina pravidel P buď neobsahuje žádné ε-pravidlo, nebo existuje jediné ε-pravidlo $S \to \varepsilon$ a zároveň S není na pravé straně žádného pravidla.

Definice 6.3 (Nezkracující bezkontextová gramatika)

Nezkracující bezkontextovou gramatikou je taková gramatika $G = (N, T, P, S)$, jejíž všechna pravidla jsou v tomto tvaru:

$$A \to \alpha, \quad |\alpha| \geq 1, \quad A \in N,\ \alpha \in (N \cup T)^* \tag{6.3}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Jak vidíme, tato definice je velmi podobná definici nezkracující gramatiky typu 1 (až na to, že zde je třeba mít na levé straně pravidla pouze jeden neterminál, nikoliv řetězec obsahující neterminál).

Věta 6.1 (Převod na nezkracující gramatiku)

Nechť G je bezkontextová gramatika. Pak existuje bezkontextová gramatika G' nezkracující taková, že $L(G') = L(G)$.

Postup konstrukce (Intuitivní metoda): Je dána bezkontextová gramatika $G = (N, T, P, S)$. Chceme sestrojit k ní ekvivalentní nezkracující gramatiku $G' = (N', T, P', S')$. Nejdřív si ukážeme postup, jak upravit pravidla tak, abychom mohli odstranit ε-pravidlo *pro jakýkoliv jiný než startovací symbol*. Postupujeme následovně:

1. Najdeme ε-pravidlo – předpokládejme, že to je $A \to \varepsilon$ pro nějaké $A \in N$. Pak pro všechna pravidla gramatiky, která mají na pravé straně nejméně jeden symbol A:
   $B \to \alpha_0 A \alpha_1 A \alpha_2 A \alpha_3 \ldots A \alpha_{n-1} A \alpha_n$ (přesně n výskytů A v pravidle)
   přidáme do množiny pravidel tato nová pravidla:
   - vynecháme jeden výskyt symbolu A:
     $B \to \alpha_0\ \alpha_1 A \alpha_2 A \alpha_3 \ldots A \alpha_{n-1} A \alpha_n$
     $B \to \alpha_0 A \alpha_1\ \alpha_2 A \alpha_3 \ldots A \alpha_{n-1} A \alpha_n$
     $\ldots$
     $B \to \alpha_0 A \alpha_1 A \alpha_2 A \alpha_3 \ldots A \alpha_{n-1}\ \alpha_n$
   - vynecháme dva výskyty symbolu A:
     $B \to \alpha_0\ \alpha_1\ \alpha_2 A \alpha_3 \ldots A \alpha_{n-1} A \alpha_n$
     $B \to \alpha_0\ \alpha_1 A \alpha_2\ \alpha_3 \ldots A \alpha_{n-1} A \alpha_n$
     $\ldots$
     $B \to \alpha_0 A \alpha_1 A \alpha_2 A \alpha_3 \ldots\ \alpha_{n-1}\ \alpha_n$
   - atd.
   - vynecháme všechny výskyty symbolu A:
     $B \to \alpha_0\ \alpha_1\ \alpha_2\ \alpha_3\ \ldots\ \alpha_{n-1}\ \alpha_n$

2. Odstraníme pravidlo $A \to \varepsilon$.
3. Vrátíme se k bodu 1.

Je třeba si uvědomit, že odstraňováním neterminálů na pravých stranách pravidel můžeme ve snaze o odstranění jednoho ε-pravidla nechtěně vyrobit další ε-pravidla. Proto je třeba postup provádět rekurzívně tak dlouho, dokud v gramatice existují ε-pravidla.

Nyní se zaměříme na případ, kdy v jazyce generovaném gramatikou je prázdné slovo, a tedy máme pravidlo $S \to \varepsilon$. V tomto případě postupujeme stejně jak je popsáno výše, ale navíc musíme zajistit, aby v jazyce gramatiky zůstalo prázdné slovo (potřebujeme ε-pravidlo pro startovací symbol) a zároveň aby se startovací symbol gramatiky nenacházel na pravé straně žádného pravidla.

Předpokládejme tedy, že $\varepsilon \in L(G)$, a již jsme odstranili všechna ε-pravidla tak, jak je popsáno výše (i pravidlo $S \to \varepsilon$). Mezivýsledkem je gramatika $G' = (N, T, P', S)$. Potom vytvoříme gramatiku $G'' = (N \cup \{S'\},\ T,\ P'', S')$ tak, že:

- přidáme nový startovací symbol S' (nově přidaný, tedy $S' \notin N$),
- přidáme pro tento symbol dvě pravidla: $S' \to S \mid \varepsilon$ (při generování neprázdných slov se napojíme na výpočet v původní gramatice a přidáme možnost vygenerování prázdného slova) – tím zajistíme, že bude existovat ε-pravidlo pro startovací symbol, ale zároveň startovací symbol nebude na pravé straně žádného pravidla:
  $P'' = P' \cup \{S' \to S \mid \varepsilon\}$ □

Příklad 6.7

Je dána bezkontextová gramatika:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aAB \mid aBbBc \mid a$
$A \to aA \mid bS \mid a$
$B \to bBA \mid bb \mid \varepsilon$
V této gramatice máme jediné ε-pravidlo – $B \to \varepsilon$. To chceme odstranit.

Vezměme si derivaci
$S \Rightarrow aAB \Rightarrow aA \Rightarrow \ldots.$
V této derivaci jsme v druhém kroku použili právě to pravidlo, které chceme odstranit. Potřebujeme, aby po odstranění tohoto pravidla bylo možné derivaci téhož výstupu provést také, proto musíme upravit množinu pravidel, aby to bylo možné.

Pokud přidáme nové pravidlo $S \to aA$ (což znamená, že na řetězec na pravé straně pravidla $S \to aAB$ jsme „uplatnili“ pravidlo $B \to \varepsilon$), pak bude existovat obdoba výše uvedené derivace:
$S \Rightarrow aA \Rightarrow \ldots$
Možným řešením je tedy „uplatnění“ pravidla, kterého se chceme zbavit, na pravých stranách pravidel. V našem případě tedy upravíme gramatiku následovně:
$G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$ Přidaná pravidla jsou znázorněna červeně.
$S \to aAB \mid aA \mid aBbBc \mid abBc \mid aBbc \mid abc \mid a$
$A \to aA \mid bS \mid a$
$B \to bBA \mid bA \mid bb$

Intuitivní postup ukázaný na předchozím příkladu má jednu vadu – odstraněním jednoho ε-pravidla mohou vzniknout další ε-pravidla. To je sice možné řešit opakovaným použitím postupu, ale především u rozsáhlejších gramatik se tím vystavujeme většímu nebezpečí „spáchání chyby“.

Deterministický (tj. konečný a jednoznačný) exaktní postup si ukážeme v následujícím postupu konstrukce. V něm se jedná především o to, jak určit všechny neterminály, které lze (po různém počtu kroků) přepsat na ε, tuto informaci pak dále využijeme stejným způsobem jako u intuitivní metody (budeme na pravých stranách pravidel postupně vynechávat různé permutace/variace s opakováním symbolů majících tuto vlastnost).

Postup konstrukce (Iterační exaktní metoda): Je dána gramatika $G = (N, T, P, S)$. Chceme sestrojit k ní ekvivalentní nezkracující gramatiku $G' = (N', T, P', S')$. Podobně jako v předchozím popisu konstrukce, i zde budeme nejdřív odstraňovat ε-pravidla bez ohledu na to, zda se jedná o startovací symbol (tj. pokud $\varepsilon \in L(G)$, dočasně toto slovo z jazyka odstraníme), a pak vyřešíme přidání prázdného slova, pokud do jazyka má patřit.

Vytvoříme množinu N_ε, což je množina všech neterminálů, které lze (po jakémkoliv počtu kroků) přepsat na ε. Tuto množinu budeme tvořit iterativním postupem.

- Jako bázi použijeme prázdnou množinu:
  $N_{\varepsilon,0} = \emptyset$
- V prvním kroku iterace přidáme všechny neterminály, pro které existuje ε-pravidlo:
  $N_{\varepsilon,1} = N_{\varepsilon,0} \cup \{X \in N \ ; \ (X \to \varepsilon) \in P\} = N_{\varepsilon,0} \cup \left\{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in N_{\varepsilon,0}^*\right\}$
  (uvědomte si, že $\emptyset^* = \{\varepsilon\}$)
- V dalších krocích postupujeme podle tohoto schématu:
  $N_{\varepsilon,i} = N_{\varepsilon,i-1} \cup \left\{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in N_{\varepsilon,i-1}^*\right\}$
- Pokud $N_{\varepsilon,i} = N_{\varepsilon,i-1} = N_\varepsilon$, končíme, máme množinu všech neterminálů, které lze po nějakém počtu kroků přepsat na ε.

Slovně: V každém kroku (kromě báze) přidáváme všechny neterminály, které lze přepsat na řetězec skládající se *pouze* z těch neterminálů, které jsme do množiny přidali v předchozích krocích. V prvním kroku jde přímo o neterminály, pro které existují ε-pravidla, v druhém kroku přidáme neterminály s pravidlem, kde na pravé straně jsou pouze neterminály, pro které existují ε-pravidla, atd.

Množinu N_ε použijeme stejně jak je naznačeno v intuitivním postupu konstrukce – přidáváme nová pravidla podle původních pravidel, ve kterých postupně vypouštíme různý počet a různé kombinace neterminálů umístěných v množině N_ε. Pokud vznikne ε-pravidlo, ignorujeme je.

Zbývá dořešit případ, kdy v jazyce generovaném gramatikou je prázdné slovo (to poznáme podle toho, že se do množiny N_ε dostane startovací symbol). Postupujeme stejně jako v předchozím popisu konstrukce.

Předpokládejme tedy, že $\varepsilon \in L(G)$, a již jsme odstranili všechna ε-pravidla tak, jak je popsáno výše (i pravidlo $S \to \varepsilon$). Mezivýsledkem je gramatika $G' = (N, T, P', S)$. Potom vytvoříme gramatiku $G'' = (N \cup \{S'\}, \ T, \ P'', S')$ tak, že:

- přidáme nový startovací symbol S' (nově přidaný, tedy $S' \notin N$),
- přidáme pro tento symbol dvě pravidla: $P'' = P' \cup \{S' \to S \mid \varepsilon\}$ □

Příklad 6.8

Postup si ukážeme na stejné gramatice, která byla v předchozím příkladu:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aAB \mid aBbBc \mid a$
$A \to aA \mid bS \mid a$
$B \to bBA \mid bb \mid \varepsilon$

Vytvoříme množinu neterminálů, které lze po nějakém počtu kroků přepsat na ε.
$N_{\varepsilon,0} = \emptyset$
$N_{\varepsilon,1} = \emptyset \cup \{B\} = \{B\}$
$N_{\varepsilon,2} = \{B\} \cup \emptyset = \{B\} = N_{\varepsilon,1} = N_\varepsilon$

Tady to bylo jednoduché, žádná zvláštní rekurze není nutná. Množina je jednoprvková, proto v pravidlech postupně vypustíme různé počty a umístění symbolu B:
$G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$
$S \to aAB \mid aA \mid aBbBc \mid abBc \mid aBbc \mid abc \mid a$
$A \to aA \mid bS \mid a$
$B \to bBA \mid bA \mid bb$

Příklad 6.9

Nyní postup vyzkoušíme na gramatice, kde již k rekurzi dojde. Navíc v jazyce gramatiky je také prázdné slovo, což není na první pohled vidět.
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aBA \mid BB \mid ac$
$A \to BS \mid aA \mid a$
$B \to bB \mid aA \mid \varepsilon$

Sestrojíme množinu neterminálů přepsatelných na ε.
$N_{\varepsilon,0} = \emptyset$
$N_{\varepsilon,1} = \emptyset \cup \{B\} = \{B\}$
$N_{\varepsilon,2} = \{B\} \cup \{S\} = \{B, S\}$ (pravidlo $S \to BB$)
$N_{\varepsilon,3} = \{B, S\} \cup \{A\} = \{B, S, A\}$ (pravidlo $A \to BS$)
$N_{\varepsilon,4} = \{B, S, A\} \cup \emptyset = \{B, S, A\} = N_{\varepsilon,3} = N_\varepsilon$

Teď nás čeká přidávání nových pravidel – v pravidlech postupně vypouštíme různé variace (s opakováním) neterminálů, které jsme získali v množině N_ε (v tomto případě všech neterminálů). Zatím se nejedná o výsledek, pro gramatiku G' zatím platí $L(G') = L(G) - \{\varepsilon\}$.
$G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$
$S \to aBA \mid aA \mid aB \mid a \mid BB \mid B \mid ac$
$A \to BS \mid S \mid B \mid aA \mid a$ (pravidlo $A \to a$ nemusíme přidávat, už tam je)
$B \to bB \mid b \mid aA \mid a$

Protože je v množině N_ε startovací symbol S, znamená to, že do jazyka gramatiky patří prázdné slovo. Proto je třeba provést ještě jednu úpravu:

$G'' = (\{S', S, A, B\},\ \{a, b, c\},\ P'',\ S')$
$S' \to S \mid \varepsilon$
$S \to aBA \mid aA \mid aB \mid a \mid BB \mid B \mid ac$
$A \to BS \mid S \mid B \mid aA \mid a$
$B \to bB \mid b \mid aA \mid a$

Důkaz (Věta 6.1): Je zřejmé, že výsledná gramatika je nezkracující (protože neobsahuje žádná ε-pravidla). Ukážeme, že změny, které jsme v gramatice provedli, jsou *ekvivalentní*, tj. nemění jazyk generovaný gramatikou.

Nejdřív probereme případ, kdy $\varepsilon \notin L(G)$. Vycházíme z gramatiky $G = (N, T, P, S)$ a sestrojená gramatika je $G' = (N, T, P', S)$.

Dokazujeme $L(G) \subseteq L(G')$:
Zde si stačí uvědomit, že místo ε-pravidel jsme do gramatiky zařadili pravidla odpovídající původním pravidlům gramatiky, kde jsme odstranili jednotlivé neterminály přepsatelné na ε v různých kombinacích, čímž jsme simulovali použití ε-pravidla na řetězec, jehož podřetězcem je pravá strana některého původního pravidla. Důsledkem je dokonce možné zkrácení derivace.

V každém případě ke každé derivaci v gramatice G dokážeme sestrojit ekvivalentní derivaci téhož slova v gramatice G'.

Dokazujeme $L(G) \supseteq L(G')$:
V množině pravidel gramatiky G' máme kromě původních neepsilonových pravidel nová pravidla, která však respektují původní pravidla v kombinaci s již odstraněnými ε-pravidly. Tedy pro kteroukoliv derivaci v gramatice G' dokážeme sestrojit ekvivalentní derivaci téhož slova v G.

Zbývá případ, kdy $\varepsilon \in L(G)$. Pokud $\varepsilon \in L(G)$, pak podle výše uvedeného postupu máme výslednou gramatiku $G'' = (N \cup \{S'\},\ T,\ P' \cup \{S' \to S \mid \varepsilon\},\ S')$.

Pro derivace slov $w \in L(G)$ takových, že $|w| \geq 1$, platí totéž co v předchozí části důkazu, jen je odvození o jeden krok delší:
$S' \Rightarrow S \Rightarrow^* w$

Zaměřme se tedy na vygenerování slova ε. V gramatice G'' použijeme derivaci $S' \Rightarrow \varepsilon$, tedy $\varepsilon \in L(G'')$. Proto jestliže $\varepsilon \in L(G)$, pak $\varepsilon \in L(G'')$.

Jestliže $\varepsilon \in L(G'')$, pak existuje pravidlo $S' \to \varepsilon$ a to se do množiny P'' dostalo jen tehdy, pokud $S \in N_\varepsilon$. Do množiny N_ε se S dostalo jen tehdy, pokud v G existuje derivace $S \Rightarrow^* \varepsilon$, a tedy $\varepsilon \in L(G)$. Proto pokud $\varepsilon \in L(G'')$, pak také $\varepsilon \in L(G)$. □

6.4 Redukovaná gramatika

Podobně jako jsme redukovali konečný automat, můžeme redukovat také bezkontextovou gramatiku. Budeme odstraňovat *nadbytečné symboly* (ze kterých nelze vygenerovat žádné terminální slovo) a *nedostupné symboly* (nenacházejí se v žádné derivaci ze startovacího symbolu).

Definice 6.4 (Nadbytečný neterminál)

$X \in N$ je nadbytečný neterminál v gramatice $G = (N, T, P, S)$, pokud neexistuje žádné terminální slovo, které lze z tohoto symbolu vygenerovat, tj. neexistuje derivace

$$X \Rightarrow^* w, \quad w \in T^* \tag{6.4}$$

Nadbytečné symboly jsou jen neterminální (z terminálního symbolu nic negenerujeme, ani terminální slovo). Naproti tomu nedostupné mohou být jak neterminály, tak i terminály:

Definice 6.5 (Nedostupný symbol)

Symbol $X \in (N \cup T)$ je nedostupný v gramatice $G = (N, T, P, S)$, jestliže se nemůže objevit v žádné větné formě, tj. neexistuje derivace

$$S \Rightarrow^* \alpha X \beta, \quad \alpha, \beta \in (N \cup T)^* \tag{6.5}$$

Definice 6.6 (Redukovaná gramatika)

Bezkontextová gramatika $G = (N, T, P, S)$ je redukovaná, pokud neobsahuje žádné nadbytečné a nedostupné symboly.

Odstraňujeme nejdřív nadbytečné neterminály a až potom nedostupné symboly.

6.4.1 Odstranění nadbytečných neterminálů

Věta 6.2 (Odstranění nadbytečných neterminálů)

Ke každé bezkontextové gramatice G existuje gramatika bez nadbytečných neterminálů G' taková, že $L(G) = L(G')$.

Postup konstrukce: Zde použijeme iterativní metodu – sestrojíme množinu neterminálů E_{def}, ze kterých lze vygenerovat terminální řetězec. Nadbytečné neterminály jsou pak ty, které do této množiny nepatří. Původní gramatiku označíme $G = (N, T, P, S)$, sestrojíme gramatiku bez nadbytečných neterminálů $G' = (N', T, P', S)$.

- Jako bázi použijeme množinu terminálních symbolů (z výsledné množiny je pak odstraníme): $E_0 = T$
- V prvním kroku iterace přidáme všechny neterminály, pro které existuje terminální nebo ε-pravidlo:
  $E_1 = E_0 \cup \{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in T^*\} = E_0 \cup \{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in E_0^*\}$
- V každém dalším kroku iterace přidáváme další neterminály, pro které existuje pravidlo, na jehož pravé straně jsou pouze symboly zařazené do množiny v předchozích krocích:
  $E_i = E_{i-1} \cup \{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in E_{i-1}^*\}$

- Pokud se už množina nemění (není co přidat), končíme:
  $E_i = E_{i-1} = E_{def}$

V množině E_{def} máme pouze takové symboly, ze kterých je možné vygenerovat terminální slovo (prázdné slovo je taky terminální).

Stanovíme novou množinu neterminálů: $N' = N \cap E_{def}$. Nová množina pravidel bude obsahovat pouze ta pravidla, která mají na levé a pravé straně pouze symboly z množiny E_{def}:
$P' = \left\{(A \to \alpha) \in P \ ; \ A \in E_{def}, \ \alpha \in E_{def}^*\right\}$ □

Příklad 6.10

V následující gramatice odstraníme nadbytečné symboly:
$G = (\{S, A, B, C, D\}, \ \{a, b, c, d\}, \ P, \ S)$
$S \to aAbC \mid c$
$A \to aA \mid Cc$
$B \to cB \mid dD$
$C \to cB \mid aA \mid b$
$D \to Bd$

Použijeme výše uvedený iterativní postup a sestrojíme množinu E_{def}. Pak určíme novou množinu neterminálů a novou množinu pravidel.
$E_0 = T = \{a, b, c, d\}$ (báze iterace)
$E_1 = \{a, b, c, d, S, C\}$ (podle $S \to c, \ C \to b$)
$E_2 = \{a, b, c, d, S, C, A\}$ (podle $A \to Cc$)
$E_3 = E_2 = E_{def}$

Nová množina neterminálů je $N' = \{S, A, B, C, D\} \cap \{a, b, c, d, S, C, A\} = \{S, A, C\}$. Nová množina pravidel P' bude obsahovat pouze ta pravidla, která mají na levé i pravé straně pouze symboly z množiny E_{def}, tedy celá výsledná gramatika vypadá takto:
$G' = (\{S, A, C\}, \ \{a, b, c, d\}, \ P', \ S)$
$S \to aAbC \mid c$
$A \to aA \mid Cc$
$C \to aA \mid b$

Poznámka:

Co by to znamenalo, kdyby v množině E_{def} nebyl startovací symbol gramatiky? Znamenalo by to, že v gramatice nelze vygenerovat žádné slovo. Z jednotlivých neterminálů sice může být možné nějaké terminální slovo získat, ale víme, že každá derivace musí začínat startovacím symbolem. Jazyk takové gramatiky by byl prázdná množina.

6.4.2 Odstranění nedostupných symbolů

Věta 6.3 (Redukce gramatiky)

Ke každé bezkontextové gramatice G existuje redukovaná gramatika (bez nadbytečných a nedostupných symbolů) G' taková, že $L(G) = L(G')$.

Odstraněním nadbytečných symbolů jsme se zabývali v předchozím textu, zbývá popsat postup odstranění nedostupných symbolů.

Postup konstrukce: Opět použijeme iterativní metodu – sestrojíme množinu symbolů, které jsou dostupné (vyskytují se v některé větné formě v derivaci ze startovacího symbolu). Nedostupné symboly jsou ty, které do této množiny nepatří. Původní gramatiku označíme $G = (N, T, P, S)$, sestrojíme gramatiku $G' = (N', T', P', S)$.

- Jako bázi použijeme množinu obsahující startovací symbol gramatiky:
  $S_0 = \{S\}$
- V prvním kroku iterace přidáme všechny symboly z pravé strany pravidel pro startovací symbol, protože právě tyto symboly jsou dostupné ze startovacího jedním krokem:
  $S_1 = S_0 \cup \{X \in (N \cup T)\ ;\ (S \to \alpha) \in P,\ |\alpha|_X \geq 1\}$
  $= S_0 \cup \{X \in (N \cup T)\ ;\ (A \to \alpha) \in P,\ A \in S_0,\ |\alpha|_X \geq 1\}$
- V každém dalším kroku iterace přidáváme další symboly, které jsou v jednom kroku dosažitelné ze symbolů množiny z předchozího kroku:
  $S_i = S_{i-1} \cup \{X \in (N \cup T)\ ;\ (A \to \alpha) \in P,\ A \in S_{i-1},\ |\alpha|_X \geq 1\}$
- Pokud se už množina nemění (není co přidat), končíme:
  $S_i = S_{i-1} = S_{def}$

V množině S_{def} máme pouze takové symboly, které jsou dostupné ze startovacího symbolu.

Stanovíme novou množinu neterminálů a terminálů: $N' = N \cap S_{def}$, $T' = T \cap S_{def}$. Nová množina pravidel bude obsahovat pouze ta pravidla, která mají na levé a pravé straně pouze symboly z množiny S_{def}:
$P' = \left\{(A \to \alpha) \in P\ ;\ A \in S_{def},\ \alpha \in S_{def}^*\right\}$ □

Příklad 6.11

Druhou část redukce (odstranění nedostupných symbolů) si ukážeme na této gramatice:
$G = (\{S, A, B, C\},\ \{a, b, c\},\ P,\ S)$
$S \to aA \mid bB \mid c$
$A \to cS \mid aA$
$B \to bB \mid cAB \mid b$
$C \to aA \mid b$

Sestrojíme množinu S_{def}. Pak určíme novou množinu neterminálů, terminálů a pravidel.
$S_0 = \{S\}$ (báze)
$S_1 = \{S, a, A, b, B, c\}$ (na pravých stranách pravidel pro symbol S)

$S_2 = \{S, a, A, b, B, c, b\}$ (podle pravidel pro A a B)
$S_3 = S_2 = S_{def}$

Nová množina neterminálů je $N \cap S_{def} = \{S, A, B, C\} \cap \{S, a, A, b, B, c, b\} = \{S, A, B\}$, nová množina terminálů je $T \cap S_{def} = \{a, b, c\} \cap \{S, a, A, b, B, c, b\} = \{a, b, c\}$ (zde se nic nemění). Množina pravidel P' bude také protříděna, výsledná gramatika je následující:
$G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$
$S \to aA \mid bB \mid c$
$A \to cS \mid aA$
$B \to bB \mid cAB \mid b$

Důkaz (Věta 6.3): To, že výsledkem dvou výše uvedených postupů je redukovaná gramatika (tedy že odstraňují nadbytečné a nedostupné symboly), plyne ze samotného postupu – do množiny E_{def} se dostanou pouze ty symboly, ze kterých lze (konečným počtem kroků) vygenerovat terminální řetězec, do množiny S_{def} řadíme pouze ty symboly, které lze vygenerovat ze startovacího symbolu (opět po konečném počtu kroků).

Algoritmy jsou také konečné, počet kroků iterace (řazení prvků do množin) je vždy maximálně takový, kolik je prvků v množině N, resp. $N \cup T$, což jsou konečné množiny.

Zbývá dokázat ekvivalenci jazyka původní a vytvořené gramatiky. Označme $G = (N, T, P, S)$ původní gramatiku, dále $G' = (N', T, P', S)$ gramatiku po odstranění nadbytečných symbolů, $G'' = (N'', T', P'', S)$ výslednou redukovanou gramatiku.

Dokazujeme $L(G) \subseteq L(G')$:
Vezměme jakékoliv slovo $w \in L(G)$.
$\Rightarrow$ existuje derivace slova w v gramatice G: $S \Rightarrow^* w$
$\Rightarrow$ všechny neterminály ve větných formách této derivace patří do E_{def}
(můžeme si představit, že obsah E_{def} tvoříme takto:
∗ procházíme různé možné derivace *proti směru odvozování* od věty w na konci derivace,
∗ do E_{def} řadíme všechny symboly, které v procházených větných formách najdeme)
$\Rightarrow$ všechny symboly nacházející se ve větných formách derivace $S \Rightarrow^* w$ patří do E_{def}
$\wedge$ všechna pravidla v derivaci použitá patří do množiny P'
$\Rightarrow$ derivace $S \Rightarrow^* w$ existuje i v gramatice G'
$\Rightarrow$ $w \in L(G')$

Dokazujeme $L(G') \subseteq L(G'')$:
Tato část důkazu bude podobná. Vezměme jakékoliv slovo $w \in L(G')$.
$\Rightarrow$ existuje derivace slova w v gramatice G': $S \Rightarrow^* w$
$\Rightarrow$ všechny symboly ve větných formách této derivace patří do S_{def}
(můžeme si představit, že obsah S_{def} tvoříme takto:
∗ procházíme různé možné derivace *ve směru odvozování* od symbolu S,
∗ do S_{def} řadíme všechny symboly, které v procházených větných formách najdeme)
$\Rightarrow$ všechny symboly nacházející se ve větných formách derivace $S \Rightarrow^* w$ patří do S_{def}
$\wedge$ všechna pravidla v derivaci použitá patří do množiny P''

$\Rightarrow$ derivace $S \Rightarrow^* w$ existuje i v gramatice G''
$\Rightarrow$ $w \in L(G'')$

Dokazujeme $L(G) \supseteq L(G') \supseteq L(G'')$:
Obě inkluze platí, protože v obou postupech jsme pravidla pouze vyřazovali, žádné nové pravidlo jsme nepřidali. Proto pokud pro (jakékoliv) slovo $w \in T^*$ existuje derivace v gramatice G'', pak stejná derivace existuje i v G' a v G.

Tím jsme dokázali ekvivalenci gramatik G, G' a G'' a korektnost a úplnost postupu redukce gramatiky. □

Příklad 6.12

Ukážeme si, proč je třeba nejdřív odstranit nadbytečné a až potom nedostupné symboly.
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to Aa \mid a$
$A \to ABc$
$B \to bB \mid b$
V prvním sloupci následující tabulky je správný postup (nejdřív odstraníme nadbytečné a pak nedostupné symboly), v druhém špatný postup (kdy tyto dva algoritmy zaměníme).

Správně:	**Špatně:**
$E_0 = T$ $E_1 = E_0 \cup \{S, B\}$ $E_2 = E_1,\ N' = \{S, B\}$ *První úprava:* $G' = (\{S, B\},\ \{a, b\},\ P',\ S)$ $S \to a$ $B \to bB \mid b$	$S_0 = \{S\}$ $S_1 = S_0 \cup \{A, a\}$ $S_2 = S_1 \cup \{B, c\}$ $S_3 = S_2 \cup \{b\}$ $S_4 = S_3,\ N' = \{S, A, B\}$ *První úprava:* $G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$ $S \to Aa \mid a$ $A \to ABc$ $B \to bB \mid b$
$S_0 = \{S\}$ $S_1 = S_0 \cup \{a\}$ $S_2 = S_1,\ N'' = \{S\}$ *Druhá úprava:* $G'' = (\{S\},\ \{a\},\ P'',\ S)$ $S \to a$	$E_0 = T$ $E_1 = E_0 \cup \{S, B\}$ $E_2 = E_1,\ N'' = \{S, B\}$ *Druhá úprava:* $G'' = (\{S, B\},\ \{a, b\},\ P'',\ S)$ $S \to a$ $B \to bB \mid b$

Jak vidíme, po úpravách v druhém sloupci nám v gramatice zůstal neterminál B, který je ve skutečnosti nedostupný ze startovacího symbolu.

Kapitola 7

Zásobníkový automat

V této kapitole se budeme zabývat zásobníkovými automaty, který jsme si stručně popsali již na začátku semestru. Po definici zásobníkového automatu se budeme zabývat jeho některými vlastnostmi a vztahem k bezkontextovým gramatikám.

7.1 Definice zásobníkového automatu

Jak víme, *zásobník* je dynamická struktura podobná frontě nebo seznamu, která se používá stylem LIFO (Last-in First-out): ten prvek, který jsme vložili jako poslední, jako první vyjmeme.

Zásobníkový automat získáme tak, že konečný automat obohatíme o zásobníkovou pásku a zajistíme, aby byl výpočet řízen především obsahem zásobníku. Tento model pracuje takto:

- vyjme symbol na vrcholu zásobníku,
- může nebo nemusí přečíst symbol ze vstupní pásky, pokud přečte, posune se o pole dál,
- dále se rozhoduje podle
  - svého vnitřního stavu,
  - symbolu, který vyndal ze zásobníku,
  - pokud četl ze vstupní pásky, pak i podle přečteného symbolu,
- činnost automatu v jednom kroku spočívá v přechodu do některého dalšího stavu a v uložení řetězce znaků do zásobníku.

Definice 7.1 (Zásobníkový automat)

Zásobníkový automat *je uspořádaná sedmice* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, *kde je*

Q ... *neprázdná konečná množina stavů*

Σ ... *neprázdná konečná abeceda automatu*

Γ ... *neprázdná konečná zásobníková abeceda*

δ ... *přechodová funkce, definovaná níže*

q_0 ... *počáteční stav,* $q_0 \in Q$

Z_0 ... *počáteční zásobníkový symbol,* $Z_0 \in \Gamma$

F ... *množina koncových stavů,* $F \subseteq Q$ *(může být i prázdná)*

Přechodová funkce δ *zásobníkového automatu* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je definována takto:*
$\delta : \ Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$ *(zápis pomocí množin)*
$\delta(q, a, Z) \ni (r, \gamma), \quad$ *kde* $q, r \in Q, \ a \in (\Sigma \cup \{\varepsilon\}), \ Z \in \Gamma, \ \gamma \in \Gamma^*$ *(symbolický zápis)*

Zásobníkový automat je obecně nedeterministický. Oproti konečnému automatu máme navíc zásobníkovou abecedu, počáteční zásobníkový symbol a bohatší přechodovou funkci.

Poznámka:

Počáteční zásobníkový symbol můžeme brát jako „zarážku“ na dně zásobníku. Je to obdoba ukazatele null (příp. nil) v dynamických datových strukturách programovacích jazyků.

Definice 7.2 (Konfigurace, počáteční a koncová konfigurace)

Konfigurace *zásobníkového automatu* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je uspořádaná trojice* (q, w, γ), *kde* $q \in Q, \ w \in \Sigma^*$ *(nepřečtená část vstupní pásky) a* $\gamma \in \Gamma^*$ *(momentální obsah zásobníku).*

Počáteční konfigurace *je konfigurace* (q_0, w_0, Z_0), *kde* q_0 *je počáteční stav automatu,* w_0 *je celé zpracovávané slovo a* Z_0 *je počáteční zásobníkový symbol.*

Na začátku výpočtu tedy máme na vstupu celé slovo, začínáme v počátečním stavu a v zásobníku máme pouze počáteční zásobníkový symbol. Definice koncové konfigurace závisí na typu zásobníkového automatu, definujeme ji proto až později.

Definice 7.3 (Přechod mezi konfiguracemi)

Relaci přechodu mezi konfiguracemi zásobníkového automatu $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *značíme symbolem* $\vdash$ *a definujeme ji takto:*

$$(q_i, a\,\omega, Z\beta) \ \vdash \ (q_j, \omega, \gamma\beta) \quad \overset{def}{\iff} \quad \delta(q_i, a, Z) \ni (q_j, \gamma), \tag{7.1}$$

kde $q_i, q_j \in Q, \quad a \in (\Sigma \cup \{\varepsilon\}), \quad \omega \in \Sigma^*, \quad Z \in \Gamma, \quad \beta, \gamma \in \Gamma^*$

Symbol $\vdash^*$ značí reflexivní a tranzitivní uzávěr relace $\vdash$, symbol $\vdash^+$ je tranzitivní uzávěr této relace, symbol $\vdash^n$ znamená přesně n přechodů mezi konfiguracemi.

Rozeznáváme tři základní typy zásobníkových automatů:

- *Zásobníkový automat končící přechodem do koncového stavu* – způsob ukončení výpočtu je podobný jako u konečného automatu: je třeba
  - přečíst celý vstup a zároveň
  - přesunout se do některého ze stavů z množiny F (do některého koncového stavu).
- *Zásobníkový automat končící s prázdným zásobníkem*: je třeba
  - přečíst celý vstup a zároveň
  - vyprázdnit celý zásobník (včetně počátečního zásobníkového symbolu).
- *Zásobníkový automat končící přechodem do koncového stavu a s prázdným zásobníkem*: je třeba splnit vše, co je v předchozích odrážkách (přečtený vstup, koncový stav, prázdný zásobník).

Definice 7.4 (Typy ZA, koncová konfigurace a rozpoznávaný jazyk ZA)

Zásobníkový automat končící přechodem do koncového stavu *je* $\mathcal{A}_F = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *s koncovou konfigurací* $(q_f, \varepsilon, \gamma),\ q_f \in F,\ \gamma \in \Gamma^*$ *a rozpoznávaný jazyk je*

$$L(\mathcal{A}_F) = \{w \in \Sigma^* \ ;\ (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \gamma),\ q_f \in F,\ \gamma \in \Gamma^*\}\,. \tag{7.2}$$

Zásobníkový automat končící s prázdným zásobníkem *je* $\mathcal{A}_\emptyset = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ *s koncovou konfigurací* $(q, \varepsilon, \varepsilon),\ q \in Q$ *a rozpoznávaný jazyk je*

$$L(\mathcal{A}_\emptyset) = \{w \in \Sigma^* \ ;\ (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon),\ q \in Q\} \tag{7.3}$$

Zásobníkový automat končící přechodem do koncového stavu a s prázdným zásobníkem *je* $\mathcal{A}_{F,\emptyset} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *s koncovou konfigurací* $(q_f, \varepsilon, \varepsilon),\ q_f \in F$ *a rozpoznávaný jazyk je*

$$L(\mathcal{A}_{F,\emptyset}) = \{w \in \Sigma^* \ ;\ (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \varepsilon),\ q_f \in F\} \tag{7.4}$$

Když vytváříme zásobníkový automat, musíme předem vědět, kterého typu bude, podle toho konstruujeme přechodovou funkci. Nicméně – pokud vytvoříme jeden z těchto typů, není problém sestrojit ekvivalentní zásobníkový automat jiného typu.

Poznámka:

Při sestavování zásobníkového automatu postupujeme poněkud systematičtěji než u konečného automatu. Všímáme si *struktury slov* jazyka. Rozdělíme (obecné) slovo na části a stanovíme, jak se v jednotlivých částech má automat chovat. Průběh zpracování v těchto částech odlišíme stavem a určíme, co v kterém stavu má být v zásobníku a jak se má se zásobníkem zacházet.

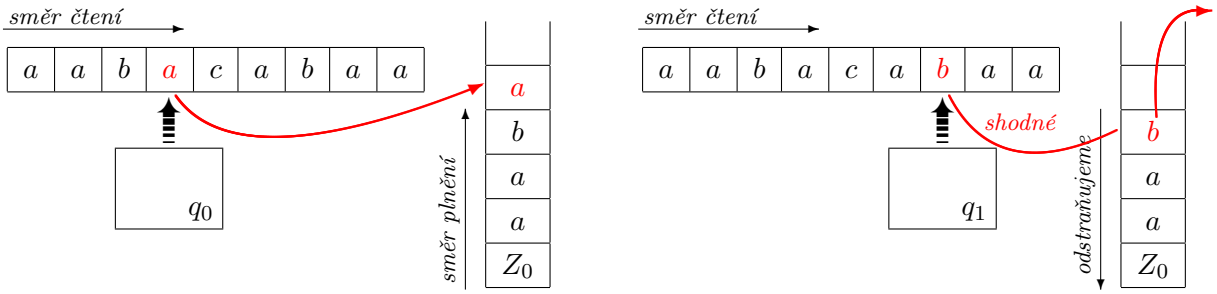
Příklad 7.1

Sestrojíme zásobníkový automat rozpoznávající jazyk $L = \left\{wcw^R \ ;\ w \in \{a, b\}^*\right\}$

Vytvoříme zásobníkový automat rozpoznávající prázdným zásobníkem:
$\mathcal{A}_\emptyset = (Q,\ \{a, b\},\ \Gamma,\ q_0,\ Z_0,\ \delta,\ \emptyset)$
Slova tohoto jazyka se skládají ze dvou částí oddělených symbolem c. Pro každou část určíme stav, tedy potřebujeme dva stavy: $Q = \{q_0, q_1\}$. Jak se náš automat má v těchto stavech chovat?

- Ve stavu q_0 načítáme první část slova, pouze ukládáme do zásobníku:
  - načteme symbol ze vstupu,
  - sice vyzvedneme symbol z vrcholu zásobníku, ale vrátíme ho zpátky beze změny,
  - načtený symbol uložíme také do zásobníku.
- Ve stavu q_1 načítáme druhou část slova a kontrolujeme s obsahem zásobníku:
  - načteme symbol ze vstupu,
  - vyzvedneme symbol z vrcholu zásobníku,
  - pokud jsou tyto dva symboly shodné, pak je vše v pořádku.

Přechod mezi stavy q_0 a q_1 nastává při načtení „oddělujícího“ symbolu c.



Vytvoříme přechodovou funkci. Nejdřív budeme předpokládat, že je na vstupu slovo obsahující i jiné symboly než jen c (tj. délka ≥ 2), pak přidáme i možnost zpracování slova o délce 1.

- Začneme ve stavu q_0, na vrcholu zásobníku je symbol Z_0 (v zásobníku zatím nic jiného nemáme); zásobníkový symbol sice vyjmeme ze zásobníku (musíme), ale opět ho tam vrátíme a přidáme symbol, který jsme načetli ze vstupu. Na vstupu může být buď a nebo b.
  $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  $\delta(q_0, b, Z_0) = (q_0, bZ_0)$
- Stejně budeme reagovat i v dalších krocích (jsme pořád v první části slova). Ať minimalizujeme počet řádků, použijeme zástupný symbol X znamenající a nebo b:
  $\delta(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
  $\delta(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
- Jsme na hranici mezi dvěma částmi slova, načteme c, změníme stav (zásobník necháme jak je, vrátíme vyjmutý symbol a nebudeme přidávat c):
  $\delta(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
- V druhé části slova provádíme synchronizaci obou částí – první část máme v zásobníku (v přesně opačném pořadí než jak tento řetězec byl na vstupu), druhou na vstupu, budeme kontrolovat, jestli je na obou místech totéž, do zásobníku nebudeme nic ukládat:
  $\delta(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
- Zpracovali jsme celou druhou část slova ze vstupu, v zásobníku by měl zůstat už jen symbol Z_0, tedy ukončíme výpočet:
  $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$
- Ještě ošetříme případ, kdy bude na vstupu nejkratší slovo jazyka, c:
  $\delta(q_0, c, Z_0) = (q_1, \varepsilon)$

Celá definice tohoto zásobníkového automatu je následující:

$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \Gamma,\ q_0,\ Z_0,\ \delta,\ \emptyset)$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
$\delta(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
$\delta(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
$\delta(q_0, c, Z_0) = (q_1, \varepsilon)$
$\delta(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

Zápis by se dal ještě více zkrátit, například u řádků z prvního bodu postupu. Zásobníková abeceda je $\Gamma = \{Z_0, a, b\}$, řadíme tam všechny symboly, které se mohou dostat do zásobníku.

Podíváme se na zpracování několika slov patřících do jazyka L. Nejdřív slovo *abcba*:

- V prvním kroku použijeme předpis $\delta(q_0, a, Z_0) = (q_0, aZ_0)$:
  $(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0)$

- V druhém kroku použijeme předpis $\delta(q_0, b, a) = (q_0, ba)$:
  $(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0) \vdash (q_0, cba, baZ_0)$
- V pátém kroku to bude předpis $\delta(q_0, c, b) = (q_1, b)$:
  $\ldots \vdash (q_0, cba, baZ_0) \vdash (q_1, ba, baZ_0)$
- V dalším kroku začneme synchronizovat – srovnávat, předpis $\delta(q_1, b, b) = (q_1, \varepsilon)$:
  $\ldots \vdash (q_1, ba, baZ_0) \vdash (q_1, a, a, Z_0)$
- Pokračujeme předpisem $\delta(q_1, a, a)$:
  $\ldots \vdash (q_1, a, a, Z_0) \vdash (q_1, \varepsilon, Z_0)$
- V posledním kroku uklidíme zásobník předpisem $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$:
  $\ldots \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Celý výpočet:
$(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0) \vdash (q_0, cba, baZ_0) \vdash (q_1, ba, baZ_0) \vdash (q_1, a, a, Z_0) \vdash (q_1, \varepsilon, Z_0) \vdash$
$\vdash (q_1, \varepsilon, \varepsilon)$

Teď trochu delší slovo *aabacabaa*:
$(q_0, aabacabaa, Z_0) \vdash (q_0, abacabaa, aZ_0) \vdash (q_0, bacabaa, aaZ_0) \vdash (q_0, acabaa, baaZ_0) \vdash$
$\vdash (q_0, cabaa, abaaZ_0) \vdash (q_1, abaa, abaaZ_0) \vdash (q_1, baa, baa, Z_0) \vdash (q_1, aa, aaZ_0) \vdash$
$\vdash (q_1, a, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Nejkratší slovo jazyka c – stačí nám jeden krok:
$(q_0, c, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Dáme na vstup několik slov nepatřících do jazyka $L(\mathcal{A})$:
$(q_0, ab, Z_0) \vdash (q_0, b, aZ_0) \vdash (q_0, \varepsilon, baZ_0) \vdash$ nelze pokračovat, $ab \notin L(\mathcal{A})$

$(q_0, ca, Z_0) \vdash (q_1, a, Z_0) \vdash (q_1, a, \varepsilon) \vdash$ nelze pokračovat, $ca \notin L(\mathcal{A})$
Všimněte si posledního kroku – můžeme použít $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$, třebaže vstup není prázdný.

$(q_0, \varepsilon, Z_0) \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A})$

Příklad 7.2

Sestrojíme zásobníkový automat končící v koncovém stavu pro jazyk $L = \{a^n b^n \ ; \ n \geq 0\}$.

Promyslíme si, jak má vypadat přechodová funkce a které stavy budeme potřebovat. Pro první polovinu slova budeme mít stav q_0, pro druhou stav q_1. První polovinu slova budeme jen načítat do zásobníku, kdežto při načítání druhé poloviny budeme naopak zásobník vyprazdňovat a zároveň srovnávat se vstupem (na jeden symbol b na vstupu musí být jeden symbol a v zásobníku).

- Pro první polovinu slova:
  $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  $\delta(q_0, a, a) = (q_0, aa)$ dohromady: $\delta(q_0, a, X) = (q_0, aX)$, kde $X \in \{Z_0, a\}$
- Přechod do druhé poloviny slova:
  $\delta(q_0, b, a) = (q_1, \varepsilon)$
- Druhá polovina slova:
  $\delta(q_1, b, a) = (q_1, \varepsilon)$

- Ukončení:
  $\delta(q_1, \varepsilon, Z_0) = (q_f, \varepsilon)$
- Automat má rozpoznávat i prázdné slovo:
  $\delta(q_0, \varepsilon, Z_0) = (q_f, \varepsilon)$

Výsledný automat:
$\mathcal{A}_F = (\{q_0, q_1, q_f\},\ \{a, b\},\ \{Z_0, a\},\ \delta,\ q_0,\ Z_0,\ \{q_f\})$ s přechodovou funkcí:

$\delta(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_0, \varepsilon, Z_0) = (q_f, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_f, \varepsilon)$

Výpočet několika slov patřících nebo nepatřících do jazyka L:
$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0) \vdash (q_0, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_f, \varepsilon, \varepsilon)$

$(q_0, \varepsilon, Z_0) \vdash (q_f, \varepsilon, \varepsilon)$

$(q_0, aab, Z_0) \vdash (q_0, ab, aZ_0) \vdash (q_0, b, aaZ_0) \vdash (q_1, \varepsilon, aZ_0) \vdash$ nelze pokračovat, $aab \notin L(\mathcal{A})$

$(q_0, abb, Z_0) \vdash (q_0, bb, aZ_0) \vdash (q_1, b, Z_0) \vdash (q_f, b, \varepsilon) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A})$

$(q_0, b, Z_0) \vdash (q_f, b, \varepsilon) \vdash$ nelze pokračovat, $b \notin L(\mathcal{A})$

$(q_0, a, Z_0) \vdash (q_0, \varepsilon, aZ_0) \vdash$ nelze pokračovat, $a \notin L(\mathcal{A})$

V tomto případě jsme sestrojili zásobníkový automat, který nejen rozpoznává koncovým stavem (tj. pokud jsme v koncovém stavu, zde q_f, a zároveň je vstup přečtený, je výpočet úspěšný), ale zároveň je v každé koncové konfiguraci prázdný zásobník. Tedy jsme sestrojili zásobníkový automat končící zároveň v koncovém stavu a s prázdným zásobníkem.

7.2 Vztah mezi typy zásobníkových automatů

Nejčastěji vytváříme zásobníkové automaty rozpoznávající prázdným zásobníkem, ale obecně je jedno, který typ použijeme. Každý z výše uvedených typů zásobníkových automatů totiž dokážeme převést na kterýkoliv jiný typ.

Věta 7.1 (Vztah mezi typy zásobníkových automatů)
Pro zásobníkové automaty končící s prázdným zásobníkem, v koncovém stavu a kombinované (s prázdným zásobníkem a v koncovém stavu) platí následující:

$$\mathscr{L}(\mathcal{A}_\emptyset) \cong \mathscr{L}(\mathcal{A}_F) \cong \mathscr{L}(\mathcal{A}_{F,\emptyset}) \tag{7.5}$$

Postup konstrukce ($\mathscr{L}(\mathcal{A}_\emptyset) \subseteq \mathscr{L}(\mathcal{A}_F)$)**:** Původní zásobníkový automat končící prázdným zásobníkem označíme $\mathcal{A}_\emptyset = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$, nový automat končící v koncovém stavu označíme $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F)$.

K zásobníkovému automatu končícímu s prázdným zásobníkem sestrojíme ekvivalentní zásobníkový automat končící v koncovém stavu takto:

- vytvoříme nový stav q_f, který bude novým koncovým stavem,
- v konfiguraci, ve které bychom v původním automatu končili, provedeme ještě jeden přechod – právě do stavu q_f,
- aby tento přechod vůbec byl možný, potřebujeme mít vlastní zásobníkový symbol ještě pod zásobníkovým symbolem použitým v původním automatu, a na to je třeba brát ohled i na začátku výpočtu.

Uvnitř nového automatu budeme simulovat ten původní. Nový automat bude mít vlastní symbol konce zásobníku, a v prvním kroku výpočtu naváže na výpočet původního automatu tím, že

- přejde do stavu, ve kterém začíná původní automat,
- do zásobníku přidá symbol konce zásobníku původního automatu (svůj tam nechá).

Srovnáme průběh výpočtu v původním a novém zásobníkovém automatu:

Původní automat: $q_0\,[Z_0] \vdash \ldots \vdash q_k\,[Z_0] \vdash q_m\,[\,]$

Nový automat: $q_0'\,[Z_0'] \vdash q_0\,[Z_0 / Z_0'] \vdash \ldots \vdash q_k\,[Z_0 / Z_0'] \vdash q_m\,[Z_0'] \vdash q_f\,[\,]$

Tedy potřebujeme, aby první přechod mezi konfiguracemi vypadal takto: $(q_0', w, Z_0') \vdash (q_0, w, Z_0 Z_0')$ (vstup se v prvním kroku nezmění), čímž se napojíme na výpočet původního automatu. Na závěr výpočtu musíme provést následující: $(q_m, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$, čímž se dostaneme do koncové konfigurace nového automatu. Stavy q_0' a q_f jsou nově přidané, tedy musí platit $q_0' \notin Q,\ q_f \notin Q$.

Přechodovou funkci původního automatu přejmeme a přidáme tyto přechody:
$\delta'(q_0', \varepsilon, Z_0') = (q_0, Z_0 Z_0')$
$\delta'(q, \varepsilon, Z_0') = (q_f, \varepsilon)$ pro všechny stavy $q \in Q$

Výsledný automat vypadá takto:
$\mathcal{A}_F = (Q \cup \{q_0', q_f\},\ \Sigma,\ \Gamma \cup \{Z_0'\},\ \delta',\ q_0',\ Z_0',\ \{q_f\})$ □

Příklad 7.3

K zásobníkovému automatu z prvního příkladu této kapitoly (začíná na straně 110) sestrojíme ekvivalentní zásobníkový automat končící v koncovém stavu. Původní automat je
$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \{Z_0, a, b\},\ q_0,\ Z_0,\ \delta,\ \emptyset)$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
$\delta(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
$\delta(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
$\delta(q_0, c, Z_0) = (q_1, \varepsilon)$
$\delta(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

Přidáme nový stav q_0', ve kterém bude začínat výpočet, stav q_f, ve kterém bude končit výpočet, a nový zásobníkový symbol Z_0'. Automat bude následující:
$\mathcal{A}_F = (\{q_0, q_1, q_0', q_f\},\ \{a, b\},\ \{Z_0, a, b, Z_0'\},\ q_0',\ Z_0',\ \delta',\ \{q_f\})$, přechodová funkce δ' je

$\delta'(q_0', \varepsilon, Z_0') = (q_0, Z_0 Z_0')$

$\delta'(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta'(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta'(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
$\delta'(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
$\delta'(q_0, \varepsilon, Z_0') = (q_f, \varepsilon)$

$\delta'(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
$\delta'(q_0, c, Z_0) = (q_1, \varepsilon)$
$\delta'(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
$\delta'(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$
$\delta'(q_1, \varepsilon, Z_0') = (q_f, \varepsilon)$

Ukážeme si zpracování několika slov patřících nebo nepatřících do jazyka původního automatu:

$(q_0', abcba, Z_0') \vdash (q_0, abcba, Z_0 Z_0') \vdash (q_0, bcba, aZ_0 Z_0') \vdash (q_0, cba, baZ_0 Z_0') \vdash (q_1, ba, baZ_0 Z_0') \vdash$
$\vdash (q_1, a, a, Z_0 Z_0') \vdash (q_1, \varepsilon, Z_0 Z_0') \vdash (q_1, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$

$(q_0', c, Z_0') \vdash (q_0, c, Z_0 Z_0') \vdash (q_1, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$

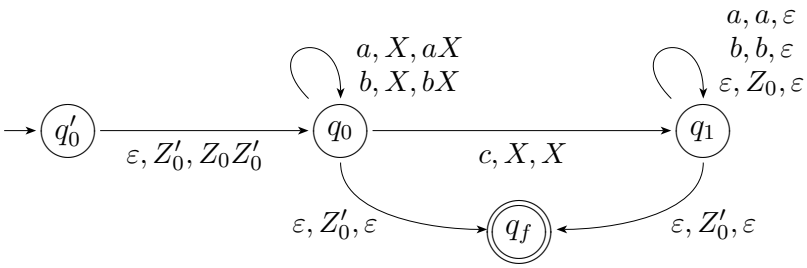
$(q_0', ab, Z_0') \vdash (q_0, ab, Z_0 Z_0') \vdash (q_0, b, aZ_0 Z_0') \vdash (q_0, \varepsilon, baZ_0 Z_0') \vdash$ nelze pokračovat, $ab \notin L(\mathcal{A}_F)$

$(q_0', ca, Z_0') \vdash (q_0, ca, Z_0 Z_0') \vdash (q_1, a, Z_0 Z_0') \vdash (q_1, a, Z_0') \vdash (q_f, a, \varepsilon) \vdash$ nelze pokračovat, $ca \notin L(\mathcal{A}_F)$

$(q_0', \varepsilon, Z_0') \vdash (q_0, \varepsilon, Z_0 Z_0') \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A}_F)$

Příklad 7.4

Reprezentaci zásobníkového automatu diagramem obvykle nepoužíváme – důvodem je horší přehlednost diagramu (v zásobníkových automatech musíme někam zapsat i práci se zásobníkem, také máme přechody bez zpracování slova na vstupu). Nicméně vytvoření diagramu je možné, například pro automat z předchozího příkladu by vypadal takto:



$X \in \{a, b, Z_0\}$

Ke každému přechodu píšeme čtený symbol ze vstupu, čtený symbol ze zásobníku a řetězec k uložení na zásobník.

Poznámka:

Všimněte si, že zásobníkový automat, který jsme v druhém příkladu vytvořili, ve skutečnosti končí jak v koncovém stavu, tak i s prázdným zásobníkem, tedy jsme zároveň dokázali vztah $\mathscr{L}(\mathcal{A}_\emptyset) \subseteq \mathscr{L}(\mathcal{A}_{F,\emptyset})$.

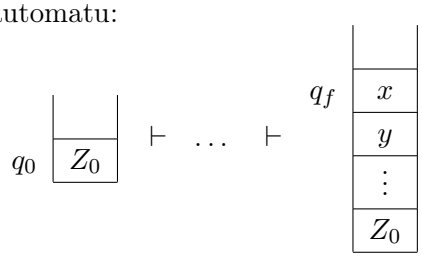
Postup konstrukce ($\mathscr{L}(\mathcal{A}_F) \subseteq \mathscr{L}(\mathcal{A}_\emptyset)$)**:** Původní zásobníkový automat končící v koncovém stavu označíme $\mathcal{A}_\emptyset = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, nový automat končící s prázdným zásobníkem označíme $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \emptyset)$.

K zásobníkovému automatu končícímu v koncovém stavu sestrojíme ekvivalentní zásobníkový automat končící s prázdným zásobníkem takto:

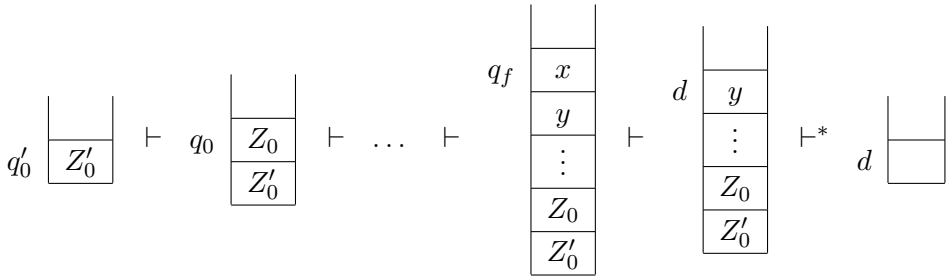
- stejně jako v předchozím postupu konstrukce využijeme toho, že můžeme mít vlastní symbol konce zásobníku,
- na konci výpočtu (až budeme v koncovém stavu původního automatu) přejdeme do nově přidaného stavu d („delete“), ve kterém následovně rekurzívně mažeme obsah zásobníku.

Uvnitř nového automatu budeme opět simulovat ten původní. Srovnáme průběh výpočtu v původním a novém zásobníkovém automatu:

Původní automat:



Nový automat:



První přechod mezi konfiguracemi má vypadat takto: $(q'_0, w, Z'_0) \vdash (q_0, w, Z_0 Z'_0)$, čímž se napojíme na výpočet původního automatu.

Na závěr výpočtu musíme provést následující: $(q_f, \varepsilon, Z\gamma Z'_0) \vdash (d, \varepsilon, \gamma Z'_0) \vdash^* (d, \varepsilon, \varepsilon)$, čímž se dostaneme do koncové konfigurace nového automatu. Stavy q'_0 a d jsou nově přidané, tedy musí platit $q'_0 \notin Q, \ d \notin Q$.

Přechodovou funkci původního automatu přejmeme a přidáme tyto přechody:

$\delta'(q'_0, \varepsilon, Z'_0) = (q_0, Z_0 Z'_0)$

$\delta'(q_f, \varepsilon, Z) = (d, \varepsilon)$ pro všechny původní koncové stavy $q_f \in F$ a zásobníkové symboly $Z \in \Gamma \cup \{Z'_0\}$.

Výsledný automat vypadá takto:

$\mathcal{A}_\emptyset = (Q \cup \{q'_0, d\}, \ \Sigma, \ \Gamma \cup \{Z'_0\}, \ \delta', \ q'_0, \ Z'_0, \ \emptyset)$ □

Poznámka:

Pokud bychom předchozí popsaný postup obohatili o jeden koncový stav: $F' = \{d\}$, získáme zásobníkový automat končící s prázdným zásobníkem a zároveň v koncovém stavu. Tím jsme si ukázali, že platí $\mathscr{L}(\mathcal{A}_F) \subseteq \mathscr{L}(\mathcal{A}_{\emptyset,F})$, čímž jsme dokončili postup konstrukce pro všechny vztahy zahrnuté v uvedené větě.

Příklad 7.5

Sestrojíme zásobníkový automat končící v koncovém stavu pro jazyk $L = \{a^n b^m \ ; \ 1 \leq m \leq n\}$. Symbolů b má být ve slově buď stejně nebo méně než symbolů a. Automat má pracovat takto:

- Ve stavu q_0 načítáme symboly a ze vstupu a ukládáme do zásobníku:
  $\delta(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
- Přecházíme do druhé části slova:
  $\delta(q_0, b, a) = (q_1, \varepsilon)$
- V druhé části slova postupně odstraňujeme symboly a ze zásobníku:
  $\delta(q_1, b, a) = (q_1, \varepsilon)$
- Pokud máme celý vstup přečtený, ukončíme výpočet, třebaže v zásobníku ještě mohou být symboly a:
  $\delta(q_1, \varepsilon, X) = (q_f, X),$ kde $X \in \{Z_0, a\}$

Výsledný automat:
$\mathcal{A} = (\{q_0, q_1, q_f\},\ \{a, b\},\ \{Z_0, a\},\ \delta,\ q_0,\ Z_0,\ \{q_f\})$, přechodová funkce:
$\delta(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, X) = (q_f, X),$ kde $X \in \{Z_0, a\}$

Ukázka zpracování několika slov patřících či nepatřících do jazyka $L(\mathcal{A})$:

$(q_0, aab, Z_0) \vdash (q_0, ab, aZ_0) \vdash (q_0, b, aaZ_0) \vdash (q_1, \varepsilon, aZ_0) \vdash (q_f, \varepsilon, aZ_0)$

$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0) \vdash (q_0, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_f, \varepsilon, Z_0)$

$(q_0, abb, Z_0) \vdash (q_0, bb, aZ_0) \vdash (q_1, b, Z_0) \vdash (q_f, b, Z_0) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A})$

$(q_0, \varepsilon, Z_0) \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A})$

Sestrojíme ekvivalentní zásobníkový automat končící s prázdným zásobníkem:
$\mathcal{A}_\emptyset = (\{q_0', q_0, q_1, q_f, d\},\ \{a, b\},\ \{Z_0', Z_0, a\},\ q_0',\ Z_0',\ \emptyset)$ s přechodovou funkcí:
$\delta'(q_0', \varepsilon, Z_0') = (q_0, Z_0 Z_0')$
$\delta'(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
$\delta'(q_0, b, a) = (q_1, \varepsilon)$
$\delta'(q_1, b, a) = (q_1, \varepsilon)$
$\delta'(q_1, \varepsilon, X) = (q_f, X),$ kde $X \in \{Z_0, a\}$
$\delta'(q_f, \varepsilon, X) = (d, \varepsilon),$ kde $X \in \{Z_0', Z_0, a\}$
$\delta'(d, \varepsilon, X) = (d, \varepsilon),$ kde $X \in \{Z_0', Z_0, a\}$

Ukázka zpracování stejných slov jako u automatu $\mathcal{A}$:

$(q_0', aab, Z_0') \vdash (q_0, aab, Z_0 Z_0') \vdash (q_0, ab, aZ_0 Z_0') \vdash (q_0, b, aaZ_0 Z_0') \vdash (q_1, \varepsilon, aZ_0 Z_0') \vdash (q_f, \varepsilon, aZ_0 Z_0') \vdash$
$\vdash (d, \varepsilon, Z_0 Z_0') \vdash (d, \varepsilon, Z_0') \vdash (d, \varepsilon, \varepsilon)$

$(q_0', aabb, Z_0') \vdash (q_0, aabb, Z_0 Z_0') \vdash (q_0, abb, aZ_0 Z_0') \vdash (q_0, bb, aaZ_0 Z_0') \vdash (q_1, b, aZ_0 Z_0') \vdash$
$\vdash (q_1, \varepsilon, Z_0 Z_0') \vdash (q_f, \varepsilon, Z_0 Z_0') \vdash (d, \varepsilon, Z_0') \vdash (d, \varepsilon, \varepsilon)$

$(q_0', abb, Z_0') \vdash (q_0, abb, Z_0 Z_0') \vdash (q_0, bb, aZ_0 Z_0') \vdash (q_1, b, Z_0 Z_0') \vdash (q_f, b, Z_0 Z_0') \vdash (d, b, Z_0') \vdash$
$\vdash (d, b, \varepsilon) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A}_\emptyset)$

$(q_0', \varepsilon, Z_0') \vdash (q_0, \varepsilon, Z_0 Z_0') \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A}_\emptyset)$

Poznámka:

Kdybychom v předchozím případě chtěli vytvořit zásobníkový automat rozpoznávající jak s prázdným zásobníkem, tak i v koncovém stavu, jen bychom mírně pozměnili zápis:
$\mathcal{A}_\emptyset = (\{q_0', q_0, q_1, q_f, d\},\ \{a, b\},\ \{Z_0', Z_0, a\},\ q_0',\ Z_0',\ \{d\})$. Přechodová funkce by byla stejná.

7.3 Vztah zásobníkových automatů a bezkontextových jazyků

Nadále budeme počítat s tím, že zásobníkové automaty všech tří základních typů jsou navzájem ekvivalentní, a tedy generují stejné třídy jazyků. Proto v důkazech budeme volně tyto typy zaměňovat.

Věta 7.2 (Bezkontextová gramatika $\longrightarrow$ zásobníkový automat)

Ke každé bezkontextové gramatice G lze vytvořit zásobníkový automat $\mathcal{A}$ tak, že $L(G) = L(A)$:

$$\mathscr{L}(CF) \subseteq \mathscr{L}(ZA) \tag{7.6}$$

Postup konstrukce: Je dána bezkontextová gramatika $G = (N, T, P, S)$. Chceme sestrojit k ní ekvivalentní zásobníkový automat $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ končící s prázdným zásobníkem.

Princip je následující: potřebujeme rozpoznávat právě ta slova, která jsou generována gramatikou. Proto vytvářený automat bude na svém zásobníku provádět simulaci derivace pro slovo, které dostane na svůj vstup. Pokud zjistí, že slovo lze v původní gramatice derivovat, pak je přijme.

Postup bude odlišný od postupu pro regulární gramatiky a konečné automaty. Máme k dispozici zásobník a ten budeme využívat. Naopak stavy pro nás nebudou důležité, vystačíme si s jediným stavem, který můžeme označit q. Abeceda je $\Sigma = T$. Přechodová funkce bude mít dvě části:

- První část odpovídá pravidlům původní gramatiky:

Pravidlo gramatiky	*Předpis v δ-funkci*
$A \to \alpha$	$\delta(q, \varepsilon, A) \ni (q, \alpha)$

  Jak vidíme, vstupu si nevšímáme (ε), ze zásobníku vyjmeme neterminál (zde A) a nahradíme řetězcem z pravé strany pravidla pro tento neterminál (α). Vše se odehrává pouze na zásobníku, nemění se stav a nepohybujeme se na vstupu.

- Druhá část gramatiky určuje, že pokud je na vrcholu zásobníku terminál, pak zjistíme, jestli je tentýž terminál na vstupu – když ano, posuneme se jak na vstupu, tak i na zásobníku:

  $\delta(q, a, a) = (q, \varepsilon)$ pro všechny symboly $a \in T$

  Tato část je také důležitá, protože v zásobníku máme samozřejmě kromě neterminálů i terminály (dostávají se tam se zápisem pravých stran pravidel, α). Zároveň kontrolujeme, jestli simulace derivace podle gramatiky probíhá správně (tedy zda simulujeme odvozování toho slova, které je na vstupu, a ne jiného).

Počátečním stavem bude stav q, počátečním zásobníkovým symbolem bude S (startovací symbol gramatiky), protože pokud máme na zásobníku provádět simulaci derivace, musíme tam na začátku mít počáteční větnou formu, což je právě jednoprvkový řetězec obsahující startovací symbol.

Protože cokoliv, co se nachází v pravidlech gramatiky, se může objevit v zásobníku, bude zásobníková abeceda obsahovat všechny neterminály i terminály původní gramatiky: $\Gamma = N \cup T$.
□

Příklad 7.6

Sestrojíme zásobníkový automat ekvivalentní gramatice $G = (\{S, A\},\ \{a, b, c\},\ P, S)$, kde P obsahuje tato pravidla:
$S \to aSbb \mid cAa$
$A \to cAa \mid \varepsilon$

Vytvoříme zásobníkový automat $\mathcal{A} = (\{q\},\ \{a, b, c\},\ \{S, A, a, b, c\},\ \delta,\ q,\ S,\ \emptyset)$ s přechodovou funkcí určenou takto:

$\delta(q, \varepsilon, S) = \{(q, aSbb), (q, cAa)\}$ podle pravidel $S \to aSbb \mid cAa$
$\delta(q, \varepsilon, A) = \{(q, cAa), (q, \varepsilon)\}$ podle pravidel $A \to cAa \mid \varepsilon$
$\delta(q, a, a) = \{(q, \varepsilon)\}$ protože $a \in T$
$\delta(q, b, b) = \{(q, \varepsilon)\}$ protože $b \in T$
$\delta(q, c, c) = \{(q, \varepsilon)\}$ protože $c \in T$

Jazyk generovaný gramatikou G a rozpoznávaný automatem $\mathcal{A}$ je
$L(G) = L(\mathcal{A}) = \left\{a^n c^k a^k b^{2n}\ ;\ n \geq 0,\ k \geq 1\right\}$.

Ukážeme si vždy derivaci některého slova v gramatice G a ekvivalentní zpracování slova v zásobníkovém automatu $\mathcal{A}$:

$S \Rightarrow cAa \Rightarrow ccAaa \Rightarrow ccaa$

$(q, ccaa, S) \vdash (q, ccaa, cAa) \vdash (q, caa, Aa) \vdash (q, caa, cAaa) \vdash (q, aa, Aaa) \vdash (q, aa, aa) \vdash (q, a, a) \vdash$
$\vdash (q, \varepsilon, \varepsilon)$
$\Longrightarrow$ $ccaa \in L(G)$ a zároveň $caa \in L(\mathcal{A})$

$S \Rightarrow aSbb \Rightarrow aaSbbbb \Rightarrow aacAabbbb \Rightarrow aacabbbb$

$(q, aacabbbb, S) \vdash (q, aacabbbb, aSbb) \vdash (q, acabbbb, Sbb) \vdash (q, acabbbb, aSbbbb) \vdash (q, cabbbb, Sbbbb) \vdash$
$\vdash (q, cabbbb, cAabbbb) \vdash (q, abbbb, Aabbbb) \vdash (q, abbbb, abbbb) \vdash (q, bbbb, bbbb) \vdash (q, bbb, bbb) \vdash$
$\vdash (q, bb, bb) \vdash (q, b, b) \vdash (q, \varepsilon, \varepsilon)$
$\Longrightarrow$ $aacabbbb \in L(G)$ a zároveň $aacabbbb \in L(\mathcal{A})$

Na vstup automatu dáme některá slova nepatřící do jazyka $L(G)$:

$(q, abb, S) \vdash (q, abb, aSbb) \vdash (q, bb, Sbb) \vdash (q, bb, cAabb) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A})$

$(q, \varepsilon, S) \vdash (q, \varepsilon, cAa) \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A})$

$(q, acab, S) \vdash (q, acab, aSbb) \vdash (q, cab, Sbb) \vdash (q, cab, cAabb) \vdash (q, ab, Aabb) \vdash (q, ab, abb) \vdash$
$\vdash (q, b, bb) \vdash (q, \varepsilon, b) \vdash$ nelze pokračovat, $acab \notin L(\mathcal{A})$

Literatura

[1] Chytil, M.: *Automaty a gramatiky*. Praha: SNTL, 1984.

[2] Meduna, Alexander. *Automata and languages: theory and applications*. London: Springer, 2000, xv, 916 s. ISBN 18-523-3074-0
Dostupné na Google Books: http://books.google.cz, jako klíčová slova zadejte celý název knihy [cit. 2008-7-1]

[3] Melichar, Bořivoj. *Jazyky a překlady*. Vyd. 1. Praha: ČVUT, 1996. ISBN 80-010-1511-4

[4] Paleta, Petr. *Co programátory ve škole neučí: aneb Softwarové inženýrství v reálné praxi*. Vyd. 1. Brno: Computer Press, 2003, 337 s. ISBN 80-251-0073-1

[5] Robic, Florent. A real Turing machine. *The Alan Turing Year* [online]. 2012 [cit. 2015-01-27]. Dostupné z: http://www.turing2012.fr/?p=530&lang=en

Přílohy

Příloha A

Řecká abeceda

Protože se v teoretické informatice používá hodně řeckých písmen (písmenek totiž „není nikdy dost“), je v této příloze celá řecká abeceda včetně informace o použití nejběžnějších symbolů.

Název	*Malé*	*Velké*	*Komentář*
alfa	α	A	malá α obvykle označuje řetězec (jakýchkoliv) znaků
beta	β	B	malá β obvykle označuje řetězec (jakýchkoliv) znaků, případně máme β záření
gamma	γ	Γ	malá β obvykle označuje řetězec (jakýchkoliv) znaků, velká Γ se používá jako zásobníková abeceda u zásobníkových automatů, svůj význam má i ve fyzice, máme také γ vlny (při měření EEG)
delta	δ	Δ	malá δ se používá při zápisu přechodové funkce automatů (funkce určující, jak se má automat v určitých konfiguracích chovat), velká Δ bývá používána jako pomocná abeceda; obecně symbol Δ určuje *rozdíl* či interval; oba symboly mají význam v matematice, fyzice, astronomii atd.
epsilon	ε	E	malé ε označuje prázdné slovo (slovo o délce 0); obecně v matematice a fyzice je to označení pro „velmi malé číslo“, používá se například při výpočtech pravděpodobnosti či odhadu chyby
zéta	ζ	Z	ζ je Riemannova funkce v matematice
éta	η	H	v případě nutnosti můžeme η použít pro označení řetězce (jakýchkoliv) znaků, určitý význam má v matematice, fyzice, astronomii
théta	θ, ϑ	Θ	ϑ je úhel impendance, se symbolem θ se setkáváme ve fonetice nebo v matematice (označení úhlu)
jóta	ι	I	pozor, ι připomíná malé latinské i, ale není nad ní tečka
kappa	κ	K	se symbolem κ se setkáváme v diferenciální geometrii (křivost křivky) a v některých dalších vědách (vč. fyziky)

lambda	λ	Λ	malé λ bývá v některých zdrojích používáno místo symbolu ε, tedy může značit prázdné slovo; v matematice se tak značí vlastní číslo nebo Lebesqueova míra, ve fyzice je to vlnová délka nebo tepelná vodivost, atd.
mí	μ	M	malé μ se v soustavě jednotek SI používá pro předponu mikro (například μm je mikrometr) – tisícinu, je to také aritmetický průměr v matematice, má svůj význam i v dalších vědách (zejména fyzice)
ný	ν	N	symbol ν značí v deskriptivní geometrii nárysnu (ν je nárysna, μ je půdorysna, atd.), další významy najdeme ve fyzice
ksí	ξ	Ξ	ve fyzice je ξ součinitel valivého odporu, v informatice se vzhledem ke složitosti zápisu používá jen minimálně (v důkazech, když potřebujeme označit řetězec jakýchkoliv znaků)
omikron	o	O	nepoužívá se, hrozí záměna se stejným latinským písmenem
pí	π	Π	malé π značí kromě jiného Ludolfovo číslo ($3.14159\ldots$), velké Π znamená násobení (podobně jako Σ je sčítání)
ró	ϱ, ρ	P	malé ϱ značí ve fyzice hustotu nebo měrný elektrický odpor, ve statistice znamená korelační koeficient, v matematice spektrální poloměr matice
sigma	σ	Σ	velké Σ pro nás znamená hlavní (vstupní) abecedu automatu (prvky z této abecedy dokáže automat zpracovávat), v matematice znamená sumu (součet), svůj význam má tento symbol i v matematice, statistice, fyzice, chemii apod.
tau	τ	T	používáme spíše ve fyzice
ypsilon	υ	Y	používáme spíše ve fyzice
fí	φ, ϕ	Φ	malé φ znamená zlatý řez (v matematice, umění, architektuře), v teorii čísel Eulerovu funkci $\varphi(n)$, značí formuli v predikátové logice, elektrický potenciál ve fyzice, úhel, apod., případně jednoduše řetězec (jakýchkoliv) znaků
chí	χ	X	používáme spíše ve statistice
psí	ψ	Ψ	ve fyzice to je jedna z částic, u nás řetězec (jakýchkoliv) znaků
omega	ω	Ω	malé ω využíváme velmi často jako řetězec (jakýchkoliv) prvků, případně řetězec terminálních prvků, velké Ω znamená ohm (jednotku elektrického odporu) ve fyzice, obecně ve fyzice je to poměrně častý symbol

Příloha B

Ukázky využití regulárních výrazů

V této příloze se podíváme podrobněji na možnosti využití regulárních výrazů v operačních systémech Windows a Linux. Náplň této kapitoly nebude zřejmě nikdy úplná a definitivní, protože podporu regulárních výrazů najdeme nejen u mnoha nástrojů v operačních systémech, ale také coby moduly v programovacích jazycích. Navazujeme na základ z kapitoly 2.1 na straně 9.

B.1 Windows

Přímo ve Windows se s jednoduchými regulárními výrazy (pokud to tak lze vůbec nazvat) setkáváme v mnoha příkazech na Příkazovém řádku. Jde o to jak co nejjednodušeji reprezentovat množinu řetězců, které mají určité společné vlastnosti (například množinu všech názvů souborů s určitou příponou). Máme však možnost využívat i plnohodnotné regulární výrazy, a to při prohledávání textových sekvencí.

B.1.1 Příkaz `dir` a další příkazy využívající zjednodušené výrazy

Příkaz `dir` je určen k výpisu obsahu adresáře (složky). Syntaxe je následující (zde uvedeme jen silně zkrácenou verzi, pro ilustraci):

```
dir [Jednotka:][Cesta][Název_souboru] [...] parametry
```

To, co je v hranatých závorkách, je nepovinné, trojtečka je zde ve funkci výpustky (tj. zde by syntaxe byla košatější než vypisujeme).

V rámci zjednodušených regulárních výrazů (zde pro určení adresáře, jehož obsah má být vypsán, nebo souboru, k němuž chceme získat informace) můžeme používat následující zástupné symboly:

Prvek	Význam
?	Libovolný znak (právě jeden)
*	Nula nebo více výskytů jakýchkoliv znaků, řetězec (jakýkoliv počet jakýchkoliv znaků)

Tabulka B.1: Zástupné symboly v zjednodušených výrazech

Příklad B.1

Využití zjednodušených výrazů si ukážeme na několika příkladech:

- `dir *.txt`
  hledá všechny soubory s příponou TXT, které se nacházejí v pracovním adresáři
- `dir dopis*.*`
  vypíše všechny soubory odpovídající zadané masce (tj. začínající řetězcem „dopis“, zbytek názvu včetně přípony může být jakýkoliv); masce odpovídá například `dopis.txt`, `dopisy.doc`, `dopisJeziskovi.odt`,...
- `dir *.exe /s /b`
  vypíše všechny soubory s příponou EXE, parametry za řetězcem způsobí rekurzívní prohledávání všech podadresářů
- `dir c:\win*`
  hledá přímo na disku `C:` vše, co začíná řetězcem `win`
- `dir nt????*.txt /s /b`
  vypíše soubory s příponou TXT začínající řetězcem `nt`, následují jakékoliv 4 znaky, prohledává rekurzívně
- `dir *.docx /a:A /o:D`
  hledá soubory s příponou DOCX, které mají nastaven atribut „k archivaci“ (tj. od posledního zálohování byly pravděpodobně změněny), seznam bude seřazen podle data poslední aktualizace (od nejnovějšího)
- `DIR *.sys /s /b > f:\ovladace.txt`
  pokud příkaz spustíme v kořenovém adresáři disku `C:`, získáme v souboru `f:\ovladace.txt` seznam všech souborů s příponou SYS nacházejících se na disku `C:` (kdekoliv), většina z toho budou soubory ovladačů různých zařízení nebo souborových systémů

Zjednodušené výrazy můžeme použít prakticky ve všech příkazech, které jako vstup berou parametry ve smyslu názvů souborů, adresářů (složek), nebo řetězce, které je nutno vyhledat.

Příklad B.2

Pár ukázek příkazů používajících jednoduché výrazy:

- `attrib *`
  pro každý soubor a adresář v pracovním adresáři vypíše jeho nastavené atributy (k archivaci, skrytý, systémový, atd.)
- `copy *.docx f:\zaloha`
  zkopíruje všechny soubory s příponou DOCX do zálohy (pro pokročilejší zálohování používáme spíše příkazy `xcopy` a `robocopy`)
- `cacls *.docx /t /e /g patrik:w`
  ke všem souborům s příponou DOCX v pracovním adresáři i jeho podadresářích (rekurze) je uživateli *patrik* přiděleno právo zápisu

- `del /s *.tmp`
  vymažeme rekurzívně všechny soubory s příponou TMP, to budou pravděpodobně dočasné soubory
- `route print -4 10.*`
  zobrazí směrovací tabulku pro adresy IPv4, vypíše pouze směrovací informace k cílům, jejichž IP adresa začíná číslem 10 (tj. zřejmě adresy přidělené v místní síti, pokud je používán adresní rozsah „A“)
- `find /i "objedn"dopis*.txt`
  projde všechny soubory odpovídající masce `dopis*.txt` a vypíše z nich všechny řádky, na kterých se nachází řetězec `objedn`

Další informace:

Pro většinu příkazů existuje možnost získat nápovědu pomocí jednoduchého parametru s otazníkem, například pro příkaz `cacls` stačí napsat

```
cacls /?
```

B.1.2 Plnohodnotné regulární výrazy při vyhledávání

Ve všech novějších verzích Windows máme k dispozici příkaz `findstr` pro vyhledávání pomocí regulárních výrazů. Inspirací při určování syntaxe příkazu byl příkaz `grep` z unixových systémů, což pro uživatele znamená určitou výhodu (když umím jeden, snadno se naučím druhý).

Syntaxe příkazu je následující (opět silně zkrácená, podrobnější najdeme v nápovědě příkazu):

```
findstr [/b] [/e] [/i] [/n] [/p] [/g:soubor] [/f:soubor] [/c:řetězec] [řetězce]
    [Název_souboru [...]]
```

Nejdůležitější parametry mají tento význam:

- `/b` hledá shodu na začátku řádku
- `/e` hledá shodu na konci řádku
- `/i` nerozlišuje malá a velká písmena
- `/n` zobrazí také čísla řádků, na kterých byla nalezena shoda
- `/p` vynechá soubory obsahující netisknutelné znaky
- `/g:soubor` z tohoto souboru čte řetězce s regulárními výrazy
- `/f:soubor` z tohoto souboru čte názvy souborů, ve kterých má vyhledávat
- `/c:řetězec` řetězec bere jako jediný regulární výraz
- `řetězce` regulární výrazy pro vyhledávání oddělené mezerami
- `Název_souboru` soubor, ve kterém má vyhledávat

Především zadáváme soubory nebo textové sekvence, které chceme prohledat, a regulární výraz určující, co vlastně hledáme. Z charakteristiky parametrů vyplývá, že regulární výraz můžeme buď napsat přímo jako parametr příkazu, nebo můžeme do souboru napsat jeden či více regulárních výrazů a pak název tohoto souboru zadat za parametrem `/g`.

Podobně název souboru (nebo více souborů pomocí zjednodušeného výrazu) k prohledávání zadáváme buď přímo (jako poslední parametr), nebo názvy souborů uložíme do jednoho souboru, jehož název předáme za parametrem `/f:`, případně můžeme text poslat přes „rouru“ jako výstup jiného příkazu (příkaz funguje i jako filtr).

Prvek	Význam
`.`	Libovolný znak
`*`	Nula nebo více výskytů předcházejícího znaku nebo třídy
`^`	Začátek řádku
`$`	Konec řádku
`[třída]`	Jakýkoli (jeden) znak z množiny v závorkách
`[^třída]`	Jakýkoli znak mimo prvky množiny
`[x-y]`	Jakékoli znaky v daném rozsahu (jeden)
`\<xyz`	Začátek slova
`xyz\>`	Konec slova

Tabulka B.2: Prvky použitelné v regulárních výrazech

V regulárním výrazu, se kterým umí pracovat tento příkaz, můžeme kromě běžných zástupných znaků (hvězdička, otazník) zadávat i další prvky, například pro symbol na daném místě v hledaném řetězci můžeme určit množinu „povolených“ znaků. Seznam používaných prvků najdeme v tabulce B.2.

Příklad B.3

Pár ukázek použití regulárního výrazu v parametru příkazu:

- `findstr "for med:=1"hlavni.pas`
  hledá řetězce `for` a `med:=1` v souboru `hlavni.pas`
- `findstr /c:"for med:=1"hlavni.pas`
  hledá řetězec `for med:=1` v souboru `hlavni.pas`
- `findstr "^[0-9][0-9]*S"abc.log`
  v zadaném souboru hledá řádky, na jejichž *začátku* (stříška) je číslo (s nejméně jednou číslicí) následované písmenem „S“ (velkým), hvězdička se vztahuje k prvkům množiny `[0-9]` (tj. jakýkoliv počet číslic)
- `findstr /i "[a-z0-9]@[a-z0-9]"konverzace.txt`
  hledá řetězce obsahující symbol zavináče, který je z obou stran obklopen alespoň jedním písmenem nebo číslem, většina nalezených řetězců budou pravděpodobně e-mailové adresy

Další informace:

Další informace o příkazu získáme jednoduše takto: `findstr /?`

B.2 Linux

Linux je unixový systém (také se píše jako „UNIX-like“), tedy víceméně vše, co je zde napsáno, platí pro jakýkoliv jiný unixový systém (MacOS X je taky unixový systém). Měli bychom mít na paměti, že v unixových systémech včetně Linuxu se rozlišují malá a velká písmena (říkáme, že systém je case-sensitive), což je opačné chování, než na jaké je zvyklý uživatel Windows.

V Linuxu můžeme regulární výrazy používat kromě jiného takto:

- zjednodušené výrazy, ale oproti použití ve Windows s dalšími možnostmi (včetně `[xyz]`, `[^xyz]` apod.) ve všech příkazech pracujících s adresáři a soubory
- `grep` – vyhledávání řetězců podle zadaného regulárního výrazu
- `sed` – vyhledávání a editace (různé akce) řetězců nejen podle zadaného regulárního výrazu
- `awk` – ještě širší možnosti než sed, programovací jazyk (podobný C) na zpracování textových souborů

A pak samozřejmě v nejrůznějších programovacích jazycích, například v Perlu.

B.2.1 Příkaz `grep`

Jak bylo dřív uvedeno, příkaz `grep` byl inspirací pro obdobný příkaz v systému Windows. Jeho určením je prohledávání textových vstupů (souborů nebo výstupů jiných programů). Syntaxe (opět silně zkrácená) je následující:

```
grep [přepínače] reg_výraz [soubor ...]
```

Význam nejdůležitějších přepínačů:

- `-i` nerozlišuje malá a velká písmena
- `-l` pouze vypíše názvy souborů, ve kterých nalezl shodu
- `-n` vypíše také číslo řádku
- `-r` rekurzívně zpracovává i podadresáře
- `-o` vypíše jen nalezený řetězec, ne celý řádek
- `-c` v souborech pouze spočítá výskyty nalezeného řetězce

Prvky, které můžeme použít v parametru příkazu `grep`, najdeme v tabulce B.3. Je jich víc než v případě obdobného příkazu ve Windows, máme možnost stanovit i počet opakování nebo použít logické „nebo“.

Příklad B.4

Pár ukázek použití:

- `grep -i "vypis"*.txt`
  hledá v souborech s příponou `.txt` slovo `vypis` bez rozlišování malých a velkých písmen
- `grep -cr "#include.*\.[hc]"*.c`
  u každého souboru s příponou `.c` vypíše počet vkládaných souborů s příponou `.h` nebo `.c`, a protože tečka je významový znak (určuje jeden jakýkoliv symbol), musíme před ni dát zpětné lomítko (escape sekvence), aby byla brána jako součást řetězce (tj. zrušit její speciální význam)

Prvek	Význam
`.`	Libovolný znak
`?`	Nula nebo jeden výskyt předcházejícího řetězce
`*`	Nula nebo více výskytů předcházejícího řetězce
`+`	Jeden nebo více výskytů předcházejícího řetězce
`{m}`	`m` opakování předcházejícího řetězce
`{m,n}`	`m` až `n` opakování předcházejícího řetězce
`{m,}`	`m` nebo více opakování předcházejícího řetězce
`^`	Začátek řádku
`$`	Konec řádku
`[třída]`	Jakýkoli (jeden) znak z množiny v závorkách
`[^třída]`	Jakýkoli znak mimo prvky množiny
`[x-y]`	Jakékoli znaky v daném rozsahu (jeden)
`r1 | r2`	logické „nebo“ – buď řetězec `r1`, nebo řetězec `r2`

Tabulka B.3: Prvky regulárních výrazů v parametru příkazu `grep`

- `grep -cr "Dear \(Mr.|Ms.|Miss\)"*.txt`
  spočítá, kolikrát se v zadaných souborech objevilo oslovení v angličtině
- `grep "[0-9]\{6\}/[0-9]\{3,4\}"soubor.txt`
  hledáme rodná čísla ve tvaru 123456/1234 – nejdřív 6 číslic, pak lomítko a tři nebo čtyři číslice
- `grep "\([0-9]\{1,3\}\)\{4\}"soubor.log`
  hledáme IP adresy ve tvaru 123.123.123.123 – čtyři skupiny číslic, v každé skupině 1 až 3 číslice

Všimněte si uzávorkování v posledních třech odrážkách příkladu – kdybychom napsali jen závorku bez opačného lomítka před ní, bylo by to chápáno jako znak, který má být v souboru také vyhledán. My však potřebujeme, aby tyto závorky měly „speciální význam“, například aby sloužily jako uzávorkování ve výrazu.

Opačné lomítko tedy slouží jako obousměrný přepínač, znak může být buď součástí hledaného řetězce nebo se může jednat o významový prvek.

Další informace:

Informace a příklady k příkazu `grep` a jeho variantám (`egrep`, `fgrep`) najdeme takto:

- manuálové stránky příkazů, například `man grep`
- na internetu (třeba na `google.com`), kde zadáme k vyhledání man grep apod.
- na internetu najdeme hodně příkladů, například grep příklady

B.2.2 Program `sed`

Program `sed` (stream editor) slouží ke zpracovávání podřetězců v textovém vstupu. Tento vstup může být opět soubor, několik souborů nebo vstupem může být výstup jiného programu (přes rouru, pak `sed` funguje jako filtr). Další důležitou součástí je tzv. *skript*, který určuje, s čím má program pracovat (tj. regulární výraz nebo jiné určení řádku/-ků) a co s tím má provést (příkaz, například zaměnit jeden řetězec za jiný, smazat podřetězec, apod.). Zkrácená syntaxe příkazu je následující:

```
sed [přepínače]... [script] [vstupní_soubor]...
```

Nejběžnější přepínače určují skript, který se má provést:

- `-e řetězec` skript k provedení (ne soubor!)
- `-f soubor` soubor se skriptem k provedení

Skript (ať už zadaný přímo v příkazu (za parametrem `-e řetězec`) nebo v souboru, jehož název napíšeme (za parametrem `-f soubor`) se skládá ze dvou částí – má formu `ap`, kde

- `a` je adresa ve zpracovávaném souboru
- `p` je příkaz, který se na místě určeném adresou má provést

Adresa (tj. určení místa ve vstupu, které má být zpracováno) může být následující:

- (bez adresy) příkaz se použije na všechny řádky
- `číslo` číslo řádku, který se má zpracovat
- `číslo~krok` všechny řádky počínaje od řádku s daným číslem, s násobkem daným krokem (tj. `číslo + i*krok`), například `1~2` budou všechny liché řádky
- `$` poslední řádek vstupu
- `/regulární_výraz/` řádky odpovídající regulárnímu výrazu, lomítka jsou nutná, uvnitř používáme vše, co u příkazu `grep`, ale před některé významové znaky dáváme zpětné lomítko jako escape sekvenci, abychom „přehodili“ význam (například `\+`, `\?`, `\{...}`, `\|`)
  Pokud následuje `I`, nerozlišují se malá a velká písmena
- `adresa1,adresa2` všechny řádky v rozmezí adres (adresy opět mohou být cokoliv z předchozího, kromě první odrážky)
- `adresa,+n` všech n řádků od zadané adresy

Výsledkem interpretace adresy je místo v textovém vstupu, které má být dále zpracováno. Může to být řetězec na některém řádku, ale také více řádků odpovídajících zadanému regulárnímu výrazu.

Za adresou ve skriptu následuje příkaz určující, co se s nalezeným řádkem má provést:

- `d` vymaž nalezený vzor
- `p` vypiš (vytiskni) nalezený vzor na výstup
- `s/co/čím/přepínače` (lomítka jsou součástí výrazu) nahraď `co` `čím`, a to způsobem určeným přepínači:
  - `g` nahraď všechny výskyty
  - `číslo` nahraď jen výskyt s pořadím `číslo`
  - `i` nerozlišuj malá a velká písmena

Příklad B.5

Podíváme se na pár ukázek použití příkazu `sed`:

- `sed -e '1,5d' soubor.txt`
  odstraní ze zadaného souboru první až pátý řádek (adresa je„ `1,5`“, tedy první až pátý řádek, příkaz je „`d`“, tedy delete)
- `sed -e 's/<[^>]*>//g' *.html`
  - adresa není uvedena, takže se zpracuje celý vstup
  - příkaz je „`s`“, tedy budeme nahrazovat
  - mezi prvními dvěma lomítky máme regulární výraz určující co má být nahrazeno (tj. vše, co je uzavřeno do ostrých závorek včetně těchto závorek, v našem případě se bude jednat o HTML tagy)
  - mezi dalším párem lomítek je „prázdno“ (prázdný řetězec), tedy nalezené řetězce budou prostě smazány
  - přepínač `g` určuje, že všechny nalezené řetězce (tagy) mají být nahrazeny (zde defacto smazány)

  v reálu to znamená, že ze všech HTML souborů v daném adresáři budou odstraněny všechny tagy `<...>` (vnitřní část zajišťuje, že pokud je na řádku více tagů, bude text mezi nimi zachován)
- `cat soubor.txt | sed -e 's/\&/\&/g' | sed -e 's/</\</g' | sed -e 's/>/\>/g' > soubor.html`
  tento poněkud komplexní příkaz v sobě kombinuje několik volání příkazu `sed`, postupně proběhnou tři transformace;
  - v prvním průchodu se v souboru zamění všechny výskyty symbolu „`&`“ řetězcem „`&`“,
  - v druhém průchodu se následně všechny výskyty symbolu „`<`“nahradí řetězcem „`<`“,
  - v třetím průchodu se všechny výskyty symbolu „`>`“nahradí řetězcem „`>`“.

  Výsledek je uložen do souboru `soubor.html`, který se po dalších úpravách (dodání některých tagů, záhlaví apod.) stane html stránkou; jde tedy o usnadnění převodu textového souboru do HTML formy, což pak nemusíme provádět ručně

Příkaz `sed` má ještě další možnosti, které usnadňují provádění hromadných transformací, ale to by bylo nad rámec rozsahu a určení této přílohy.

Další informace:

Informace a příklady k příkazu `sed` najdeme takto:

- manuálová stránka příkazu `man sed`
- na internetu (třeba na `google.com`), kde zadáme k vyhledání man sed
- na internetu najdeme hodně příkladů, například sed command examples

B.2.3 Další příkazy

Co se týče zjednodušených výrazů (v širším významu než v případě Windows), opět je můžeme používat v parametrech mnoha různých příkazů. Dokonce množina příkazů, ve kterých se dají používat, je mnohem širší než ve Windows, protože unixové systémy jsou tradičně vybaveny drobnými programy prakticky „na všechno“.

Příklad B.6

Pár motivačních příkladů:

- `ls -l *.html`
  zajímají nás všechny soubory s příponou HTML a podrobné informace o nich
- `find -name '*.txt' -print`
  vyhledá *rekurzívně* všechny soubory s příponou TXT a jejich seznam vypíše
- `du -h ` `ls *.txt` ``
  vypíše místo na disku zabrané jednotlivými soubory s příponou DOC nacházejícími se v pracovním adresáři (zpětné jednoduché uvozovky jsou zde nutné, protože „vnitřní příkaz“ je nutno interpretovat dřív než bude dosazen jako parametr příkazu `du`)
- `/sbin/modeprobe -l *conntrack*`
  chceme vypsat seznam všech modulů jádra (i těch, které zrovna nejsou v jádře načteny), jejichž název obsahuje řetězec `conntrack` (v tomto případě se jedná o moduly související s firewallem a jeho funkcí SPI – Stateful Packet Inspection)

Podobná syntaxe se používá i v mnohých konfiguračních souborech, např. v souboru `syslog.conf` (pokud pro logování používáme *syslog*).

Když se na problematiku podíváme z opačného konce, můžeme najít program, který se regulárními výrazy a souvisejícími transformacemi zabývá dokonce ještě sofistikovaněji než program `sed` – program `awk` umožňuje provádět v textových sekvencích transformace prakticky jakéhokoliv charakteru. Jedná se o komplexní příkaz, jehož transformační předpisy hodně připomínají programování v jazyce C. Je natolik složitý, že programátoři v poslední době místo něj používají spíše jednodušší Perl.

Další informace:

Další informace najdeme (jak jinak) opět v manuálových stránkách a na internetu:

- manuálové stránky příkazů: `man awk`, ...
- na internetu (třeba na `google.com`), kde zadáme k vyhledání man awk apod.
- na internetu najdeme hodně příkladů, například awk command examples
- http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_04.html