TEORETICKÁ INFORMATIKA 1. ČÁST

Konečné automaty a regulární jazyky

Václav Vais

Obsah

Předmluva

Tento materiál se snaží být v jistém smyslu uceleným (ale určitě ne vyčerpávajícím) textem o *konečných automatech a regulárních jazycích*, psaným sice „z pozic“ teoretické informatiky, ale pro studenty inženýrských, zejména informatických oborů. Text pokrývá dva z pěti tématických okruhů, kterým se věnuje předmět KIV/TI ve druhém ročníku bakalářského studia studijního programu Inženýrská informatika.

Text není záznamem přednášek, rozsahem i podrobnostmi přesahuje objem, který obvykle bývá k daném tématu odpřednášen (a který také bývá zkoušen). Smyslem tohoto materiálu je mimo jiné i ukázat přesah do dalších předmětů studia a naznačit použití teoretických modelů k řešení reálných problémů.

Při tvorbě tohoto textu jsem se snažil zejména o následující:

- v teoretických partiích neustoupit z matematické přesnosti, ale prezentované definice modelů a jejich vlastností doplnit podrobným slovním vysvětlením (tam, kde jsem to považoval za vhodné, jsem někdy přistoupil i k méně přesným, nicméně názornějším formulacím; takové formulace jsou obvykle uvedeny v uvozovkách),
- pro oslabení falešného pocitu „vždyť ta křivá písmenka k ničemu praktickému nejsou“ (a věřím, že tedy i ke zvýšení motivace studentů ke studiu předmětu KIV/TI) jsem uvedl několik reálných příkladů z aplikačních oblastí, kde se teoreticky probírané modely uplatňují,
- snažil jsem se ukázat principy a postupy, s jejichž pomocí lze konečné automaty implementovat, a to jak softwarově, tak i hardwarově; tyto kapitoly (stejně tak jako některé příklady z aplikačních oblastí) lze chápat jako jakési „krátké exkurze“ do předmětů z vyšších ročníků studia; nicméně považoval jsem za vhodné, aby ti hloubavější nebo lépe motivovaní studenti měli možnost vidět už teď, že není složité abstraktní matematické modely dovést standardními postupy až k programové nebo obvodové realizaci.

U zkoušky bude vyžadována znalost v rozsahu přednášek, tj. v rozsahu zveřejněných přednáškových slajdů, tedy ne v rozsahu tohoto rozšiřujícího textu. Jeho prostudování ovšem může student získat širší nadhled nad přednášenou problematikou, a to i v kontextu reálných aplikací a realizací. Kromě toho by měly „vysvětlující“ části textu napomoci k pochopení těch teoretických partií, které mohou být pro posluchače při dnešní úrovni obecných matematických znalostí a intenzitě návštěv kontaktních výukových akcí obtížnější.

Václav Vais
září 2017

1. Konečné automaty

1.1. Základní principy konečných automatů a jejich uplatnění

Konečný automat (dále jen KA) budeme chápat jako abstraktní model určitého specifického typu výpočtu. Výpočet ovšem neprobíhá s čísly, ale s objekty, které budeme nazývat *symboly*. Tento model má v informatice a výpočetní technice široké použití, jako např.:

- při návrhu sekvenčních logických obvodů (ty jsou základními stavebními prvky téměř veškerého hardware),
- při návrhu, specifikaci a implementaci protokolů pro komunikaci v distribuovaných systémech a počítačových sítích,
- v překladačích programovacích jazyků,
- při řešení jednodušších úloh z oblasti umělé inteligence,
- při modelování architektury softwarových komponent,
- v řídicích systémech logického typu.

KA budeme chápat jako virtuální stroj, který se v každém okamžiku své existence nachází v některém ze stavů z konečné *množiny stavů* (tento stav bývá zvykem nazývat *současný stav* nebo *aktuální stav*). V daném okamžiku se automat nachází právě v jednom stavu, nemůže být v několika stavech současně.

Tento virtuální stroj je obvykle modelem nějakého reálného systému (např. nějakého elektronického zařízení, algoritmu pro řízení komunikace v počítačové síti, algoritmu pro zpracování textového řetězce). Stav modelovaných reálných systémů se může měnit na základě vnějšího podnětu z okolí (u elektronického zařízení změnou úrovně vstupního signálu, u komunikace například příchodem požadavku na navázání spojení se vzdáleným www serverem, u zpracování textového řetězce je podnětem vstup jednoho konkrétního znaku). Tyto vnější podněty přicházejí v diskrétních časových okamžicích (tj. nejedná se o spojitě se měnící vstupy). Na úrovni KA modelu takové podněty budeme reprezentovat *vstupními symboly* z konečné množiny vstupních symbolů. Příchod vnějšího podnětu bude v KA modelován tím, že automat zpracuje konkrétní vstupní symbol.

Důsledkem zpracování vstupního symbolu bude to, že v automatu dojde na základě aktuálního stavu a zpracovaného vstupního symbolu *k přechodu* do *následujícího stavu.* Uvědomme si ale, že následující stav může být shodný se stavem současným, takže v tom případě ke změně stavu vlastně nedojde, přestože z hlediska terminologie budeme říkat, že automat provedl přechod. Ke zpracovaným vstupním symbolům se již KA nemůže vracet.

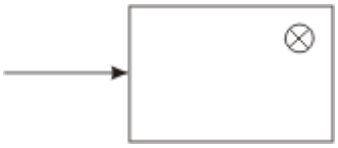
Základními prvky při popisu KA tedy budou *konečná množina stavů, konečná množina vstupních symbolů,* a *přechodová funkce,* která bude pro každý aktuální stav a každý vstupní symbol jednoznačně definovat následující stav. Je samozřejmé, že každé reálné zařízení musí svoji činnost začít v nějakém jednoznačně definovaném stavu, proto i v KA bude definován jeden prvek z množiny stavů jako *počáteční stav*.

Z hlediska různých účelů použití KA modelů má význam definovat více variant KA, jež se budou navzájem lišit způsobem, kterým budou poskytovat svému okolí výstupní informaci. Podíváme-li se na KA jako na „černou skříňku“ (zajímá nás tedy pouze to, jak automat komunikuje s okolím, tj. jak z okolí přijímá informaci a jakou informaci do okolí vydává; naopak nás vůbec nezajímá „co se děje uvnitř“), lze rozlišovat tři typy KA:

- rozpoznávací KA (akceptor)
- klasifikační KA (klasifikátor)
- KA s výstupní funkcí (translátor)

KA všech tří typů zpracovávají řetězce vstupních symbolů *(vstupní řetězce).*

Rozpoznávací automat o zpracovaném řetězci vydá jednoznačné rozhodnutí typu ano/ne (toto rozhodnutí můžeme interpretovat jako „tento řetězec akceptuji, je správný, má požadovanou vlastnost“ nebo „tento řetězec zamítám, není správný, nemá požadovanou vlastnost)“; symbolicky můžeme toto rozhodnutí znázornit jako žárovku na výstupu, která buď svítí, nebo nesvítí).



Klasifikační automat zpracovaný řetězec zařadí do jedné z *n* tříd. Symbolicky můžeme toto rozhodnutí znázornit jako *n* žárovek, z nichž v každém okamžiku svítí právě jedna. Každá žárovka reprezentuje právě jednu klasifikační třídu, automat tedy jednoznačně určuje, do které třídy zpracovaný řetězec patří.

Automat s výstupní funkcí na základě vstupního řetězce vytvoří *výstupní řetězec* ze symbolů z *množiny výstupních symbolů*. To, že automat generuje výstupní řetězce, je v symbolickém obrázku znázorněno výstupní šipkou.

Typické příklady obecně známých zařízení, které lze popsat konečným automatem, jsou například řídicí jednotky nejrůznějších automatů na prodej zboží (kávové automaty, automaty na prodej kusového zboží, jízdenek MHD, ...), řídicí jednotky výtahů, systémy pro řízení provozu na křižovatkách, atd. Uvedené příklady vesměs představují příklady systémů modelovaných KA s výstupní funkcí.

Na tomto místě je ovšem vhodné upozornit na to, že konečný automat je relativně slabý výpočetní model (tj. množina problémů, kterou je pomocí něho možné řešit, je omezená) a že existují i obecnější výpočetní modely, jako například zásobníkový automat nebo Turingův stroj (to je nejobecnější výpočetní model vůbec) Těmi se ale budou zabývat předměty ve vyšších ročnících studia.

1.2. Matematická definice konečného automatu

Rozpoznávací konečný automat je uspořádaná pětice

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

kde Q je konečná neprázdná množina stavů
Σ je konečná neprázdná množina vstupních symbolů (vstupní abeceda)
$q_0 \in Q$ je počáteční stav
$\delta : Q \times \Sigma \to Q$ je přechodová funkce
$F \subseteq Q$ je množina koncových (výstupních) stavů

Množina koncových stavů F v definici rozpoznávacího automatu souvisí s rozhodováním typu *ano/ne* (a tedy i s žárovkou v symbolickém znázornění automatu). Pokud řetězec převede automat do některého ze stavů z množiny koncových stavů *F*, vydává tím automat o řetězci informaci „ano, řetězec akceptuji, je správný“ („žárovka svítí“), v opačném případě „ne, řetězec zamítám, není správný“ („žárovka nesvítí“).

Klasifikační konečný automat je uspořádaná pětice

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

kde Q je konečná neprázdná množina stavů
Σ je konečná neprázdná množina vstupních symbolů
$q_0 \in Q$ je počáteční stav
$\delta : Q \times \Sigma \to Q$ je přechodová funkce
$\{Q_i\}$ je rozklad množiny stavů

Rozkladem množiny stavů $\{Q_i\}$ v definici klasifikačního automatu se rozumí takový systém podmnožin množiny stavů *Q*, že se každý stav z *Q* nachází právě v jednom prvku z $\{Q_i\}$ (prvky z $\{Q_i\}$ se nazývají bloky rozkladu, jsou navzájem disjunktní a jejich sjednocením je množina stavů Q. Každý blok rozkladu odpovídá jedné klasifikační třídě. Podle toho, do kterého bloku rozkladu patří stav, do něhož automat přejde konkrétním vstupním řetězcem, je jednoznačně rozhodnuto o zařazení tohoto řetězce do patřičné třídy.

Automat s výstupní funkcí Mealyho typu je uspořádaná šestice

$$A = (Q, \Sigma, O, \delta, q_0, \lambda)$$

kde Q je konečná neprázdná množina stavů
Σ je konečná neprázdná množina vstupních symbolů
O je konečná neprázdná množina výstupních symbolů
$q_0 \in Q$ je počáteční stav
$\delta : Q \times \Sigma \to Q$ je přechodová funkce
$\lambda : Q \times \Sigma \to O$ je výstupní funkce

Automat s výstupní funkcí Moorova typu je uspořádaná šestice

$$A = (Q, \Sigma, O, \delta, q_0, \lambda)$$

kde Q je konečná neprázdná množina stavů
Σ je konečná neprázdná množina vstupních symbolů
O je konečná neprázdná množina výstupních symbolů
$q_0 \in Q$ je počáteční stav
$\delta : Q \times \Sigma \to Q$ je přechodová funkce
$\lambda : Q \to O$ je výstupní funkce

Rozdíl mezi automatem Mealyho typu a automatem Moorova typu spočívá v tom, že u Mealyho automatu je výstupní symbol jednoznačně definován aktuálním stavem a aktuálním vstupním symbolem, zatímco u Moorova typu výstupní symbol závisí pouze na aktuálním stavu. Znamená to, že u Mealyho automatu výstupní symbol souvisí s přechodem, zatímco u Moorova automatu výstupní symbol souvisí pouze se stavem, v němž se automat nachází.

Důsledkem je, že délka výstupního řetězce generovaného Mealyho automatem je stejná, jako je délka zpracovaného vstupního řetězce (při každém přechodu je vygenerován jeden výstupní symbol). Délka řetězce generovaného Moorovým automatem je o jedničku větší než délka vstupního řetězce (zpracuje-li Moorův automat vstupní řetězec o délce jednoho znaku, bude mít výstupní řetězec dva znaky – první znak odpovídá počátečnímu stavu, druhý znak stavu, do kterého se automat dostal jediným přechodem, který provedl).

Obecnější poznámky k výše uvedeným definicím KA:

1. Je zřejmé, že rozpoznávací automat lze chápat jako zvláštní případ klasifikačního automatu, který by měl rozklad $\{Q_i\}$ definován jako $\{F, \overline{F}\}$

2. Z formálního matematického pohledu, lze dokázat, že ke každému Mealyho automatu lze setrojit Moorův automat, který bude (až na první znak) generovat stejné řetězce a naopak. Existují algoritmy pro převod automatu jednoho typu na automat dru-

hého typu. Z aplikačního hlediska ovšem nelze říci, že jeden model může mechanicky nahradit ten druhý.

Mealyho model se používá k popisu (resp. návrhu) systémů, jejichž výstupy mají pulzní charakter. Příklad systému s pulzními výstupy – řídicí jednotka automatické pračky vysílá signály pulzního charakteru; např. po vygenerování výstupního signálu „otevři elektromagnetický ventil pro napouštění vody“ systém přejde do stavu „napouštění vody“, po přijetí signálu od příslušného hladinového čidla o tom, že bylo dosaženo požadované hladiny, systém přejde do následujícího stavu a v souvislosti s tímto přechodem vyšle pulzní signál „zavři elektromagnetický ventil pro napouštění vody“, apod).

Moorův automat je naopak vhodným nástrojem k popisu systémů, jejichž výstupy mají hladinový charakter. Příklad systému s hladinovými výstupy – světelná signalizace na křižovatce – světla na semaforech jsou po určitou dobu konstantní, změní se až přechodem křižovatky do nového stavu, výstup systému je tedy jednoznačně určen jeho stavem).

3. Je třeba si uvědomit, že stav automatu představuje veškerou podstatnou vstupní historii automatu, tj. prostřednictvím stavu si automat „pamatuje“ veškerou informaci o dosud zpracované části vstupního řetězce, která je nutná pro řešení dané úlohy.

   Příklad: má-li rozpoznávací automat akceptovat všechny řetězce ze symbolů *a* a *b*, které obsahují lichý počet symbolů *b*, stačí mu k tomu dva stavy – jeden představuje, že dosud zpracovaná část řetězce obsahuje sudý počet *b*, druhý stav představuje lichý počet *b.* Pro řešení této konkrétní úlohy není více stavů zapotřebí. Ze stavu KA jsme schopni o vstupním řetězci říci, zda prověřovanou vlastnost má či nemá, nejsme ovšem schopni tento řetězec zpětně zrekonstruovat.

   Konečný automat nemá k dispozici žádnou jinou paměť, vše potřebné si „pamatuje“ prostřednictvím svého stavu.

4. V uvedených definicích nikde nefiguruje čas, přestože se KA (zejména ty s výstupní funkcí) používají k modelování, resp. návrhu systémů, u nichž hraje reálný čas důležitou roli; většinou jsou takové systémy dokonce synchronizovány hodinovými signály. Jinak řečeno – to, že se v reálném elektronickém systému reakce na vstupní změnu neprojeví na výstupu okamžitě, ale s jistým časovým zpožděním, KA model není schopen postihnout, stejně tak není schopen zohlednit ani časové prodlevy mezi podněty, které reálný systém ovlivňují zvnějšku.

5. Rozpoznávací automat a klasifikační automat mají uplatnění zejména při úlohách souvisejících s rozpoznáváním, resp. s klasifikací podle příznaků, což jsou typické úlohy při práci s formálními jazyky a v umělé inteligenci. Tyto problémy se řeší převážně softwarovými nástroji. Naopak Moorův i Mealyho model našly uplatnění nejprve v oblasti hardware a s nimi související metody byly detailně rozpracovány zejména v souvislosti s návrhem sekvenčních logických obvodů.

6. Pro některé aplikace řešené softwarově má význam i zobecnění Mealyho automatu v tom smyslu, že připustíme, aby při jednom přechodu bylo vygenerováno i více výstupních symbolů než jeden (ale také nemusí být vygenerován žádný symbol). Jinak

řečeno - výstupní funkci lze definovat i jako $\lambda : Q \times \Sigma \to O^*$, kde symbol O^* představuje nekonečnou množinu všech řetězců, které lze vytvořit z prvků konečné množiny výstupních symbolů O, a to včetně prázdného řetězce e. V hardwarových aplikacích tento model ovšem uplatnění nenajde (problém se synchronizací a časováním).

7. Všechny výše uvedené definice KA mají tu vlastnost, že k přechodu může dojít pouze na základě zpracování vstupního symbolu a že v každém aktuálním stavu a pro každý vstupní symbol je následující stav definován jednoznačně. Všechny výše popsané automaty lze proto označit přívlastkem *deterministický*. V kapitole 2.11 uvidíme, že existují (a mají i praktický význam) modely, které připouštějí i nejednoznačné chování, tj. například nemusí být jednoznačně určen následující stav, nebo je dokonce umožněno, aby v automatu došlo k samovolnému přechodu, aniž by byl zpracován vstupní symbol. Takovým automatům později budeme říkat *nedeterministické*.

1.3. Způsoby reprezentace konečných automatů

Matematický popis KA pomocí definice uvedený v předchozím textu je sice jednoznačný a úplný, nicméně pro praktické použití bývají výhodnější jiné způsoby popisu automatu:

- přechodový graf (stavový diagram) – grafická reprezentace, pro výukové účely nejvhodnější, do určitého stupně složitosti obrázku je názorná, při velkém počtu stavů a při křížících se hranách přestává být zrakem zvládnutelná
- tabulka – tabulková reprezentace, méně názorná, nicméně zvládnutelná i při větších rozměrech tabulky; vhodné východisko pro implementaci
- stavový strom – kompromis mezi oběma předchozími způsoby

Příklad:

Mějme rozpoznávací automat $A = (Q, \Sigma, \delta, q_0, F)$ zadaný takto:

množina stavů $Q = \{q_0, q_1, q_2, q_3\}$ množina vstupních symbolů $\Sigma = \{0,1\}$

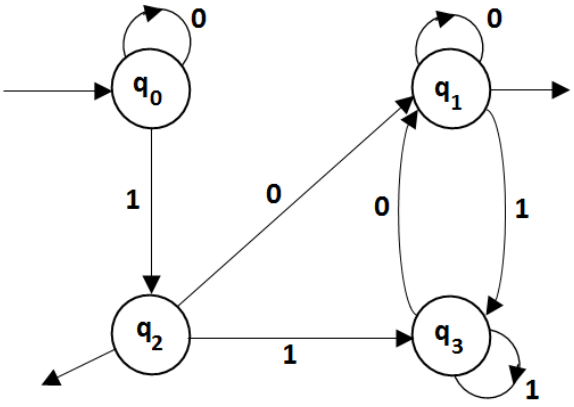
počáteční stav q_0 množina koncových stavů $F = \{q_1, q_2\}$

přechodová funkce δ :

$$\delta(q_0,0) = q_0 \qquad \delta(q_0,1) = q_2$$
$$\delta(q_1,0) = q_1 \qquad \delta(q_1,1) = q_3$$
$$\delta(q_2,0) = q_1 \qquad \delta(q_2,1) = q_3$$
$$\delta(q_3,0) = q_1 \qquad \delta(q_3,1) = q_3$$

Reprezentace KA přechodovým grafem:

Stavům KA odpovídají vrcholy grafu (kolečka), přechodům orientované hrany mezi nimi. Hrany jsou ohodnoceny vstupními symboly, které představují podněty k provedení přechodu. Počáteční stav je označen vstupní šipkou, koncové stavy jsou označeny výstupními šipkami.



V (úplném) stavovém diagramu z každého stavu vychází tolik výstupních hran, kolik je prvků ve vstupní abecedě (pro každý vstupní symbol jedna hrana). Někdy lze použít zjednodušeného stavového diagramu, ve kterém nejsou zakresleny všechny hrany. Chybějící hrany se potom interpretují různým způsobem podle toho, jedná-li se o rozpoznávací automat nebo automat s výstupní funkcí (u klasifikačních automatů bývá zvykem kreslit všechny přechodové hrany).

Reprezentace KA tabulkou:

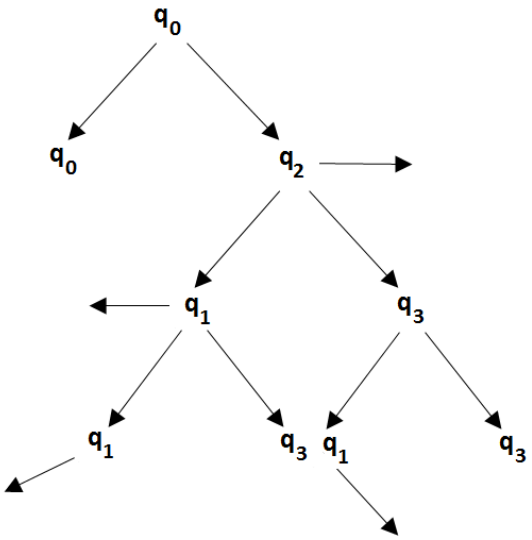
Stavům odpovídají řádky tabulky, vstupním symbolům pak její sloupce. Každé políčko tabulky tedy odpovídá právě jedné dvojici <stav, vstupní symbol>, v odpovídajících pozicích jsou zapsány následující stavy příslušných přechodů. Řádek odpovídající počátečnímu stavu je označen vstupní šipkou, řádky odpovídající koncovým stavům jsou označeny výstupními šipkami.

		0	**1**
→	**q_0**	q_0	q_2
←	**q_1**	q_1	q_3
←	**q_2**	q_1	q_3
	q_3	q_1	q_3

Reprezentace tabulkou je často používána při softwarové implementaci KA modelů. Počáteční stav je v tom případě použit k iniciaci proměnné, která bude reprezentovat stav automatu, koncové stavy jsou obvykle reprezentovány jedničkami v bitovém poli, které má stejný počet prvků, jako je stavů.

Reprezentace KA stavovým stromem:

Strom začínáme vytvářet od počátečního stavu. Umístíme jej v obrázku stromu nejvýš (strom budeme vytvářet shora dolů). Postupně budeme přidávat pro každý vstupní symbol jednu hranu orientovanou směrem dolů (resp. šikmo dolů) a s ní i jeden výskyt následujícího stavu. Rozvoj stromu pokračuje tak dlouho, dokud se v listech stromu neobjeví stavy, jejichž výskyty jsou již někde ve stromu rozvinuty. Počáteční stav není třeba nijak označovat, je zřejmý z toho, že je kořenem stromu. **Koncové stavy jsou označeny výstupními šipkami u všech jejich výskytů**. To proto, aby při zjišťování, zda je konkrétní stav koncový, či nikoli, nebylo nutné procházet celý strom a hledat všechny výskyty zkoumaného stavu.



Zjednodušená reprezentace KA stavovým diagramem:

U rozpoznávacích automatů a automatů s výstupní funkcí může čtenář narazit na stavové diagramy, ve kterých některé přechodové hrany nejsou zakresleny.

Chybí-li v přechodovém grafu rozpoznávacího KA v konkrétním stavu pro konkrétní vstupní symbol výstupní hrana, interpretujeme to tak, že je-li v tomto stavu na vstupu KA tento vstupní symbol, automat zpracovávaný řetězec zamítne, a to bez ohledu na to, jaké další znaky ve vstupním řetězci následují. Jinak řečeno: nezakreslenou hranu si můžeme představit tak, že převádí automat do (ve zjednodušené reprezentaci nezakresleného) chybového stavu. Tento chybový stav nepatří do množiny koncových stavů F a pro každý vstupní symbol v něm je smyčka, stav je tedy *absorpční* (automat tento stav již nikdy neopustí).

U automatu s výstupní funkcí chybějící hrany interpretujeme tak, že automat zpracováním vstupního symbolu zůstane v aktuálním stavu. Jedná-li se navíc o automat Mealyho typu, generuje automat při tomto přechodu speciální *neutrální výstupní symbol (mezeru)*, který je z pohledu modelované reality bezvýznamný (na výstupu automatu se „nestane nic“, pouze se takto formálně vyhovuje definici Mealyho KA v tom smyslu, že při každém přechodu je generován výstupní symbol).

1.4. Chování konečného automatu

Chceme-li nějakým způsobem popisovat chování konkrétního automatu, můžeme k tomu mít dva důvody:

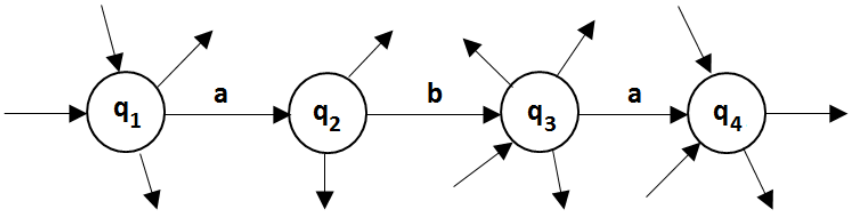
- zajímá nás, co je výsledkem zpracování konkrétního vstupního řetězce; k tomu lze přistoupit ze dvou různých hledisek a to
    - z pohledu vztahu mezi vstupním řetězcem a stavem, do něhož byl automat tímto řetězcem převeden (relace vstup – stav; tento pohled lze uplatnit u všech výše definovaných typů automatů stejným způsobem)
    - z pohledu vztahu mezi vstupním řetězcem a informací, kterou zpracováním tohoto řetězce automat vydává (relace vstup – výstup; bude se samozřejmě lišit podle typu automatu)

- chceme sledovat vývoj automatu „krok za krokem“, tj. jak se mění stav postupným zpracováním vstupního řetězce, případně také jak je u automatů s výstupní funkcí postupně generován výstupní řetězec

Vztah mezi zpracovaným vstupním řetězcem a stavem, zobecněná přechodová funkce

Z definice přechodové funkce δ (deterministického) KA je zřejmé, že tato funkce pro každý stav a každý vstupní symbol jednoznačně definuje následující stav. Jednoduchou úvahou lze dospět k tomu, že tedy přechodová funkce (nepřímo) jednoznačně definuje i následující stav pro každý stav a každý vstupní řetězec (zpracovává-li automat řetězec délky *n*, provede *n* jednoznačných přechodů, takže stav, do něhož je automat převeden zpracováním *n* vstupních symbolů, je určen také jednoznačně).

Názorně: zkoumejme, jak automat reprezentovaný následujícím výsekem z přechodového grafu zareaguje ve stavu q_1 na vstupní řetězec *aba* :



Automat zřejmě provede tři přechody:

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

Označíme-li symbolem δ^* funkci, kterou budeme chápat jako rozšíření přechodové funkce δ v tom smyslu, že bude definovat následující stav nejen pro jednotlivé vstupní symboly, ale pro řetězce ze symbolů vstupní abecedy, můžeme v našem případě psát

$$\delta^*(q_1, aba) = q_4$$

Funkci δ^* budeme nazývat *zobecněná přechodová funkce*. Z výše uvedeného je zřejmé, že je tato funkce jednoznačně určena přechodovou funkcí δ.

Přechodová funkce: $\delta : Q \times \Sigma \to Q$

Zobecněná přechodová funkce: $\delta^* : Q \times \Sigma^* \to Q$

Definiční obor přechodové funkce: $D(\delta) = Q \times \Sigma$ tj. množina všech uspořádaných dvojic (stav, vstupní symbol)

Definiční obor zobecněné přechodové funkce:

$D(\delta^*) = Q \times \Sigma^* = Q \times (\{e\} \cup \Sigma \cup \Sigma^2 \cup \ldots) = Q \times \bigcup_{i=0}^{\infty} \Sigma^i$ tj. množina všech uspořádaných dvojic (stav, vstupní řetězec).

V zápisu jsou sice použity symboly, které jsou korektně zavedeny až v kapitole 2.2, nicméně pro pochopení výše uvedeného by mělo postačit následující vysvětlení.

- Symbol Σ^2 představuje množinu všech řetězců délky 2 znaky, Σ^3 množinu všech řetězců délky 3 znaky, ……, Σ^i množinu všech řetězců délky *i* znaků.
- Nesmíme zapomenout na to, že i prázdný řetězec *e* je vstupním řetězcem (je to řetězec délky 0 znaků) a patří tedy do množiny Σ^*.

Vztah mezi definičními obory $D(\delta^*)$ a $D(\delta)$ můžeme znázornit takto:

Zřejmě platí $D(\delta) \subseteq D(\delta^*)$ a $\delta^*(q,a) = \delta(q,a)$ $\forall q \in Q, \forall a \in \Sigma$, lze tedy konstatovat, že zobecněná přechodová funkce δ^* je rozšířením přechodové funkce δ na definiční obor $Q \times \Sigma^*$. Někteří autoři používají pro zobecněnou přechodovou funkci stejný symbol

jako pro funkci přechodovou, tedy δ, protože význam symbolu lze buď poznat podle kontextu, nebo obě funkce rozlišit nelze, v tom případě ale mají stejné funkční hodnoty.

Zobecněná přechodová funkce δ^* je definována rekurzivně pomocí přechodové funkce δ takto:

$$\delta^*(q,wa) = \delta(\delta^*(q,w),a) \qquad \forall q \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma$$
$$\delta^*(q,e) = q \qquad \forall q \in Q$$

Vysvětlení k rekurzivní definici funkce δ^*: je zřejmé, že při zpracování vstupního řetězce délky *n* bude jako poslední přechod proveden přechod iniciovaný posledním znakem vstupního řetězce. Je proto vhodné vyjádřit si neprázdný vstupní řetězec tak, že formálně osamostatníme poslední znak, tedy jako wa. Stav, do něhož automat přejde posledním symbolem a, bude definován hodnotou přechodové funkce δ pro jakýsi stav x a vstup a. Do stavu x se automat dostal z výchozího stavu q vstupním řetězcem w, proto jej můžeme vyjádřit opět pomocí zobecněné přechodové funkce jako $\delta^*(q,w)$. Definici hodnoty zobecněné přechodové funkce pro stav q a vstupní řetězec délky n jsme tedy převedli na hodnotu téže funkce pro tentýž stav a vstupní řetězec délky o 1 kratší (princip rekurze). Aby byla definice úplná, musíme ještě dodefinovat hodnoty pro prázdný vstupní řetězec jako $\delta^*(q,e) = q$ (KA je deterministický, nemůže změnit svůj stav, aniž by zpracoval vstupní písmeno, prázdný řetězec žádný přechod nezpůsobí, proto automat zůstane ve stavu q).

Z toho, že je zobecněná přechodová funkce definována výše uvedeným vztahem jako tranzitivní uzávěr, vyplývá její důležitá vlastnost:

$$(\delta^*(q,u) = \delta^*(q,v)) \Rightarrow (\delta^*(q,uw) = \delta^*(q,vw)) \qquad \forall q \in Q, \forall u,v,w \in \Sigma^*$$

Vysvětlení výše uvedeného tvrzení: předpoklad implikace říká, že automat přejde ze stavu q vstupním podřetězcem u do stejného stavu, jako by přešel z q vstupním podřetězcem v. Platnost důsledku je intuitivně pochopitelná, protože to, jak bude automat poté v obou případech reagovat na následující podřetězec w, závisí pouze na stavu, v němž je automat, když začne podřetězec w zpracovávat (a ten stav je podle předpokladu po zpracování u i po zpracování v stejný). Nezáleží na tom, jakými přechody se automat do tohoto stavu dostal (stav reprezentuje veškerou vstupní historii automatu).

Je zřejmé, že zobecněná přechodová funkce jednoznačně definuje na množině všech vstupních řetězců Σ^* rozklad o n blocích (kde n označuje počet stavů KA). Každý blok rozkladu odpovídá jednomu stavu automatu a patří do něj právě ty vstupní řetězce, které převádí automat z počátečního stavu do stavu odpovídajícího konkrétnímu bloku. Tento rozklad množiny všech vstupních řetězců jednoznačně indukuje relaci ekvivalence na množině všech vstupních řetězců. Ve stejné třídě ekvivalence budou ty vstupní řetězce, které z počátečního stavu převedou automat do stejného stavu. Tuto ekvivalenci označíme operátorem $\approx$. Protože je relace δ^* tranzitivním uzávěrem relace δ, je ekvivalence $\approx$ *pravou kongruencí*, tj. platí

$$(u \approx v) \Rightarrow (uw \approx vw) \qquad \forall u,v,w \in \Sigma^*$$

Navíc je ekvivalence $\approx$ pravou kongruencí *konečného indexu* (má konečný počet tříd, protože KA má konečný počet stavů). Relace s těmito vlastnostmi mají význam v teorii regulárních jazyků, jak uvidíme později (viz Nerodova věta v kapitole 2.19).

Vztah mezi vstupním řetězcem a výstupní informací (relace vstup – výstup), překladová funkce

Při tomto přístupu budeme na KA nahlížet jako na nějakou transformaci vstupních dat (viz symbolická znázornění v odstavci 1.1). Tuto transformaci označíme symbolem *T*.

Rozpoznávací automat o každém vstupním řetězci vydá rozhodnutí typu ano/ne, můžeme tedy psát

$$T : \Sigma^* \to \{0,1\}$$

kde 0 reprezentuje rozhodnutí „řetězec zamítám, není správný, nemá požadovanou vlastnost“, 1 reprezentuje rozhodnutí „řetězec akceptuji, je správný, má požadovanou vlastnost“.

Klasifikační automat o každém vstupním řetězci vydá rozhodnutí typu *1 z n*

$$T : \Sigma^* \to \{1,2,....,n\}$$

kde $i \in N, i \le n$ reprezentuje klasifikační třídu, do které je vstupní řetězec automatem zařazen.

Automat s výstupní funkcí transformuje vstupní řetězce na výstupní řetězce.

$$T : \Sigma^* \to O^*$$

Tuto transformaci realizuje překladová funkce β, která je jednoznačně určena výstupní funkcí λ a zobecněnou přechodovou funkcí δ^*. Překladová funkce β je definována rekurzivně.

Překladová funkce β u Mealyho automatu:

$$\beta : Q \times \Sigma^* \to O* \quad , \quad \delta^* : Q \times \Sigma^* \to Q \quad , \quad \lambda : Q \times \Sigma \to O$$

$$\begin{aligned} &\beta(q, wa) = \beta(q, w)\lambda\left(\delta^*(q, w), a\right) && \forall q \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma \\ &\beta(q, e) = e && \forall q \in Q \end{aligned}$$

Překladová funkce β u Moorova automatu:

$$\beta : Q \times \Sigma^* \to O* \quad , \quad \delta^* : Q \times \Sigma^* \to Q \quad , \quad \lambda : Q \to O$$

$$\begin{aligned} &\beta(q, wa) = \beta(q, w)\lambda\left(\delta^*(q, wa)\right) && \forall q \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma \\ &\beta(q, e) = \lambda(q) && \forall q \in Q \end{aligned}$$

Vysvětlení k rekurzivní definici překladové funkce: podobně jako u definice zobecněné přechodové funkce δ^* vyjádříme neprázdný vstupní řetězec jako *wa* (formálně osamostatníme

poslední znak). Výstupní řetězec pak bude zřetězením výstupního řetězce příslušejícího vstupnímu řetězci *w* (lze jej vyjádřit také překladovou funkcí β) a posledního symbolu výstupního řetězce, který bude definován výstupní funkcí λ (sekvenční vlastnost). V případě Mealyho automatu je poslední výstupní symbol určen posledním vstupním symbolem a stavem, do něhož se automat dostal před jeho zpracováním. V případě Moorova automatu je poslední výstupní symbol určen pouze posledním stavem, tj. stavem, do něhož automat přešel zpracováním celého řetězce.

Vlastnosti překladové funkce β:

1. u Mealyho automatu funkce zachovává délku řetězců (tj. vstupní řetězec má stejnou délku jako výstupní řetězec), u Moorova automatu transformace zvětšuje délku řetězce o 1 (souvisí to tím, jak je definován výstupní řetězec pro prázdný vstupní řetězec),
2. mají-li dva vstupní řetězce shodné počátky v délce *n* znaků, jsou shodné i počátky výstupních řetězců v délce *n* znaků (u Mealyho automatu), resp. v délce *n+1* znaků (u Moorova automatu).

Vývoj konečného automatu, konfigurace automatu

Chceme-li sledovat vývoj KA automatu „krok za krokem“, zajímá nás, jak se mění stav postupným zpracováním vstupního řetězce, případně jak je u automatů s výstupní funkcí postupně generován výstupní řetězec.

Úplnou informaci o stavu zpracování řetězce v daném kroku poskytuje *konfigurace automatu.* Pod pojmem konfigurace automatu budeme u rozpoznávacího nebo klasifikačního automatu rozumět uspořádanou dvojici *(stav, dosud nezpracovaná část vstupního řetězce)*, u automatu s výstupní funkcí pak uspořádanou trojici *(stav, dosud nezpracovaná část vstupního řetězce, dosud vygenerovaná část výstupního řetězce).*

Vývoj KA při zpracování konkrétního vstupního řetězce lze potom vyjádřit jako posloupnost konfigurací automatu.

Relačním operátorem $\mapsto$ budeme označovat přechod mezi konfiguracemi automatu.

Platí

$$(q_1, aw) \mapsto (q_2, w) \Leftrightarrow \delta(q_1, a) = q_2 \qquad \forall q_1, q_2 \in Q, \forall a \in \Sigma, \forall w \in \Sigma^*$$

Vysvětlení: Je-li KA ve stavu q_1 a na vstupu je znak *a*, přejde automat do stavu, který je přechodovou funkcí definován jako $\delta(q_1, a)$. Písmeno a je tímto zpracováno a dosud nezpracovanou částí vstupního řetězce zůstává řetězec, který zpracované *a* následuje, tedy w.

Příklad:

Mějme rozpoznávací automat $A = (Q, \Sigma, \delta, q_0, F)$ zadaný tabulkou:

		0	1
→	**q_0**	q_3	q_2
←	**q_1**	q_1	q_3
←	**q_2**	q_2	q_3
	q_3	q_1	q_2

Zpracování vstupního řetězce 010010 tímto automatem můžeme vyjádřit jako posloupnost konfigurací takto:

$$(q_0,010010) \mapsto (q_3,10010) \mapsto (q_2,0010) \mapsto (q_2,010) \mapsto (q_2,10) \mapsto (q_3,0) \mapsto (q_1,e)$$

Protože $q_1 \in F$ (q_1 patří do množiny koncových stavů), je zpracovaný vstupní řetězec automatem akceptován.

1.5. Příklady z aplikačních oblastí

1.5.1. Komunikační protokol TCP jako konečný automat

Cílem této kapitoly není detailní popis protokolu TCP, ale objasnění principu popisu komunikačních protokolů stavovými diagramy.

Protokol TCP zajišťuje v počítačových sítích s architekturou TCP/IP spojovanou transportní službu. Protokol zabezpečuje vytvoření a provoz (a při ukončování relace i zrušení) obousměrného virtuálního přenosového kanálu, do kterého vysílající aplikace postupně vkládá datové pakety, a přijímající aplikace je zase z kanálu odebírá. Spojovaná služba zajišťuje, že přenášené datové pakety jsou doručeny ve správném pořadí (na rozdíl od služeb nespojovaných, kde mohou datové pakety mezi vysílačem a přijímačem putovat různými cestami, takže není zaručeno, že dojdou ve správném pořadí). Spojovanou službu TCP využívají takové aplikace, jako např. www, elektronická pošta, přenos souborů.

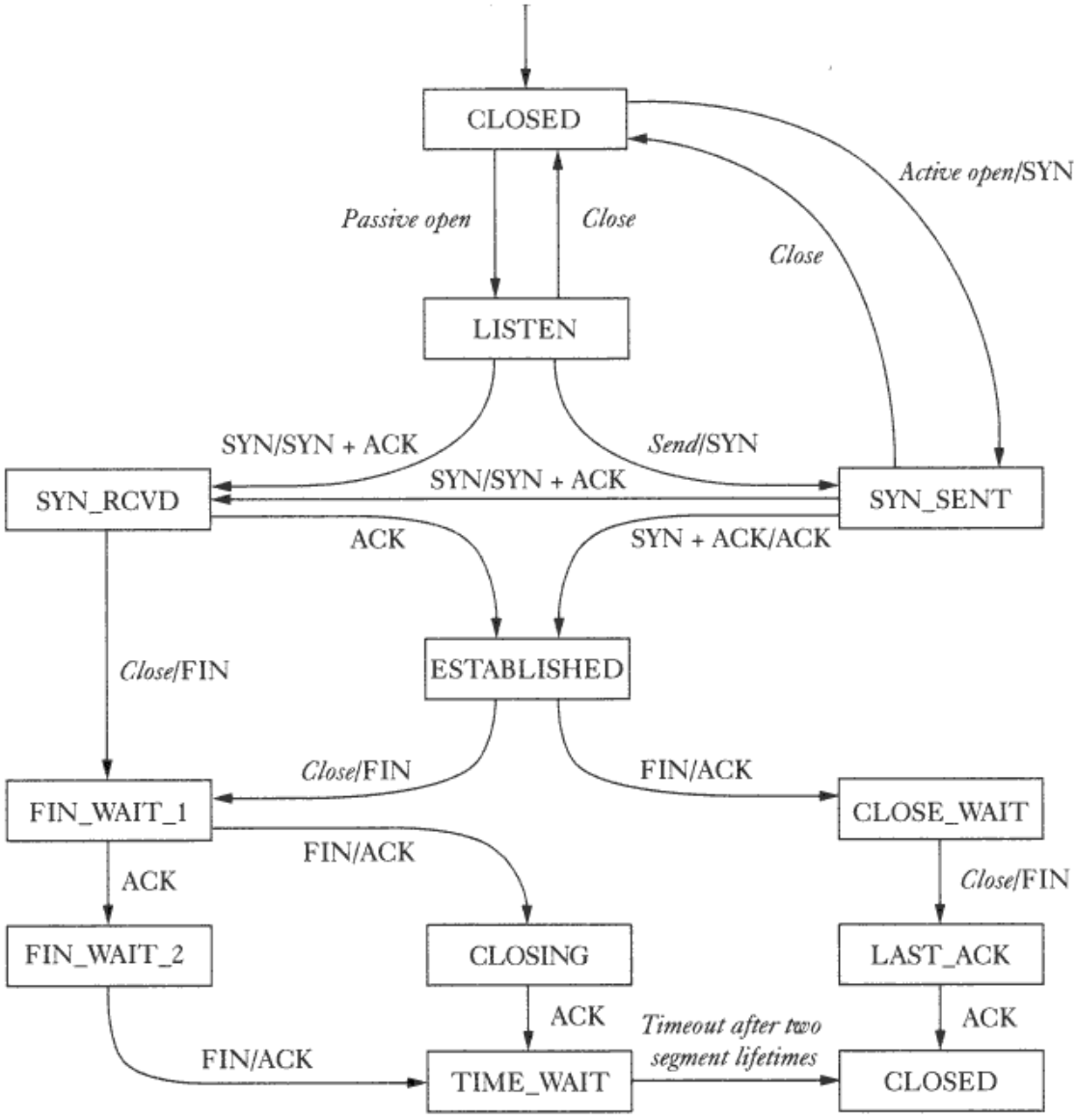
Přenosovou jednotkou v TCP je paket. Kromě datové části může mít paket nastaveno i několik příznaků v záhlaví paketu, které se využívají k řízení spojení (datová část paketu může i chybět, takový paket pak slouží pouze k řízení spojení). Řídícími příznaky jsou např.:

SYN - příznak, který se používá při navazování TCP spojeni pro synchronizaci čísel sekvence (čísla sekvence = mechanizmus „číslování dat“ umožňující potvrzování správnosti přijatých datových paketů)

FIN - příznak, který se používá při ukončování TCP spojeni

ACK - příznak, jehož nastavení říká, že paket obsahuje platné potvrzení sekvence

Následující stavový diagram představuje část závazné definice protokolu TCP v RFC 793, která popisuje, jakým způsobem probíhá navazování a ukončování spojení:



Obdélníčky představují stavy spojení z pohledu konkrétního účastníka spojení (tj. serveru nebo přistupujícího klienta). Textové řetězce, které ohodnocují hrany, budeme interpretovat takto:

- pokud se v řetězci vyskytne znak / , znamená to, že nalevo od něj je událost, která přechod vyvolá a napravo od lomítka je informace reprezentující výstupní akci

- pokud se v řetězci nevyskytne znak / , reprezentuje tato hrana přechod, jenž je iniciován vstupní událostí popsanou řetězcem a negeneruje žádnou výstupní akci
- je-li vstupní událost psaná kurzívou, představuje událost, kterou vyvolala nadřízená vrstva protokolu (zjednodušíme-li, tedy aplikace na „mé stanici“). Jsou uvažovány tyto události:
    - *Passive open* – pasivní otevření spojení serverem, tedy otevření portu se standardním číslem portu; server je připraven navázat spojení s jakýmkoli klientem, který o to na daném portu požádá
    - *Active open* - aktivní otevření spojení klientem, tedy začátek navazování spojení klienta s konkrétním serverem;
    - *Send* - (v architektuře klient – server nepoužívané) – překlopení pasivního otevření na aktivní, tj. stanice přestává pasivně čekat na navázání spojení a sama je aktivně navazuje,
    - *Close* - požadavek na ukončení spojení iniciovaný aplikační vrstvou
- je-li vstupní událost psána standardním písmem, představuje konkrétní příznakový bit z paketu přijatého od partnerské TCP vrstvy (tedy od „stanice“ s níž „má stanice“ komunikuje),
- texty za lomítkem představují řídicí příznaky, které budou nastaveny v odesílaném paketu; je-li v řetězci znak + , znamená to, že bude v odesílaném paketu nastaveno více řídicích příznaků.

Pojem „má stanice“ ve výše uvedeném textu představuje toho partnera spojení, z jehož pohledu stav spojení sledujeme. Uvědomme si, že jedno TCP spojení je reprezentováno dvěma stavovými diagramy, jedním na „mé stanici“, jedním na stanici, s níž „má stanice“ komunikuje. Ve fázi, kdy je spojení navázáno, jsou obě strany spojení ve stejném stavu ESTABLISHED, fáze navazování a rušení spojení ale nejsou symetrické, stavy obou stran se v průběhu těchto fází liší.

Cílem této kapitoly není detailní popis protokolu TCP, detailněji si tedy popíšeme pouze některé stavy a některé přechody.

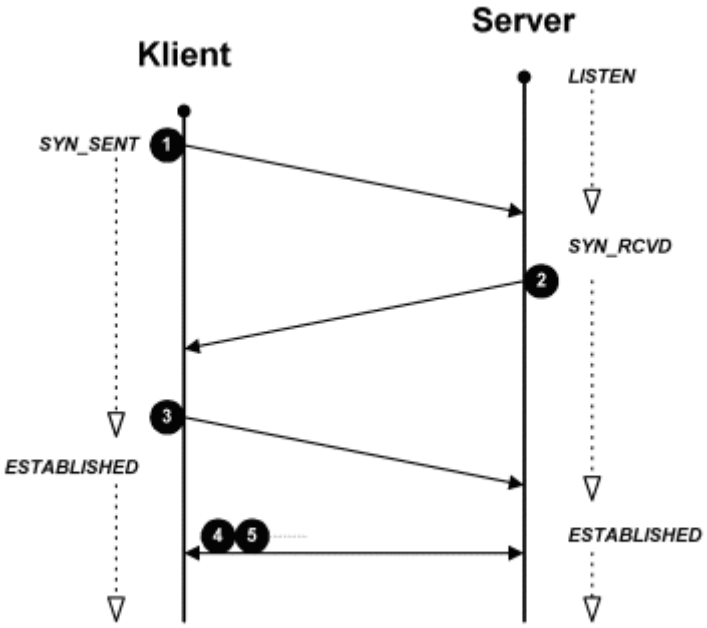
- stav CLOSED reprezentuje stav, kdy spojení není navázáno, tedy vlastně ještě neexistuje (nenechme se zmást dvěma výskyty téhož stavu v obrázku, oba reprezentují totéž)
- stav LISTEN je stav spojení na straně serveru, ve kterém server čeká na příchozí spojení (na straně serveru je otevřen port)
- stav SYN-SENT je stav spojení na straně klienta, ve kterém probíhá navazování nového spojení (klient odeslal SYN paket, předpokládá, že server čeká na příchozí spojení ve stavu LISTEN a že spojení potvrdí paketem s příznaky SYN a ACK)
- stav SYN-RCVD je stav spojení na straně serveru, který reprezentuje navazování nového spojení klientem (server přijal SYN paket, odpověděl zasláním SYN a ACK paketu a očekává potvrzení paketem ACK)
- stav ESTABLISHED je stav, ve kterém je spojení již vytvořeno a probíhají v něm datové přenosy

Nyní můžeme ze stavového diagramu odvodit, jak bude vypadat komunikace na úrovni TCP paketů při navazování spojení mezi klientem a serverem. Předpokládáme přitom, že je na

serveru otevřen příslušný port, stavový diagram spojení z pohledu serveru je tedy ve stavu LISTEN, spojení na straně klienta neexistuje, z pohledu stavového diagramu je tedy ve stavu CLOSED.

1. Na základě podnětu z aplikační vrstvy klientské stanice začíná klient na úrovni TCP navazovat spojení odesláním paketu s řídicím příznakem SYN. Změna stavu ve stavovém diagramu:
    - na straně klienta – vstupní událost *Active open* iniciovala přechod ze stavu CLOSED do stavu SYN-SENT; při tomto přechodu je odeslán řídicí paket SYN
2. TCP vrstva serveru přijme od protistrany řídicí paket SYN a odpoví paketem s nastavenými příznaky SYN a ACK (nastavení sekvenčního čísla kvůli potvrzování). Změna stavu ve stavovém diagramu:
    - na straně serveru – vstupní událost ACK iniciovala přechod ze stavu LISTEN do stavu SYN-RCVD; při tomto přechodu je odeslán řídicí paket SYN + ACK
3. TCP vrstva klienta přijme řídicí paket SYN + ACK a odpoví paketem s nastaveným příznakem ACK. Změna stavu ve stavovém diagramu:
    - na straně klienta – vstupní událost SYN + ACK iniciovala přechod ze stavu SYN-SENT do stavu ESTABLISHED; při tomto přechodu je odeslán řídicí paket ACK
4. TCP vrstva serveru přijme řídicí paket ACK. Potvrdí jej také paketem ACK a přejde do stavu ESTABLISHED
    - na straně serveru – vstupní událost ACK iniciovala přechod ze stavu SYN-RCVD do stavu ESTABLISHED při tomto přechodu je odeslán řídicí paket SYN + ACK

Časový průběh této komunikace ilustruje následující obrázek:



Po této komunikaci, obvykle označované jako *three-way handshake,* následuje fáze přenosu dat.

Vysvětlováním mechanizmu ukončování spojení se zabývat nebudeme. Čtenáře, kterým k pochopení nestačí stavový diagram protokolu, odkážeme na učební materiály k předmětům z oblasti počítačových sítí.

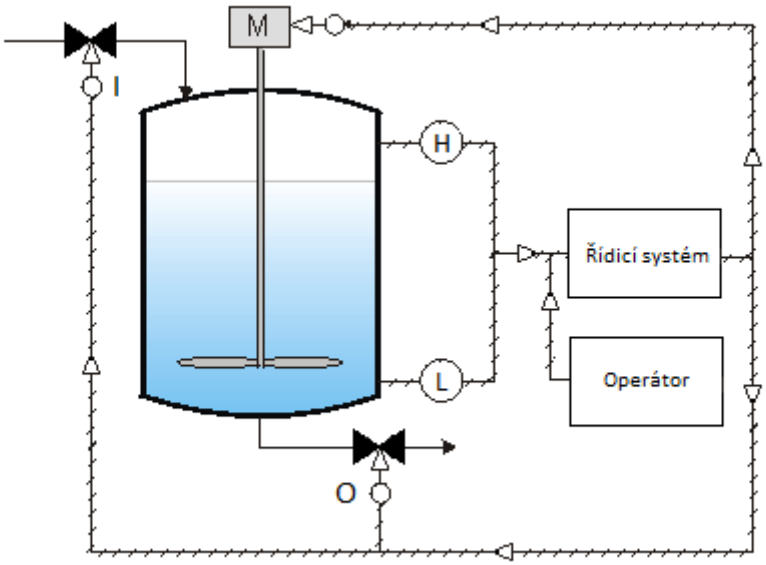
1.5.2. Logické řízení

Automatické řízení budeme pro účely této kapitoly chápat obecně jako cílené působení nějakého *řídicího systému* na *řízený objekt* tak, aby bylo dosaženo určeného cíle.

Řídicí systémy zpracovávají vstupní informace o stavu řízeného objektu, které poskytují *snímače (čidla)*. *Logické řízení* je specifický typ automatického řízení, u něhož vstupy od snímačů nabývají pouze konečného počtu hodnot (obvykle jsou dvouhodnotové, např. signál od hladinového čidla „hladina tekutiny v nádrži je pod, resp. nad sledovanou úrovní“). Řídicí systém pak generuje výstupy, tj. řídicí signály (u logického řízení opět obvykle dvouhodnotové), kterými ovlivňuje *akční členy*, jež provádějí zásah do řízeného objektu (např. zapnutí/vypnutí pohonu, otevření/zavření ventilu apod). Hodnota řídicích signálů obvykle nezávisí pouze na okamžitých hodnotách vstupů, ale i na předchozích hodnotách vstupů, čili na stavu procesu.

Jako příklad uvedeme model logického řízení míchací nádrže, inspirovaný úlohou z http://uprt.vscht.cz/ucebnice/mrt/F5/F5k52-prlr.htm (Kadlec, Kmínek). Technologické schéma je znázorněno na obrázku. Nádrž se bude cyklicky napouštět a vypouštět, takže hladina tekuté směsi uvnitř se bude měnit. Do procesu může zasahovat operátor, jenž má kromě spuštění a ukončení procesu možnost jeho pozastavení, spočívající v tom, že se uzavře přítok i odtok, ale bude dál probíhat míchání. Po obnovení pozastaveného procesu se bude pokračovat v té fázi, v níž byl proces přerušen (tj. buď napouštěním nebo vypouštěním). Logické řízení má navíc zajistit, aby míchadlo neběželo, když je nádrž prázdná (tj. hladina je pod spodní sledovanou úrovní). Když operátor proces ukončí, musí dojít k vyprázdnění nádrže za pokračujícího míchání.

Plné čáry v následujícím schématu představují přítok a odtok tekuté směsi, přerušované čáry představují informační a řídicí signály. Čidly jsou snímače hladiny H a L, akčními členy elektromagnetické ventily I a O a motor míchadla M.



Snímače hladiny H a L budou v modelu reprezentovat vstupní symboly H1, L1, H0, L0, které představují signály „hladina stoupla nad sledovanou úroveň“, resp. „hladina klesla pod sledovanou úroveň“:

H1 …… hladina stoupla nad horní sledovanou úroveň
H0 ……hladina klesla pod horní sledovanou úroveň
L1 …… hladina stoupla nad spodní sledovanou úroveň
L0 …… hladina klesla pod spodní sledovanou úroveň

Řídicí signály ovládající akční členy, tj. pohon míchadla, napouštěcí ventil a vypouštěcí ventil, chápeme jako signály s pulzním charakterem. V našem modelu budou reprezentovány výstupními symboly I1, I0, O1, O0, M1, M0 s těmito významy:

I1 …… otevři napouštěcí ventil
I0 …… uzavři napouštěcí ventil
O1 …… otevři vypouštěcí ventil
O0 …… uzavři vypouštěcí ventil
M1 …… spusť pohon míchadla
M0 …… vypni pohon míchadla

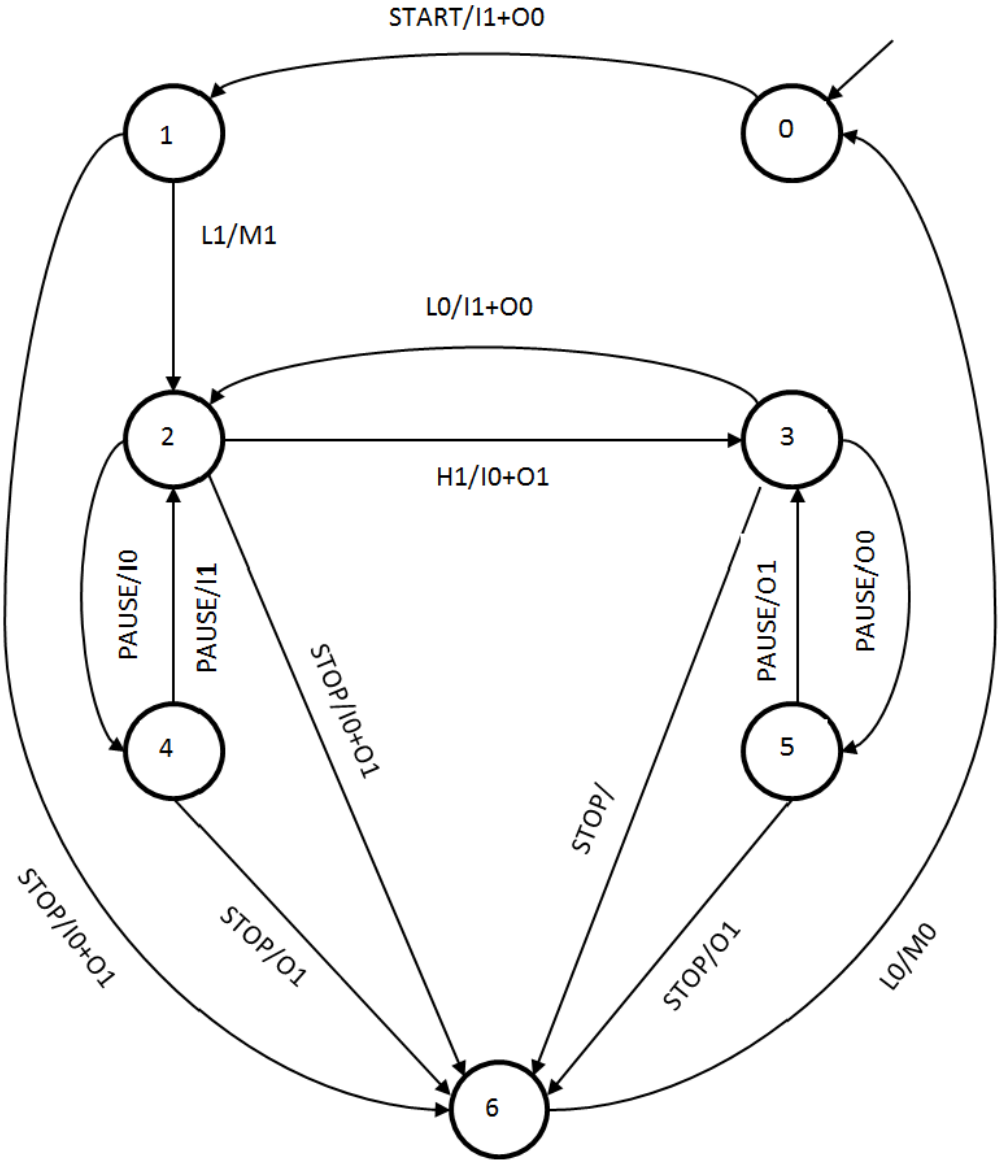
Zásahy operátora budou v modelu reprezentovat vstupní symboly START, STOP a PAUSE s těmito významy:

START…… spusť proces
STOP……. ukonči proces
PAUSE….. pozastav proces, resp. obnov pozastavený proces

Řídicí systém míchací nádrže lze popsat stavovým diagramem na následující straně. Jednotlivé stavy modelu lze charakterizovat takto:

STAV 0 …… SYSTÉM NENÍ V ČINNOSTI
STAV 1 …… NAPOUŠTĚNÍ BEZ MÍCHÁNÍ, ČEKÁNÍ NA POTŘEBNOU MINIMÁLNÍ HLADINU

STAV 2 …… NAPOUŠTĚNÍ S MÍCHÁNÍM
STAV 3 …… VYPOUŠTĚNÍ
STAV 4 …… PROCES POZASTAVEN PŘI NAPOUŠTĚNÍ
STAV 5 …… PROCES POZASTAVEN PŘI VYPOUŠTĚNÍ
STAV 6 …… KONEČNÉ VYPOUŠTĚNÍ PŘED UKONČENÍM PROCESU



Textové řetězce, které ohodnocují hrany diagramu, budeme interpretovat takto:

- nalevo od znaku / je symbol reprezentující signál od čidla (resp. operátora), který příslušný přechod vyvolá,
- texty za lomítkem představují výstup řídicího systému, tedy signály, které budou budit akční členy; je-li v řetězci znak + , znamená to, že bude buzeno více akčních členů současně,

Vyskytne-li se v některém stavu vstup, pro který v přechodovém grafu neexistuje hrana, znamená to, že jej řídicí systém ignoruje (zůstává v daném stavu, negeneruje žádný výstupní

signál). To, že při přechodu ze stavu 3 do stavu 6 vstupem STOP nejsou za lomítkem uvedeny žádné výstupní symboly, znamená, že není třeba budit žádný akční člen (ve stavu 3 – VYPOUŠTĚNÍ je vstupní ventil uzavřen, výstupní ventil otevřen a pohon míchadla běží, což je přesně to, co potřebujeme pro konečné vypouštění při ukončování procesu).

K realizaci řídicího systému míchací nádrže by postačil nepříliš složitý jednoúčelový sekvenční logický obvod navržený způsobem nastíněným v kapitole 1.7. Ve většině praktických průmyslových aplikací se ale uplatňují univerzální logické řídicí systémy, jejichž funkce je možné programovat podle potřeby (programovatelné logické automaty PLC). Pro programování automatů se používají buď jednoúčelové programovací aparáty nebo běžné osobní počítače se speciálním programovým vybavením.

1.5.3. Vstupní konverze číselných konstant

Pod pojmem *vstupní konverze* budeme rozumět postup, kterým z textového řetězce, jenž představuje zápis číselné konstanty, vytvoříme její reprezentaci ve vnitřním formátu počítače, tj. v pevné řádové čárce pro konstanty typu INTEGER nebo v pohyblivé řádové čárce pro konstanty typu REAL. Tuto úlohu standardně řeší překladače vyšších programovacích jazyků při překladu zdrojového programu.

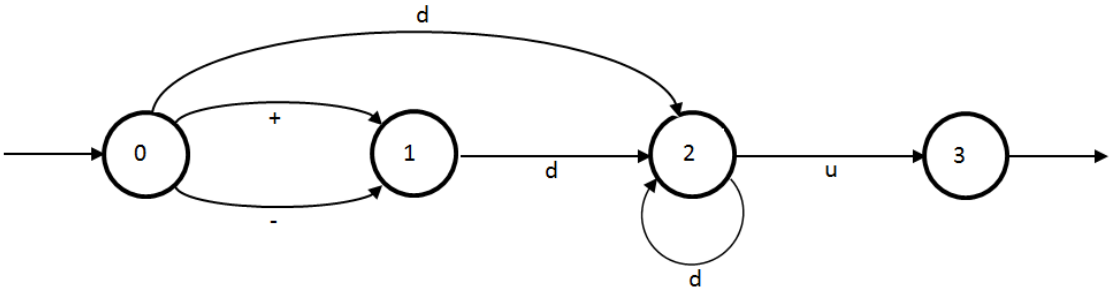
Pro ilustraci – reprezentace některých konstant ve znakové reprezentaci a ve formátu INTEGER s délkou slova 4 byty (obsahy jednotlivých bytů jsou zapsány hexadecimálně):

Konstanta	**Reprezentace ve znakovém poli (kódování ASCII)**							**Reprezentace INTEGER**			
314	33	31	34	20				00	00	01	3A
-314	2D	33	31	34	20			FF	FF	FE	C6
+27123	2B	32	37	31	32	33	20	00	00	69	F3

V naší ilustraci jsou všechny řetězce reprezentující konstanty zakončeny mezerou (kód 20_H), v textu zdrojového programu tomu tak samozřejmě být nemusí. Obecně je číselná konstanta ve zdrojovém programu zakončena *omezovačem* (může to být např. mezera, aritmetický operátor nebo i jiné znaky).

Začněme otázkou: lze problém vstupní konverze vůbec řešit konečným automatem? Víme, že KA si všechnu podstatnou informaci o dosud zpracované části vstupního řetězce „pamatuje“ prostřednictvím svého stavu. Uvědomíme-li si, že v počítači můžeme ve formátu INTEGER zobrazit celá čísla pouze z konečného intervalu <MININT, MAXINT> (v případě zobrazení na 4 bytech je to z intervalu <-2.147.483.648, 2.147.483.647>), je zřejmé, že konečný počet stavů k řešení této úlohy stačí (v našem případě jich musí být přinejmenším 2^{32}). Jakým způsobem ale takový automat navrhnout, jak nakreslit jeho přechodový graf?
Než přejdeme k vlastní vstupní konverzi, vyřešíme jednodušší (nicméně s naším problémem související) úlohu: navrhneme rozpoznávací konečný automat, který bude rozpoznávat ře-

tězce představující syntakticky správně zapsané konstanty typu INTEGER. Přechodový graf tohoto automatu bude vypadat takto:



Interpretace symbolů vstupní abecedy:

d …….. libovolná dekadická číslice
u …….. omezovač (ukončovací znak) vstupního řetězce
+ , - . standardní matematické operátory

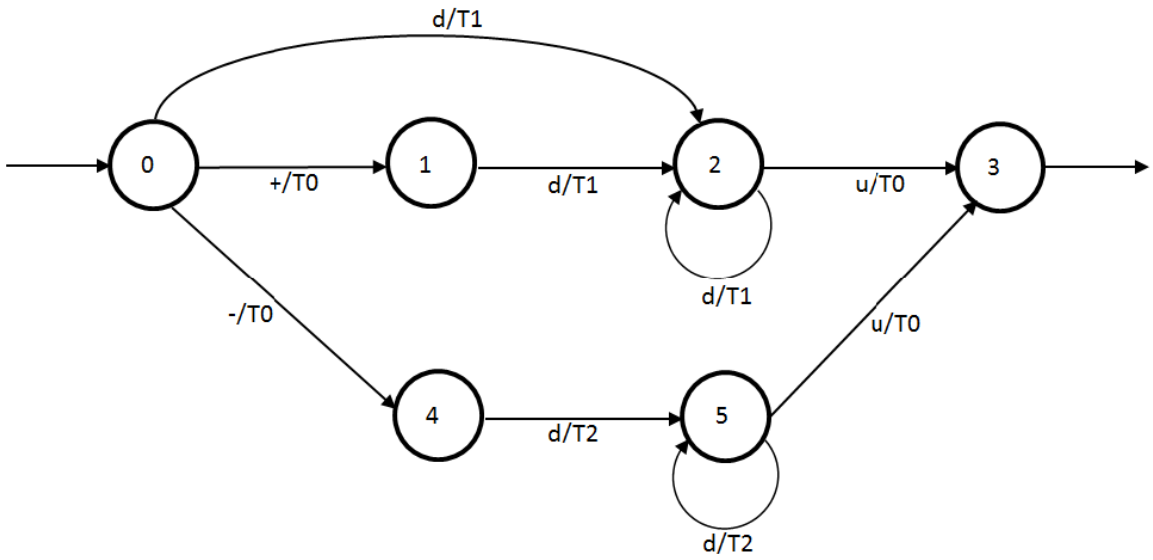
Přijde-li v některém stavu vstupní symbol, pro který v přechodovém grafu neexistuje hrana, znamená to, že řetězec nepředstavuje korektní zápis celočíselné konstanty a je automatem zamítnut. Je zřejmé, že je automat převeden z počátečního do koncového stavu tehdy a jen tehdy, pokud řetězec v textovém poli představuje korektně zapsanou celočíselnou konstantu.

K popisu automatu, který bude provádět vstupní konverzi, můžeme využít přechodový graf rozpoznávacího automatu. Musíme si ale uvědomit, že aby si KA provádějící konverzi mohl „pamatovat“ hodnotu dosud zpracované části vstupního řetězce, může jednomu stavu rozpoznávacího automatu odpovídat větší počet stavů navrhovaného konverzního automatu. Je zřejmé, že počátečnímu stavu 0 bude odpovídat jediný počáteční stav, stavu 1 budou odpovídat dva stavy konverzního automatu (jeden pro konstantu s kladným znaménkem, druhý se záporným), stavům 2 a 3 bude odpovídat vždy 2^{32} stavů konverzního automatu. Jedno kolečko v přechodovém grafu tedy může představovat více stavů „srovnaných za sebou ve třetím rozměru dvourozměrné nákresny“. Nemůžeme je tedy už nazývat stavem, ale budeme mu říkat *makrostav*. Stavy tvořící jeden makrostav mezi sebou budeme rozlišovat pomocí hodnot *pomocných stavových proměnných*. Je tedy zřejmé, že při přechodu mezi makrostavy je třeba definovat také to, jak se budou transformovat hodnoty pomocných stavových proměnných, stejně tak je nutné definovat i počáteční hodnoty pomocných stavových proměnných (jsou „součástí stavu“).

Pro srozumitelnost popisu i jednodušší implementaci je výhodné, pokud algoritmus transformace pomocných stavových proměnných závisí pouze na vstupním symbolu a makrostavu a nezávisí na hodnotě pomocných stavových proměnných samých (jinak řečeno – bez ohledu na konkrétní hodnoty pomocných stavových proměnných je pro každou dvojici *(makrostav, vstupní symbol)* přechod jednoznačný a transformace pomocných stavových proměnných se provádí stejným algoritmem).

K vytvoření vnitřní reprezentace ve formátu INTEGER použijeme postup založený na Hornerově schématu pro výpočet hodnoty mnohočlenu. Koeficienty mnohočlenu jsou číslice ve znakovém poli, mnohočlen vyčíslíme pro x = 10. Neformální připomenutí elementárních znalostí z matematiky: „dosud vypočtenou hodnotu vynásobíme deseti a přičteme (resp. v případě záporného čísla odečteme) další číslici; začínáme s počáteční hodnotou 0“.

Je zřejmé, že v tomto případě nelze přímo použít výše uvedený přechodový graf rozpoznávacího automatu, protože aktuální číslici zpracováváme jinak, představuje-li zpracovávaný řetězec konstantu kladnou, jinak, zpracováváme-li konstantu zápornou. Proto zpracování kladného a záporného znaménka oddělíme:



„Stavy“ 2, 3 a 5 je nutné chápat jako makrostavy, tedy jako množiny stavů. V rámci jednoho makrostavu budou jednotlivé stavy rozlišeny hodnotou pomocné stavové proměnné *hodnota* . Proměnné *hodnota* bude typu INTEGER a její počáteční hodnota bude 0. Symboly T0, T1, T2 u přechodů za lomítkem budeme obecně chápat jako nějaké transformace pomocných stavových proměnných, které budou realizovány v souvislosti s přechodem mezi makrostavy. V našem příkladu budou transformace vypadat takto:

T0 : prázdná;
T1 : *hodnota* := *hodnota**10 + (*VSTUP* – ord(“0“));
T2 : *hodnota* := *hodnota**10 - (*VSTUP* – ord(“0“));

VSTUP budeme chápat jako aktuálně zpracovávaný znak ze vstupního textového pole (je v kódu ASCII, proto je třeba od kódové reprezentace číslice odečíst kódovou reprezentaci číslice 0).

Na úrovni tohoto modelu není řešeno, zda syntakticky korektní řetězec reprezentuje konstantu z povoleného rozsahu <MININT, MAXINT>, to je nutné zajistit při implementaci vhodným ošetřením přetečení při instrukcích aritmetických operací.

Idea programátorského řešení vstupní konverze na základě KA modelu je nastíněna v odstavci 1.6 .

Pozorný čtenář si v průběhu studia tohoto odstavce mohl uvědomit (zdánlivý) rozpor mezi zavedením pomocných stavových proměnných na jedné straně a tvrzením „Konečný automat nemá k dispozici žádnou jinou paměť, vše potřebné si pamatuje prostřednictvím svého stavu" z odstavce 1.2.1. Co je tedy při softwarové implementaci množinou stavů Q ve smyslu matematické definice KA ze zmíněného odstavce?

Obecně je množina stavů Q podmnožinou kartézského součinu množiny makrostavů a množin možných hodnot všech pomocných stavových proměnných:

$Q \subseteq Q_m \times I_{p1} \times I_{p2} \times ..... \times I_{pm}$, kde

Q_m představuje množinu všech makrostavů (tj. množinu všech „koleček" v přechodovém grafu),

I_{pi} představuje množinu všech hodnot, kterých může nabývat i-tá pomocná stavová proměnná (je to dáno jejích typem).

V našem příkladu má Q_m 6 prvků, pomocná proměnná *hodnota* může nabývat 2^{32} různých hodnot (INTEGER na 4 bytech), kartézský součin $Q_p = Q_m \times I_{hodnota}$ bude mít $6*2^{32}$ prvků, ovšem je zřejmé, že každý z makrostavů 0, 1 a 4 představuje jediný stav, takže množina stavů Q našeho automatu bude mít celkem $3 + 3*2^{32}$ prvků.

1.5.4. Lexikální analyzátor překladače programovacího jazyka

Lexikální analyzátor (dále jen LA) je část překladače, která má za úkol formovat ze vstupního zdrojového textu *lexémy*, tj. základní prvky příslušného jazyka, jako jsou např. klíčová slova *(begin, while, if, ….)*, identifikátory, konstanty nebo operátory (*, +, <=, ….). Zformované identifikátory LA ukládá do *tabulky symbolů*. Lexémy jsou pak reprezentovány ve formě *tokenů*, jež jsou dále poskytovány ke zpracování *syntaktickému analyzátoru*. Token se skládá ze dvou částí. První je jeho identifikace (tj. o jaký typ lexémy jde), druhá je jeho hodnota (např. u identifikátoru odkaz do tabulky symbolů) . Dalším úkolem LA je odstranění komentářů a nepodstatných znaků *(whitespace)* ze zdrojového programu. Vzhledem k tomu, že symboly programovacího jazyka bývají popsány regulární gramatikou, lze při konstrukci LA využít principy konečného automatu.

Principy funkce LA demonstruje (formálně ne zcela korektní) obrázek na následující straně, kombinující prvky přechodového grafu a vývojových diagramů (inspirace Gries). Před jeho studiem je nutné si uvědomit následující:

- LA bude implementován jako procedura, která bude volána ze syntaktického analyzátoru za účelem zformování jedné lexémy; návratovou hodnotou volání bude token, který syntaktický analyzátor zpracuje a poté opět vyvolá LA za účelem zformování další lexémy,
- v rámci zjednodušení obrázku
    - uvažujeme pouze celočíselné konstanty;

    - uvažujeme pouze tři dvouznakové operátory a omezovače, a to operátor // a omezovače komentáře /* a */ ; všechny ostatní operátory jsou jednoznakové,
    - předpokládáme, že všechna klíčová slova (a také direktivy překladače) začínají písmenem.
- vstupními symboly automatu jsou symboly typu **D** (reprezentuje jakoukoli dekadickou číslici), symboly typu **L** (malá i velká písmena a další nečíselné symboly, které se mohou vyskytovat v identifikátorech), znak **/** ; symbol **other** představuje jakýkoli jiný symbol, pro který z daného stavu nevede konkrétní výstupní hrana,
- vstupní symboly jsou v grafu zapsány **standardním tučným písmem,**
- akce prováděné nad globálními proměnnými překladače jsou psány *KURZÍVOU,*
- k popisu transformace proměnných jsou použity i rozhodovací bloky pro větvení algoritmu transformace v závislosti na splnění/nesplnění definované podmínky,
- při návratu z procedury se jako návratová hodnota vrací token ve tvaru *(typ lexémy, kód lexémy v rámci typu),* případně indikace chyby,
- symboly uvozené znakem $ představují kódové reprezentace typů tokenů a konkrétních operátorů / a //; všechny ostatní jednoznakové operátory a identifikátory jsou zakódovány svým pořadím v tabulce TD, resp. TI (viz níže).

Popis jednotlivých stavů:

S …….. počáteční stav formování lexémy

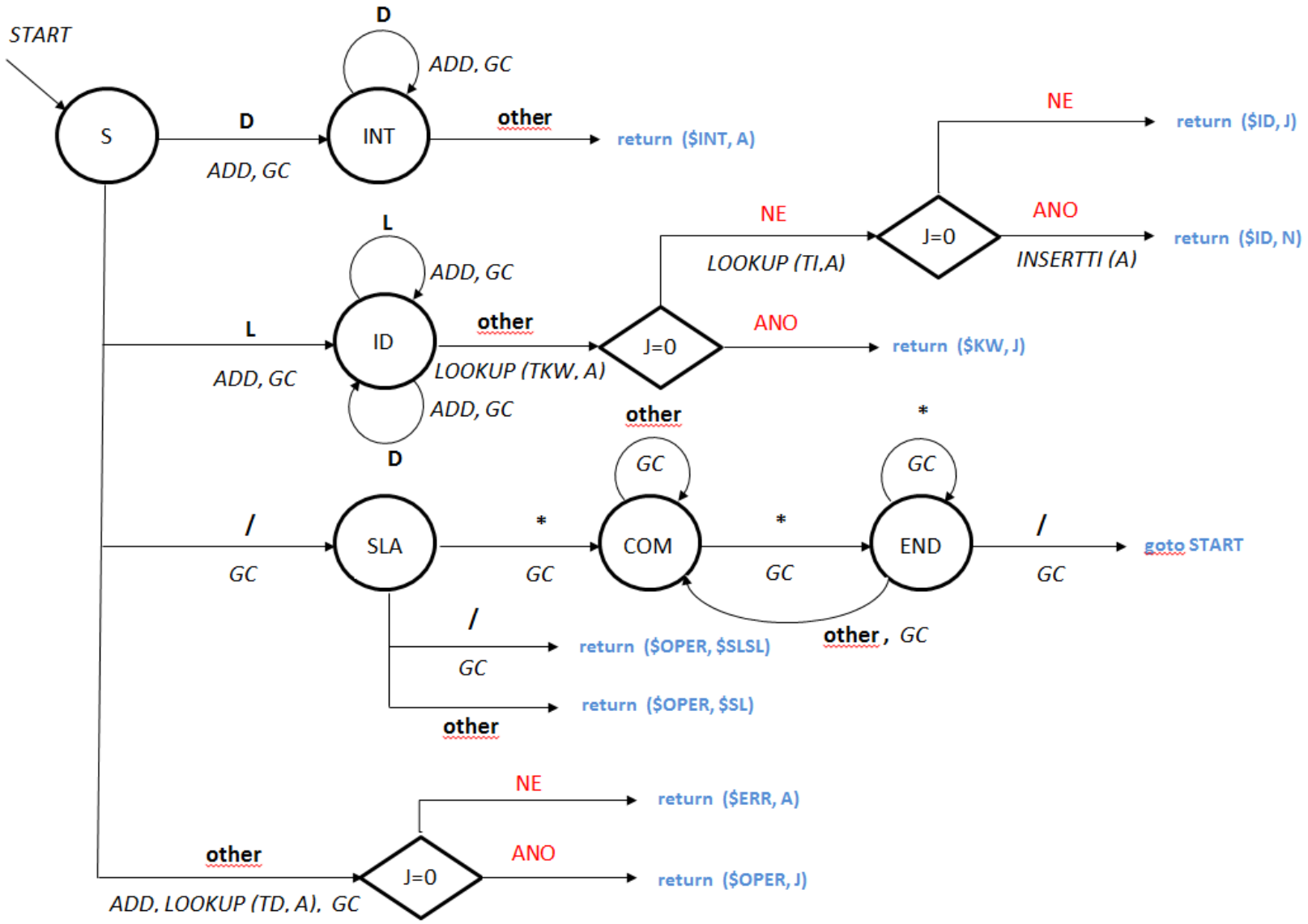
INT ….. stav, v němž je formována lexéma reprezentující celočíselnou konstanta bez znaménka

ID ….. stav, v němž je formována lexéma klíčové slovo nebo identifikátor

SLA.... stav, v němž je prvním a posledním zpracovaným znakem lexémy znak /

COM stav, v němž je vypouštěn komentář a posledním zpracovaným znakem nebyl znak *

END stav, v němž je zpracováván komentář a posledním zpracovaným znakem byl znak *


START
S
D
INT
D
ADD, GC
ADD, GC
other
return ($INT, A)
L
ID
L
ADD, GC
ADD, GC
other
LOOKUP (TKW, A)
ADD, GC
D
J=0
NE
LOOKUP (TI,A)
ANO
INSERTTI (A)
NE
return ($ID, J)
ANO
return ($ID, N)
J=0
ANO
return ($KW, J)
other
/
SLA
GC
*
COM
GC
*
END
GC
/
GC
other
*
GC
/
GC
goto START
/
return ($OPER, $SLSL)
other
return ($OPER, $SL)
other , GC
other
J=0
NE
return ($ERR, A)
ANO
return ($OPER, J)
ADD, LOOKUP (TD, A), GC

,

Použité proměnné:

- v proměnné CHAR je vždy aktuálně zpracovávaný znak zdrojového programu; před prvním voláním LA je jí přiřazen první znak zdrojového programu,
- v proměnné CLASS je vždy typ aktuálně zpracovávaného znaku; je nastavována současně s proměnnou CHAR; podle obsahu proměnné CLASS automat vykonává přechody,
- v proměnné A je postupně formován řetězec znaků**,** tvořící lexému; při každém volání LA je proměnná A inicializována prázdným řetězcem,
- tabulka klíčových slov jazyka TKW; v průběhu překladu se nemění*,*
- tabulka použitých identifikátorů TI; na začátku překladu je prázdná, v průběhu překladu do ní LA vkládá nově se vyskytnuvší identifikátory *,*
- tabulka jednoznakových omezovačů TD; v průběhu překladu se nemění (tato tabulka není nezbytně nutná, jejím smyslem je umožnit rozpoznat a zakódovat všechny jednoznakové omezovače, lze to realizovat i jinak),
- proměnná J je používána jako index a výstup při hledání řetězce v tabulkách TKW, TI, TD,
- proměnná N obsahuje aktuální počet záznamů v tabulce identifikátorů TI; před prvním voláním LA je jí přiřazena hodnota 0.

Použité procedury:

- procedura *GC* má za úkol přečíst další znak zdrojového programu, přiřadit ho proměnné CHAR a jeho typ proměnné CLASS; popis typu symbolů **D** a **L** vyz výše, všechny ostatní symboly lze chápat jako individuality; tato procedura bude řešit i odstranění nevýznamných znaků *(whitespace),*
- procedura *ADD* k řetězci obsaženému v proměnné A „přiřetězí“ obsah proměnné CHAR,
- procedura LOOKUP má dva parametry – tabulku řetězců (tj. klíčových slov, identifikátorů nebo jednoznakových omezovačů) a hledaný řetězec; zjistí, zda je řetězec v tabulce obsažen (v tom případě vrátí jeho pozici v proměnné J, v opačném případě vrátí v proměnné J hodnotu 0)*,*
- procedura INSERTTI má jediný parametr – řetězec (nově nalezený identifikátor), který má být vložen do tabulky identifikátorů TI; procedura jej vloží na konec tabulky a proměnnou N zvýší o 1.

Je zřejmé, že pokud bychom uvažovali i konstanty typu REAL, zkomplikovala by se „první vodorovná větev“ obrázku. Pokud bychom připustili i další dvouznakové operátory, jako např. <= nebo ++ , museli bychom vytvořit další „vodorovné větve“ a zpracovávat první symbol dvojznaku podobně jako ve stávající větvi pro lexém // .

1.6. Principy softwarové implementace konečných automatů

V této kapitole naznačíme jednu z možností, jak lze efektivně softwarově realizovat i složité algoritmy založené na konečněautomatových modelech. Je použit model Mealyho automatu, který umožňuje generovat výstupní akce v souvislosti s přechody automatu.

Použité datové typy:

STAV ..... výčtový typ, který slouží k pojmenování makrostavů; pokud je pojmenování pouze číslené, může být použit i typ INTEGER,
VSTUP..... výčtový typ (principielně podmnožina typu CHAR),
TYP ..... výčtový typ; použije se v případě, kdy automat zpracovává více znaků stejným způsobem (např. jako typy typy **D** a **L** v lexikálním analyzátoru v 1.5.4).

Použité proměnné:

stav ..... proměnná typu STAV,
prech_tab ..... dvourozměrné pole typu STAV; první index je typu STAV, druhý index typu VSTUP (nebo typu TYP); reprezentuje přechodovou tabulku,
transf_tab ..... dvourozměrné pole typu INTEGER; první index je typu STAV, druhý index typu VSTUP (nebo typu TYP); jsou v ní uloženy identifikační kódy transformací pomocných stavových proměnných,
vyst_tab ..... dvourozměrné pole typu INTEGER; první index je typu STAV, druhý index typu VSTUP (nebo typu TYP); reprezentuje výstupní tabulku, jsou v ní uloženy identifikační kódy výstupních akcí, které automat při konkrétním přechodu mezi makrostavy vykoná.

Použité procedury:

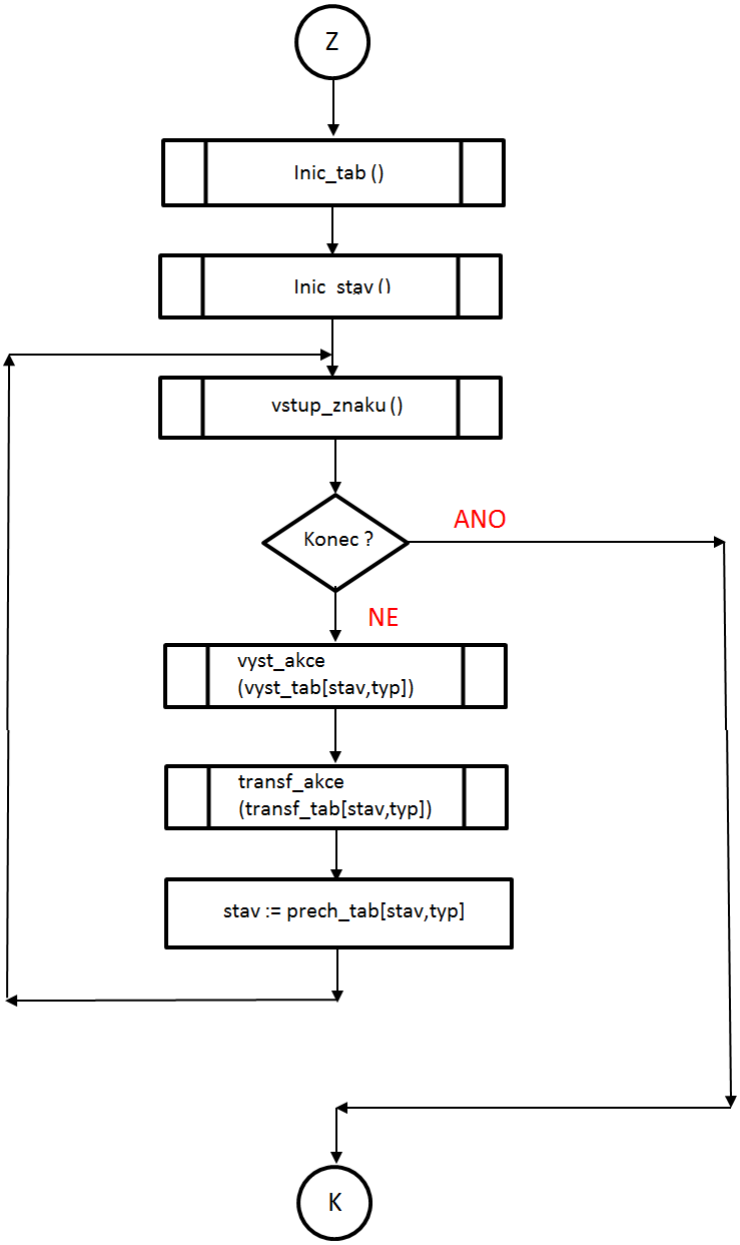
Inic_tab ..... bez parametrů; provádí naplnění polí prech_tab, transf_tab a vyst_tab,
Inic_stav ..... bez parametrů; provádí inicializaci proměnné stav a pomocných stavových proměnných,
vstup_znaku ..... bez parametrů; provádí vstup jednoho znaku, uloží jej do proměnné vstup a odpovídajícím způsobem nastaví proměnnou typ; vstupním znakem může být i ukončovací znak, na základě kterého program skončí,
transf_akce ..... parametrem je identifikační kód transformace; provádí transformaci pomocných stavových proměnných podle předaného kódu,
vyst_akce .....parametrem je identifikační kód výstupní akce, kterou procedura provede.

V procedurách nesmí existovat lokální proměnné, které by si uchovávaly svoji hodnotu mezi dvěma voláními procedury !! Jinak řečeno - pokud je potřeba použít v procedurách lokální proměnné, musí být při každém vstupu do procedury opětovně inicializovány. Důvod je zřejmý: roli stavu konečného automatu při tomto způsobu implementace hraje proměnná

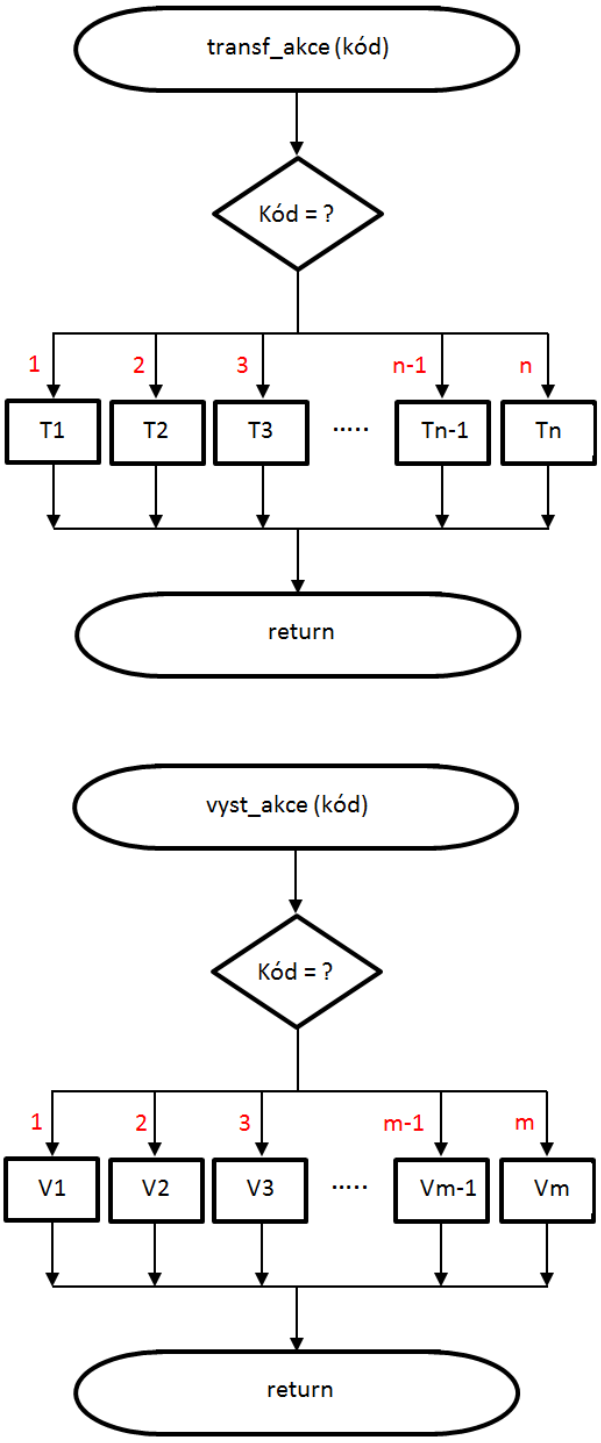
stav a pomocné stavové proměnné, jakékoli jiné „uchovávání stavu" v proměnných je v implementovaném modelu nekorektní.

Žádná procedura nesmí měnit hodnotu proměnné stav, pouze procedura transf_akce smí měnit hodnoty pomocných stavových proměnných.

Struktura hlavního programu:



Struktura procedur transf_akce a vyst_akce:



Bloky T1 až Tn realizují transformace pomocných stavových proměnných při přechodech mezi makrostavy (viz příklad v odstavci 1.5.3). Bloky V1 až Vm realizují výstupní akce do okolí automatu, prováděné při přechodech mezi makrostavy (akce mohou být mnohem obecnější, než jen výstup jednoho znaku z výstupní abecedy) .

V tomto odstavci byl popsán princip implementace KA Mealyho typu. S drobnými úpravami je možné stejným způsobem implementovat i automat Moorova typu, který umožňuje generovat výstupní akce na základě stavů, v nichž se automat v průběhu výpočtu vyskytuje:

- vyst_tab bude pouze jednorozměrné pole typu INTEGER; index bude typu STAV,
- ve vývojovém diagramu hlavního programu přibude bezprostředně za blokem s voláním procedury inic() blok s voláním procedury vyst_akce(vyst_tab[stav]).

1.7. Principy hardwarové realizace konečných automatů

V této kapitole naznačíme, jakým způsobem lze realizovat *sekvenční logický obvod* popsaný konečným automatem. Při návrhu využijeme *synchronní klopné obvody* a *hradla*. Tyto obvody jsou schopny pracovat s elektrickými signály dvou úrovní (budeme je reprezentovat jako 0 a 1).

Hradla jsou obecně vícevstupová (některá mohou mít až 8 vstupů) a nad vstupy realizují nějakou logickou operaci (např. logický součin AND, negaci logického součinu NAND, logický součet OR, negaci logického součtu NOR, ...), jejíž výsledek je výstupem hradla. Chování hradel lze popsat příslušnou rovnicí , např. pro čtyřvstupové hradlo NAND rovnicí $o = \neg(i_1 \wedge i_2 \wedge i_3 \wedge i_4)$.

Vhodným spojováním hradel lze vytvářet *kombinační logické obvody*, které realizují tzv. *logické funkce*. Výstup kombinačních obvodů závisí pouze na okamžitých kombinacích vstupních proměnných. Kombinační obvody tedy „nemají paměť" předchozí historie vstupů, jedné kombinaci vstupních proměnných proto odpovídá vždy stejná výstupní hodnota. Obecně lze funkci kombinačního obvodu) popsat rovnicí $o^{(i)} = f(i_1^{(i)}, i_2^{(i)}, i_3^{(i)}, ..., i_n^{(i)})$ (všechny hodnoty jsou vztaženy ke stejnému diskrétnímu časovému okamžiku i).

Pokud má kombinační obvod více výstupů (tj. realizuje vektorovou logickou funkci), používá se pro něj někdy název *kombinační síť*. Kombinační síť lze popsat soustavou rovnic

$$
\begin{aligned}
o_1^{(i)} &= f_1(i_1^{(i)}, i_2^{(i)}, i_3^{(i)}, ..., i_n^{(i)}) \\
o_2^{(i)} &= f_2(i_1^{(i)}, i_2^{(i)}, i_3^{(i)}, ..., i_n^{(i)}) \\
&..... \\
o_m^{(i)} &= f_m(i_1^{(i)}, i_2^{(i)}, i_3^{(i)}, ..., i_n^{(i)}),
\end{aligned}
$$

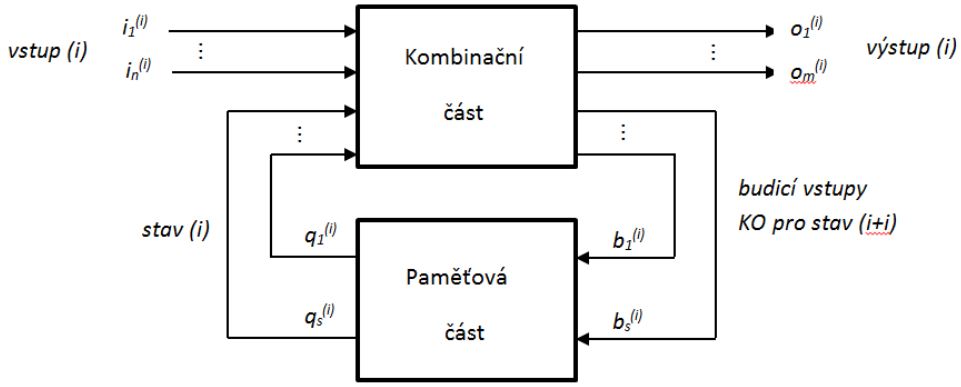
kde *n* je počet vstupních dvouhodnotových proměnných a *m* počet výstupních dvouhodnotových proměnných.

Synchronní klopné obvody (dále KO) slouží jako paměťové prvky, mají tedy svůj vnitřní stav, který může nabývat hodnoty 0 nebo 1. KO má podle typu jeden nebo dva vstupy, které určují, jakým způsobem se změní aktuální stav KO. Stav KO se může změnit jen v definovaných okamžicích hodinových pulzů, kterými jsou synchronní klopné obvody řízeny. Funkci klopné-

ho obvodu lze tedy obecně popsat funkcí $q^{(i+1)} = f(i^{(i)}, q^{(i)})$. Následující stav KO (tj. „to, co si KO pamatuje") je tedy funkcí aktuálních hodnot vstupu a stavu. Výstupem KO je jeho stav v daném časovém okamžiku, tedy $o^{(i)} = q^{(i)}$. Běžný KO kromě výstupu (tj. aktuálního stavu) poskytuje i jeho negaci $\neg q^{(i)}$.

Sekvenční logické obvody jsou obvody, u kterých výstupní hodnoty závisejí nejen na aktuální kombinaci hodnot vstupů, ale také na jejich historii, která je uchovávána jako stav obvodu (na rozdíl od kombinačních obvodů tedy sekvenční obvody „mají paměť").

Obecnou strukturu sekvenčního obvodu znázorňuje následující obrázek:



Význam indexů v obrázku: *n* počet dvoustavových vstupů, *m* počet dvoustavových výstupů, *s* počet klopných obvodů v paměťové části.

Kombinační část realizuje logické funkce pro výpočet hodnot *m* výstupů a *s* (případně 2*s*) budicích signálů KO (podle typu použitých KO). Reálně do ní vstupuje *n* vstupů a *s* dvojic (stav, negace stavu) z výstupů KO. Paměťovou část tvoří *s* klopných obvodů.

Pro ilustraci uvedeme tabulky popisující funkci klopného obvodu typu J-K, který má dva budicí vstupy (J, K):

Q_i	J	K	Q_{i+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

J	K	Q_{i+1}	Tato kombinace J K
0	0	Q_i	stav nemění
0	1	0	nastaví $Q_{i+1} = 0$
1	0	1	nastaví $Q_{i+1} = 1$
1	1	negace Q_i	stav neguje

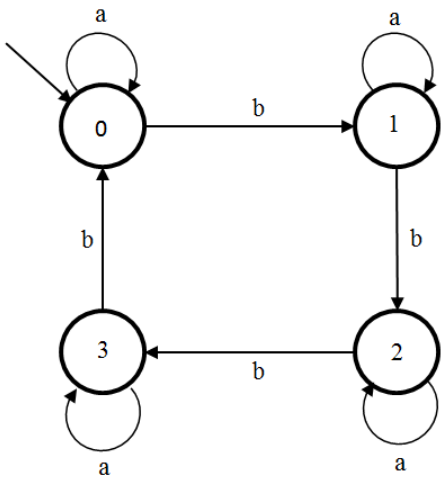
(pravá tabulka je pouze komprimací tabulky levé, neobsahuje žádnou informaci navíc)

Z výše uvedených tabulek můžeme odvodit, jaké hodnoty budících vstupů J a K potřebujeme k dosažení konkrétní změny stavu (znak – znamená, že na hodnotě tohoto vstupu nezáleží):

$Q_i \rightarrow Q_{i+1}$	J	K	$Q_i \rightarrow Q_{i+1}$	J	K
$0 \rightarrow 0$	0	-	$1 \rightarrow 0$	-	1
$0 \rightarrow 1$	1	-	$1 \rightarrow 1$	-	0

Nezbytným krokem pří hardwarové realizaci KA je kódování stavů a vstupních symbolů. Vzhledem k tomu, že výše popsané logické obvody pracují pouze se dvěma úrovněmi signálu (0 a 1), je třeba stavy KA reprezentovat pomocí kombinací stavů většího počtu klopných obvodů, stejně tak symboly vstupní (výstupní) abecedy budou kódovány kombinacemi většího počtu dvouhodnotových logických vstupů (výstupů). Je-li *n* počet prvků množiny stavů (resp. vstupních nebo výstupních symbolů), potřebujeme k realizaci $s = log_2\, n$ klopných obvodů (resp. dvouhodnotových vstupů nebo výstupů); v případě, že *m* není celé číslo, pak samozřejmě nejbližší vyšší celé číslo.

Příklad: S využitím klopných obvodů typu J-K a hradel AND navrhněte sekvenční logický obvod popsaný automatem Moorova typu s následujícím přechodovým grafem a výstupní funkcí:



$$\lambda(0) = z$$
$$\lambda(1) = y$$
$$\lambda(2) = y$$
$$\lambda(3) = z$$

Řešení:

Kódování stavů, vstupních a výstupních symbolů:

K zakódování čtyř stavů jsou zapotřebí dva klopné obvody KO2 a KO1, jejich stavy označíme jako Q^2 a Q^1. K zakódování dvou vstupních symbolů a dvou výstupních symbolů stačí jeden

dvoustavový vstup a jeden dvoustavový výstup. Zvolené kódování reprezentují následující tabulky.

vstup	X
a	0
b	1

výstup	Y
y	0
z	1

stav	Q^2	Q^1	Y
0	0	0	1
1	0	1	0
2	1	1	0
3	1	0	1

Z tabulky stavů je zřejmé, že výstup Y je roven negaci stavu KO1 $Y = \neg Q^1$, což ulehčuje generování výstupu (výstup bude přímo výstupem klopného obvodu KO1).

Zakódovaná přechodová tabulka KA + tabulka budicích funkcí klopných obvodů:

Vstup	Stav aktuální		Stav následující		Změny stavů KO		Budicí vstupy KO			
X	Q^2_i	Q^1_i	Q^2_{i+1}	Q^1_{i+1}	$Q^2_i \rightarrow Q^2_{i+1}$	$Q^1_i \rightarrow Q^1_{i+1}$	J_2	K_2	J_1	K_1
0	0	0	0	0	0 → 0	0 → 0	0	-	0	-
1	0	0	0	1	0 → 0	0 → 1	0	-	1	-
0	0	1	0	1	0 → 0	1 → 1	0	-	-	0
1	0	1	1	1	0 → 1	1 → 1	1	-	-	0
0	1	0	1	0	1 → 1	0 → 0	-	0	0	-
1	1	0	0	0	1 → 0	0 → 0	-	1	0	-
0	1	1	1	1	1 → 1	1 → 1	-	0	-	0
1	1	1	1	0	1 → 1	1 → 0	-	0	-	1

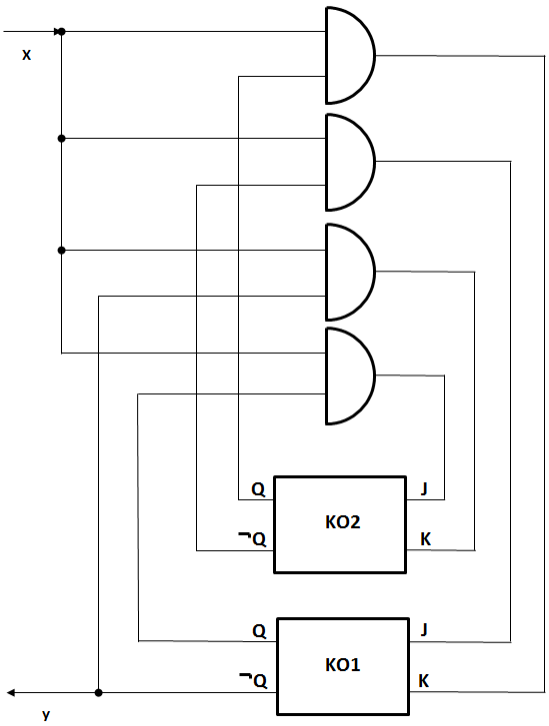
Po vhodném dodefinování neurčených položek v tabulce budicích signálů (to je výsledkem tzv. *minimalizace logických funkcí*) vyjádříme budící funkce v součinovém tvaru:

$$J_2 = X \wedge Q^1, \quad K_2 = X \wedge \neg Q^1, \quad J_1 = X \wedge \neg Q^2, \quad K_1 = X \wedge Q^2$$

Je zřejmé, že v našem případě k sestavení kombinační části sekvenčního obvodu stačí čtyři dvouvstupová hradla AND (pro každý budicí signál jedno), pro generování výstupu již žádná další hradla nejsou potřeba (lze použít přímo výstup z klopného obvodu).

Při realizaci s „historickými“ integrovanými obvody bipolární technologie TTL by byly v zapojení použity dva integrované obvody: 1ks IO 7408 (4x dvouvstupové hradlo AND) a 1 ks IO 74107 (2x J-K Flip-Flop s asynchronním nulováním).

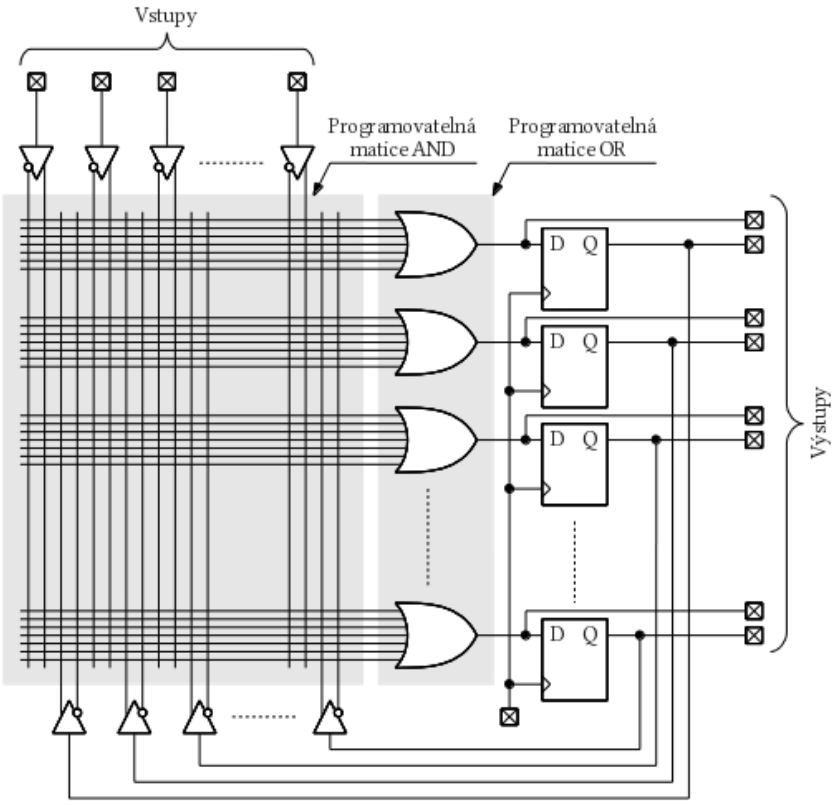
Následující obrázek znázorňuje schéma zapojení „logické části“ (nejsou nakresleny obvody související s generováním hodinových pulzů a s napájením), u KO nejsou znázorněny vstupy hodinových signálů a vstupy pro asynchronní nulování (pro převedení obvodu do počátečního stavu).



Efektivním nástrojem pro realizaci sekvenčních logických obvodů jsou uživatelsky programovatelné logické obvody PLD *(Programmable Logical Device)*. Tyto obvody vycházejí z faktu, že každou logickou funkci lze vyjádřit v disjunktivní formě, tedy jako „logický součet logických součinů“ (Sum Of Products – SOP).

PLD obsahuje řádově tisíce součtových hradel, součinových hradel a klopných obvodů v pevné struktuře, navzájem propojených dvěma programovatelnými propojovacími maticemi AND a OR. Matice AND umožňuje z naprogramováním vybraných vstupů (resp. jejich negací) a stavů klopných obvodů (resp. jejich negací) vytvářet jejich logické součiny - *termy* . Matice OR umožňuje z (naprogramováním vybraných) termů vytvářet jejich logické součty. Tyto součty reprezentují budicí funkce klopných obvodů (obvykle typu D s jedním budicím vstupem) nebo přímo výstupy. Součinové termy mohou být současně využity pro více vytvářených součtů.

Vnitřní strukturu PLA znázorňuje obrázek:



Každá vodorovná čára v programovatelné matici AND představuje vždy jedno součinové hradlo. Průsečík vodorovné a svislé čáry představuje „programovatelný spoj“ (je/není propojeno). Na vstupy každého součinového hradla lze tedy připojit libovolnou kombinaci vstupních signálů, zpětných vazeb z výstupů klopných obvodů a jejich negací. Stejným způsobem je programovatelná i matice OR (v obrázku už není detailně rozkreslena), na vstupy každého součtového hradla lze tedy přivést libovolnou kombinaci termů.

V současné době mají ze všech uživatelem programovatelných obvodů nejobecnější strukturu a největší množství využitelných logických prostředků obvody označované jako FPGA *(Field Programmable Gate Array)*. Jsou v nich integrovány řádově milióny hradel.

Nezbytným nástrojem pro práci s těmito programovatelnými obvody jsou návrhové systémy. Popis navrhované konstrukce bývá nejčastěji textový – pomocí speciálních jazyků pro popis hardware (např. ABEL nebo VHDL) nebo grafický (používají se editory schémat nebo stavových diagramů).

2. Regulární jazyky

2.1. Formální jazyk versus přirozený jazyk

Lidská řeč je stará jako lidstvo samo a vyvíjí se spojitě s rozvojem společnosti. Přirozený jazyk slouží k vzájemné komunikaci mezi lidmi a lze jej popsat dvěma základními strukturami. První z nich je slovník, tedy množina slov, které mají v daném jazyce nějaký smysl. Druhou součástí je gramatika, tedy soubor pravidel, který určuje, jakým způsobem lze v daném jazyce ze slov vytvářet vyšší celky (věty). Přirozený jazyk je schopen vyjádřit emoce, pocity, myšlenky. Přirozené jazyky jsou víceznačné, používají metafory a idiomy (tj. dokáží přenést význam původního pojmenování na jiný objekt či jev), vykazují nepravidelnosti a výjimky z obecných pravidel. Z toho vyplývají obtíže související s hledáním jednoznačného formálního popisu přirozeného jazyka, s jeho následným strojovým zpracováním a přiřazením významu.

Formální jazyky jsou oproti tomu jazyky, které uměle vytvořil člověk pro nějaký specifický účel. Matematika používá svůj formální jazyk (formální logiku) k vyjádření vztahů mezi čísly, matematickými objekty a symboly. Programovací jazyky jsou formální jazyky, které byly vytvořeny jako nástroj pro jednoznačný popis algoritmů, respektive úkonů, jež má vykonat počítač. Existují specifikační jazyky navržené jako nástroje pro popis, analýzu a ověřování vlastností různých typů systémů (např. komunikačních protokolů, částí hardware apod).

Formální jazyk budeme v prvním přiblížení chápat jako množinu řetězců vytvořených z prvků nějaké množiny znaků (abecedy). Později uvidíme, že i formální jazyk lze popsat nástrojem, který bude nazván gramatikou.

Regulární jazyky jsou specifickou třídou formálních jazyků. K automatické syntaktické analýze regulárních kazyků postačují konečné automaty.

2.2. Základní pojmy

Abeceda je konečná neprázdná množina symbolů. Prvky této abecedy se nazývají *písmena*. Abecedy označujeme velkými tiskacími písmeny latinky nebo řecké abecedy.

Řetězec (slovo) nad abecedou Σ je konečná posloupnost (lépe řečeno uspořádaná n-tice) písmen $a_1, a_2, a_3, \ldots\ldots, a_n$ z abecedy Σ. Tato posloupnost se zapisuje ve tvaru $a_1 a_2 a_3 \ldots\ldots a_n$.

Délkou řetězce $a_1 a_2 a_3, \ldots\ldots a_n$ rozumíme počet jeho znaků, tedy n.

Prázdný řetězec (prázdné slovo) nad abecedou Σ je posloupnost, která neobsahuje žádné písmeno. Prázdné slovo se označuje symbolem e (v některé literatuře symbolem λ). Délka prázdného slova je 0.

Předponou řetězce $a_1, a_2, a_3, \ldots\ldots, a_n$ rozumíme libovolný z řetězců e , a_1, a_1a_2, $a_1a_2a_3$, ……, $a_1a_2a_3\ldots\ldots a_n$.

Je zřejmé, že každý řetězec má o jednu předponu více, než je jeho délka. Prázdný řetězec e má jedinou předponu e

Příklad:

Abeceda $\Sigma = \{a, b, c\}$

Příklady řetězců nad touto abecedou: $ab, acabc, b, ca, e$.

Dělky těchto řetězců: 2, 5, 1, 2, 0.

Předpony řetězce ab: e, a, ab

Předpony řetězce $acabc$: $e, a, ac, aca, acab, acabc$

Předpony řetězce b : e, b

Předpony řetězce ca : e, c, ca

Předpony řetězce e : e

Uzávěr abecedy Σ je množina všech konečných neprázdných řetězců vytvořených z písmen abecedy Σ . Uzávěr abecedy Σ se označuje symbolem Σ^+. Uzávěr abecedy je nekonečnou spočetnou množinou řetězců.

Iterace abecedy Σ je množina všech konečných řetězců vytvořených z písmen abecedy Σ. Iterace abecedy Σ se označuje symbolem Σ^*. Iterace abecedy je nekonečnou spočetnou množinou řetězců.

Zřejmě platí $\Sigma^* = \Sigma^+ \cup \{e\}$ (uzávěr a iterace se liší pouze v jediném prvku – prázdném řetězci; ten není prvkem uzávěru, ale je prvkem iterace).

Příklad:

Abeceda $A = \{0,1\}$

Uzávěr $A^+ = \{0,1,00,01,10,11,000,001,010,011,100,101,110,111,\ldots\ldots\}$

Iterace $A^* = \{e,0,1,00,01,10,11,000,001,010,011,100,101,110,111,\ldots\ldots\}$

Jazykem L *nad abecedou* Σ nazveme libovolnou množinu řetězců $L \subseteq \Sigma^*$ (tedy libovolnou podmnožinu množiny všech řetězců nad danou abecedou).

Výše uvedená definice pohlíží na jazyk jako na množinu řetězců nad danou abecedou, aniž by specifikovala jakékoli jejich další vlastnosti. Je to přístup velmi obecný, který poskytuje jen málo nástrojů pro specifikaci jazyků (vlastně jen zápis výčtem všech řetězců, které do jazyka patří, nebo popis společnou vlastností všech řetězců). Později poznáme efektivnější způsoby popisu jazyků (akceptační popis pomocí rozpoznávacího automatu a generativní popis pomocí gramatiky), nicméně ani tato „množinová“ definice jazyka není v rozporu s chápáním přirozených ani formálních jazyků.
Základní úlohou teorie jazyků je rozhodnout, zda konkrétní řetězec patří či nepatří do konkrétního jazyka. Obecný postup, kterým se tato úloha řeší, se nazývá syntaktická ana-

lýza řetězce. U přirozených jazyků je syntaktická analýza složitá, algoritmicky těžko řešitelná.

Jazyk může být konečný (obsahuje konečný počet řetězců) nebo nekonečný (obsahuje nekonečný počet řetězců). Na rozhodnutí, zda konkrétní řetězec patří do konečného jazyka, vždy stačí rozpoznávací konečný automat. U nekonečných jazyků na toto rozhodnutí konečný automat obecně nestačí, nicméně existují nekonečné jazyky, které lze konečným automatem rozpoznat. V tom případě budeme říkat, že je jazyk rozpoznáván konečným automatem.

2.3. Operace nad řetězci a jazyky

Podobným způsobem, jako matematika zavádí elementární operace nad čísly, lze zavést i operace nad řetězci. Budou to samozřejmě jiné operace než např. sečítání a násobení, nicméně z obecnějšího pohledu lze na řetězce a operace nad nimi nahlížet jako na algebraickou strukturu a zkoumat její vlastnosti.

Nechť u a v jsou řetězce nad abecedou Σ, $u = u_1u_2u_3.....u_n$ a $v = v_1v_2v_3.....v_m$, kde n a m jsou celá čísla, $n \geq 0, m \geq 0$.

Operace zřetězení (konkatenace): $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$

Binární operace zřetězení definuje ke každým dvěma řetězcům u a v jejich zřetězení takto:

$$u \cdot v = u_1u_2u_3.....u_nv_1v_2v_3.....v_m$$

Operace zřetězení je asociativní, ale není komutativní:

$(u \cdot v) \cdot w = u \cdot (v \cdot w) = u \cdot v \cdot w \qquad \forall u, v, w \in \Sigma^*$

$u \cdot v = v \cdot u$ obecně neplatí (i když můžeme nalézt konkrétní řetězce u a v, pro které vztah platí)

Prázdný řetězec je vůči řetězení neutrální prvek:

$u \cdot e = e \cdot u \qquad \forall u \in \Sigma^*$

Je zřejmé, že délka výsledku zřětezení je rovna součtu délek jeho operandů.

Operace mocnina řetězce : $\Sigma^* \times N_0 \rightarrow \Sigma^*$

Binární operace mocnina řetězce definuje pro každý řetězec u a libovolné celé nezáporné číslo n n-tou mocninu řetězce takto:

$u^n = uu^{n-1} = u^{n-1}u$
$u^0 = e$

Z výše uvedené rekurzivní definice je zřejmé, že platí $u^1 = u$, $u^2 = uu$, $u^3 = uuu$, atd.

Je zřejmá formální analogie mezi řetězením řetězců a násobením čísel a tudíž i mezi mocninou řetězce a mocninou čísla, ovšem s tím, že na rozdíl od násobení čísel není řetězení komutativní. Nad řetězci tedy lze v součtovýn výrazech provádět i úkony analogické vytýkání, ale je třeba rozlišovat vytknutí podřetězce zprava a vytknutí zleva. Analogicky lze výrazy i „roznásobovat“, ovšem opět je nutné rozlišovat zřetězení zleva a zřetězení zprava.

Operace délka řetězce : $\Sigma^* \to N_0$

Unární operace délka řetězce definuje pro každý řetězec u jeho délku, tj. celé nezáporné číslo n takto:

$$|u| = |u_1 u_2 u_3 \ldots\ldots u_n| = n$$
$$|e| = 0$$

Pro označení této operace se používá symbol používaný pro absolutní hodnotu čísla $|u|$.

Operace reverze (obrácení) řetězce : $\Sigma^* \to \Sigma^*$

Unární operace reverze (obrácení) řetězce definuje pro každý řetězec u reverzní řetězec u^R stejné délky takto:

$$u^R = u_n u_{n-1} \ldots\ldots u_2 u_1$$
$$e^R = e$$

Platí $(u^R)^R = u \qquad \forall u \in \Sigma^*$

V odstavci 2.2 jsme zavedli jazyk jako libovolnou množinu řetězců nad danou abecedou (množinový popis jazyka). Nad jazyky tedy můžeme používat všechny operace, které jsou definovány pro množiny:

Nechť L_1 a L_2 v jsou jazyky nad abecedou Σ.

Operace sjednocení jazyků

Binární operace sjednocení jazyků definuje ke každým dvěma jazykům L_1 a L_2 jejich sjednocení $L = L_1 \cup L_2$ takto:

$$L = \{w \mid w \in L_1 \vee w \in L_2\}$$

Řetězec w tedy patří do sjednocení $L_1 \cup L_2$ právě tehdy, patří-li alespoň do jednoho z jazyků L_1 a L_2.

Operace sjednocení je asociativní a komutativní.

Operace průnik jazyků

Binární operace průnik jazyků definuje ke každým dvěma jazykům L_1 a L_2 jejich průnik $L = L_1 \cap L_2$ takto:

$$L = \{w \mid w \in L_1 \wedge w \in L_2\}$$

Řetězec w tedy patří do průniku $L_1 \cap L_2$ L právě tehdy, patří-li do obou jazyků L_1 a L_2.

Operace průniku je asociativní a komutativní.

Operace rozdíl jazyků

Binární operace rozdíl jazyků definuje ke každým dvěma jazykům L_1 a L_2 jejich rozdíl $L = L_1 \setminus L_2$ takto:

$$L = \{w \mid w \in L_1 \wedge w \notin L_2\}$$

Řetězec w tedy patří do rozdílu $L_1 \setminus L_2$ právě tehdy, patří-li do jazyka L_1 a přitom nepatří do L_2.

Operace rozdílu není asociativní ani komutativní.

Operace doplněk jazyka

Unární operace doplněk jazyka definuje ke každému jazyku L jeho doplněk v Σ^* takto:

$$\overline{\overline{L}} = \{w \mid w \in \Sigma^* \wedge w \notin L\}$$

Řetězec w tedy patří do jazyka $\overline{L}$ právě tehdy, nepatří-li do jazyka L.

Všechny dosud zavedené operace byly standardní množinové operace známé ze střední školy. Vzhledem k tomu, že jazyky jsou množinami řetězců, má smysl definovat ještě další specifické operace, kterými jsou zřetězení jazyků a mocnina jazyka.

Operace zřetězení (konkatenace) jazyků

Binární operace zřetězení jazyků definuje ke každým dvěma jazykům L_1 a L_2 jejich zřetězení $L = L_1 \cdot L_2$ takto:

$$L = \{w \mid w = uv \wedge u \in L_1 \wedge w \in L_2\}$$

Řetězec w tedy patří do zřetězení $L_1 \cdot L_2$ právě tehdy, je-li zřetězením dvou řetězců, z nichž první patří do jazyka L_1 a druhý do jazyka L_2.

Operace zřetězení jazyků je asociativní a není komutativní.

Operace mocnina jazyka

Binární operace mocnina jazyka definuje pro každý jazyk L a libovolné celé nezáporné číslo n n-tou mocninu jazyka takto:

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

Z výše uvedené rekurzivní definice je zřejmé, že platí $L^1 = L$, $L^2 = L \cdot L$, $L^3 = L \cdot L \cdot L$, atd.

Je zřejmé, že s využitím operace mocnina jazyka můžeme dříve uvedené pojmy uzávěr a iterace abecedy vyjádřit takto:

$$L^+ = L \cup L^2 \cup L^3 \cup ..... = \bigcup_{i=1}^{\infty} L^i \qquad L^* = \{e\} \cup L \cup L^2 \cup L^3 \cup ..... = \bigcup_{i=0}^{\infty} L^i$$

Příklad:

Jazyk $L_1 = \{e, 00, 10\}$
Jazyk $L_2 = \{a, b, aa, bb\}$

Zřetězení $L = L_1 \cdot L_2 = \{a, b, aa, bb, 00a, 00b, 00aa, 00bb, 10a, 10b, 10aa, 10bb\}$

2.4. Příklady rozpoznávání jazyků

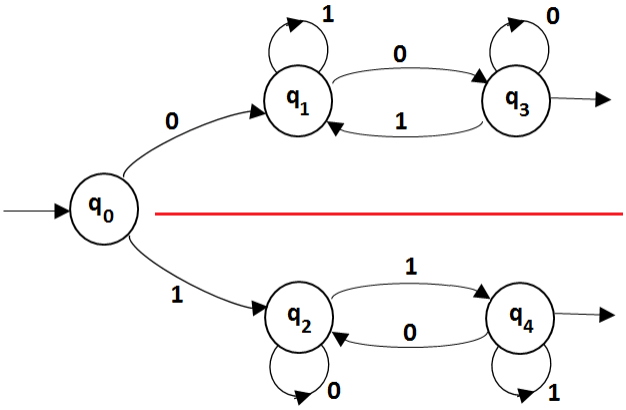
Příklad: Jazyk L je definován nad abecedou $\{0,1\}$ jako množina všech řetězců délky alespoň 2 takových, že řetězec začíná a končí stejným symbolem, tedy $L = \{w \mid w = w_1 w_2 .....w_n \wedge n \geq 2 \wedge w_1 = w_n\}$. Lze tento jazyk rozpoznávat konečným automatem?

Předpokládejme, že takový automat existuje. Jakou informaci v tom případě musí uchovávat jeho stavy? Informaci o prvním a posledním zpracovaném písmenu vstupního řetězce. V následujícím přechodovém grafu mají stavy automatu tento význam:

q_0 dosud nebyl zpracováno žádný vstupní znak
q_1 prvním zpracovaným znakem byla 0 a pokud má řetězec délku alespoň 2, byla posledním zpracovaným znakem 1
q_2 prvním zpracovaným znakem byla 1 a pokud má řetězec délku alespoň 2, byla posledním zpracovaným znakem 0
q_3 řetězec má délku alespoň 2, prvním zpracovaným znakem byla 0 a posledním zpracovaným znakem 0

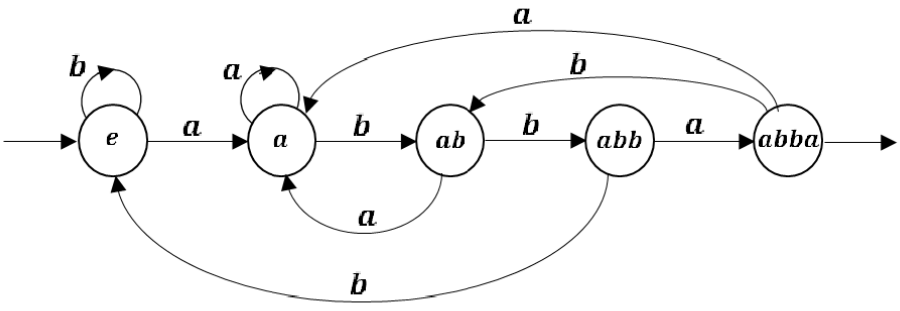
q_4 řetězec má délku alespoň 2, prvním zpracovaným znakem byla 1 a posledním zpracovaným znakem 1

Ze zadání je zřejmé, že stav *q0* bude stavem počátečním, stavy q_3 a q_4 budou stavy koncovými. Ve stavech nad červenou čárou si automat „pamatuje“ první znak 0, ve stavech pod čárou byl prvním znakem řetězce znak 1, z pochopitelných důvodů hrany grafu nemohou tuto čáru překročit.



Příklad: Jazyk L je definován nad abecedou $\{a,b\}$ jako množina všech řetězců w, které končí podřetězcem abba, tedy $L=\{w \mid w=xabba \quad \forall x \in \{a,b\}^*\}$. Je tento jazyk rozpoznáván konečným automatem?

Předpokládejme, že takový automat existuje. Jakou informaci v tom případě musí uchovávat jeho stavy? Informaci o tom, jakou předponou hledaného řetězce končí dosud zpracovaná část vstupního řetězce (jinak řečeno – jaká část z hledaného podřetězce již byla nalezena). V následujícím přechodovém grafu jsou stavy označeny předponami hledaného řetězce:

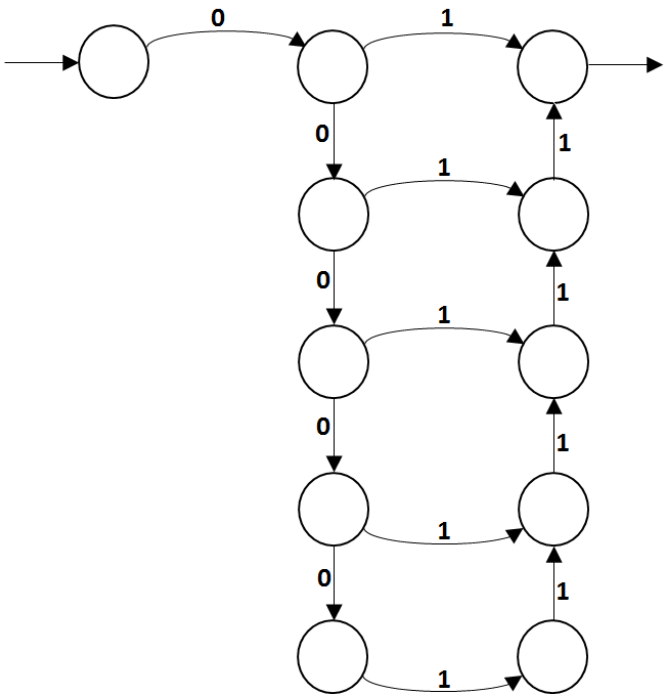


Příklad: Jazyk L je definován nad abecedou $\{0,1\}$ takto: $L=\{w \mid w=0^n1^n ; n \geq 1\}$. Lze tento jazyk rozpoznat konečným automatem?

Nejprve zkusme vytvořit několik řetězců jazyka L: $L = \{01,0011,000111,00001111,.....\}$.

Je zřejmé, že každý řetězec jazyka L tvoří určitý počet znaků 0 následovaný stejným počtem znaků 1. Za těmito znaky 1 se už žádné další znaky nevyskytují.

Předpokládejme, že konečný automat rozpoznávající tento jazyk existuje. Jakou informaci v tom případě musí uchovávat jeho stavy? Automat si musí „prostřednictvím svých stavů“ pamatovat počet znaků 0 na začátku řetězce (aby později mohl odpočítat „správný počet” znaků 1). Kolik musí mít automat stavů ? - Přinejmenším tolik, kolik nul může být na začátku řetězce (tj. n). ALE n NENÍ V ZADÁNÍ SHORA OMEZENO => počet stavů není omezen => spor s předpokladem, na řešení problému tedy konečný automat nestačí. Konečným automatem lze úlohu řešit pouze za předpokladu, že bude n shora omezeno, např. $1 \leq n \leq 5$:



Uvedený přechodový graf je neúplný (lépe řečeno zjednodušený), v některých stavech nejsou zakresleny hrany pro všechny vstupní symboly. Tyto chybějící hrany interpretujeme tak, že pokud se v takovém stavu na vstupu automatu vyskytne chybějící vstupní symbol, automat zpracovávaný řetězec zamítne.

V této kapitole jsme poznali, že jsou jazyky, které lze rozpoznat konečným automatem (tj. k danému jazyku existuje rozpoznávací konečný automat, jenž po zpracování libovolného řetězce jazyka skončí v koncovém stavu). Existují ale také jazyky, na jejichž rozpoznání konečný automat nestačí. Co o tom rozhoduje?

Zatím jsme k popisu používali společnou vlastnost řetězců, tj. množinový popis jazyka. Ukážeme si, že pro rozhodnutí o tom, zda jazyk je či není rozpoznatelný konečným automatem, je nutné specifikovat jazyk precizněji (specifikace výčtem nebo společnou vlastností nestačí).

V tuto chvíli známe dva způsoby popisu formálního jazyka:

- množinový (tedy výčtem nebo společnou vlastností)
- akceptační (popis automatem, který jazyk rozpoznává. POZOR! Automat nemusí být nutně konečný, existují i obecnější automatové modely)

Brzy poznáme ještě třetí způsob popisu, a to

- generativní (popis gramatikou, tedy jakýmisi pravidly pro vytváření řetězců)

2.5. Princip generativního popisu jazyka

Základní myšlenkou generativního popisu je popsat „správné řetězce“ (tedy řetězce, jež patří do jazyka) pomocí nějakých formalizovaných pravidel (analogie gramatických pravidel v přirozených jazycích). Princip generativního popisu ukážeme na následujícím příkladu.

Příklad: Gramatika pro generování jednoduchých vět nad omezenou množinou anglických slov. Pravidla pro odvozování řetězců jsou vyjádřena v tomto tvaru:

Pravidlo 1 :	<věta>	→	<podmětná část><přísudková část>
Pravidlo 2 :	<podmětná část>	→	<podmět>
Pravidlo 3 :	<podmětná část>	→	<přívlastek><podmět>
Pravidlo 4 :	<přísudková část>	→	<přísudek>
Pravidlo 5 :	<přísudková část>	→	<přísudek><předmět>
Pravidlo 6 :	<podmět>	→	**John**
Pravidlo 7 :	<podmět>	→	**Mary**
Pravidlo 8 :	<přívlastek>	→	**our**
Pravidlo 9 :	<přívlastek>	→	**your**
Pravidlo 10 :	<přísudek>	→	**drinks**
Pravidlo 11 :	<přísudek>	→	**buys**
Pravidlo 12 :	<předmět>	→	**tea**
Pravidlo 13 :	<předmět>	→	**gin**

Pravidla budeme nazývat *přepisovací pravidla* a budeme je číst takto: „ věta *se přepisuje* na podmětná část přísudková část“, „podmětná část *se přepisuje* na podmět“, atd.

Výše uvedená pravidla pouze formalizují běžná nejjednodušší pravidla skladby angličtiny a současně definují množinu slov, z nichž budou tvořeny věty jazyka. Je zřejmé, co jednotlivá pravidla vyjadřují:

Pravidlo 1 : „Věta se skládá z podmětné a přísudkové části.”
Pravidlo 2 a 3 : „Podmětnou část tvoří buď jen podmět nebo přívlastek s podmětem.”
.....
Pravdilo 6 a 7 : „Podmětem může být buď slovo **John** nebo slovo **Mary**.”
.....

Pravidlo 12 a 13 : „Předmětem může být buď slovo **tea** nebo slovo **gin**."

V pravidlech se vyskytují objekty dvojího typu - české názvy větných členů uvedené ve špičatých závorkách a anglická slova zvýrazněná tučným písmem. Pomocí pravidel lze provést *odvození řetězce* (*derivaci řetězce*). České názvy větných členů jsou pouze pomocné symboly, které mají význam v průběhu odvozování (takové symboly budeme později nazývat *neterminální symboly*), ale na rozdíl od anglických slov (ty budeme nazývat *terminální symboly*) se ve finálním vygenerovaném řetězci neobjeví. Při odvozování vycházíme z *počátečního symbolu* (v našem příkladu je to symbol <věta>). Postupně nahrazujeme (přepisujeme) pomocné symboly jejich rozvoji z pravých stran příslušných přepisovacích pravidel. Odvozování končí, jestliže se v řetězci vyskytují pouze anglická slova a není tam už žádný pomocný symbol.

Odvození řetězce **our Mary drinks gin** můžeme formálně vyjádřit takto (přesný význam použitého operátoru => bude vysvětlen později; číslo nad ním říká, které pravidlo bylo použito k přepsání):

<věta> $\overset{1}{\Rightarrow}$ <podmětná část> <přísudková část> $\overset{3}{\Rightarrow}$

$\Rightarrow$ <přívlastek><podmět><přísudková část> $\overset{8}{\Rightarrow}$ **our** <podmět><přísudková část> $\overset{7}{\Rightarrow}$

$\Rightarrow$ **our Mary** <přísudková část> $\overset{5}{\Rightarrow}$ **our Mary** <přísudek><předmět> $\overset{10}{\Rightarrow}$

$\Rightarrow$ **our Mary drinks** <předmět> $\overset{13}{\Rightarrow}$ **our Mary drinks gin**

Standardní postup odvozování je takový, že se v aktuálním řetězci vždy přepisuje (rozvíjí) první neterminální symbol zleva (v literatuře se pro takový postup používá pojem *leftmost derivation*). Tak byl odvozen i náš řetězec.

Proč bylo při odvození použito pravidlo 3 <podmětná část> $\rightarrow$ <přívlastek><podmět> a ne pravidlo 2 <podmětná část> $\rightarrow$ <podmět> ? Bylo to určeno tím, jaký řetězec jsme chtěli odvodit. Použití pravidla 2 (tj. podmět bez přívlastku) by k požadovanému řetězci nevedlo. Ale ať použijeme při odvozování jakékoli pravidlo, finální *terminální řetězec*, ve kterém se vyskytují pouze anglická slova, vždy patří do jazyka popsaného naší gramatikou.

Příklady dalších řetězců, které lze pomocí našich pravidel odvodit: **your John drinks tea, Mary buys gin,** o**ur John buys tea.** Celkem tak lze odvodit 36 různých řetězců, jazyk z příkladu je tedy konečný.

Nalezneme-li pro zadaný řetězec jeho odvození, dokážeme tím, že tento řetězec patří do jazyka popsaného gramatikou. Tato úloha se nazývá *syntaktická analýza řetězce*. Pro speciální třídy jazyků, o nichž se zmíníme později, je syntaktická analýza řešitelná postupy, jejichž teoeretickými modely jsou různé typy automatů, v nejjednodušším případě pak rozpoznávací konečný automat.

2.6. Definice gramatiky jazyka

Gramatika je uspořádaná čtveřice $G = (N, T, S, P)$,

kde N je konečná neprázdná množina neterminálních symbolů
T je konečná neprázdná množina terminálních symbolů
$N \cap T = \emptyset$
$S \in N$ je počáteční symbol
P je množina přepisovacích pravidel ve tvaru $\alpha \to \beta$, kde
$\alpha \in (N \cup T)^* N (N \cup T)^*$ a $\beta \in (N \cup T)^*$.

Význam terminálních a neterminálních symbolů i počátečního symbolu byl objasněn v předchozí kapitole. Je pochopitelné, že symbol nemůže být současně terminální i neterminální, proto jsou množiny N a T disjunktní.

Tvar přepisovacích pravidel v definici je obecnější, než byl tvar pravidel z příkladu v kapitole 2.5. Z popisu symbolů α a β je zřejmé, že α je prvkem množiny řetězců, která vznikne zřetězením tří množin: množiny všech řetězců nad sjednocením $N \cup T$, množiny všech neterminálních symbolů N a (ještě jednou) množiny všech řetězců nad sjednocením $N \cup T$. Znamená to, že na levé straně přepisovacího pravidla může být libovolný řetězec tvořený terminálními a neterminálními symboly, ale musí obsahovat alespoň jeden neterminální symbol. Symbol β na pravé straně přepisovacího pravidla pak představuje libovolný řetězec tvořený terminálními a neterminálními symboly a může to být i prázdný řetězec. Nutnost výskytu alespoň jednoho neterminálního symbolu v řetězci na levé straně přepisovacího pravidla vyplývá z požadavku, aby terminální řetězec už nebylo možné dál přepisovat a jeho dosažením odvození skončilo. Na levých stranách pravidel z příkladu v 2.5 je vždy jen jediný symbol (samozřejmě neterminální), jsou tedy nejjednodušším možném tvaru.

Běžně používané konvence:

- neterminální symboly se označují špičatými závorkami (např. <podmět>, <identifikátor>, <příkaz while>) nebo velkými tiskacími písmeny (např. S, A, B)
- terminální symboly se označují číslicemi nebo malými písmeny ze začátku latinské abecedy (POZOR! S výjimkou symbolu e, který se používá pro označení prázdného řetězce)
- řetězce se označují malými písmeny z konce latinské abecedy nebo řeckými písmeny

V popisu odvození řetězce v kapitole 2.5 jsme při přepisování používali operátor $\Rightarrow$. Tento operátor budeme nazývat operátorem *přímého přepsání*. Odvození řetězce se tedy v tomto případě skládalo ze sedmi přímých přepsání. Přímé přepsání lze slovně charakterizovat takto: „V dosud odvozeném řetězci najdeme levou stranu přepisovacího pravidla a nahradíme ji příslušnou pravou stranou. Kontext (tj. část před a za nalezenou levou stranou přepisovacího pravidla) musí zůstat zachován.“ Přesně lze přímé přepsání definovat takto:

Nechť w a z jsou řetězce takové, že $w \in (N \cup T)^* N (N \cup T)^*$ a $z \in (N \cup T)^*$. Říkáme, že

řetězec w lze *přímo přepsat* na řetězec z, jestliže existují řetězce $x_1, x_2, u, v \in (N \cup T)^*$ takové, že $w = x_1 u x_2$, $z = x_1 v x_2$ a $u \to v \in P$. Značení: $w \Rightarrow z$.

Řetězce x_1 a x_2 nazýváme společným termínem *kontext* (x_1 je *levý kontext*, x_2 *pravý kontext*).

Zobecněním pojmu přímé přepsání je pojem *přepsání*, kterým budeme chápat konečnou posloupnost přímých přepsání:

Nechť w a z jsou řetězce takové, že $w \in (N \cup T)^* N (N \cup T)^*$ a $z \in (N \cup T)^*$. Říkáme, že řetězec w *lze přepsat* na řetězec z, jestliže existují řetězce $w_0, w_1, ..., w_n$ takové, že $w = w_0$, $z = w_n$ a $w_0 \Rightarrow w_1 \Rightarrow ..... \Rightarrow w_n$. Značení: $w \overset{*}{\Rightarrow} z$.

V kapitole 2.5 jsme konstatovali, že gramatika je nástrojem k popisu jazyka. Každá gramatika tedy jednozmačmě definuje nějakou množinu řetězců. Tuto množinu můžeme slovně popsat jako množinu všech terminálních řetězců, které lze v dané gramatice odvodit z jejího počátečního symbolu pomocí jejích přepisovacích pravidel. Tuto množinu nazveme *jazyk generovaný gramatikou.* Exaktně jej definujeme takto:

Jazykem $L(G)$ generovaným gramatikou G rozumíme množinu řetězců

$$L(G) = \{w \mid w \in T^* \wedge S \overset{*}{\Rightarrow} w\}.$$

Z definice je zřejmé, že do jazyka patří pouze terminální řetězce a že každý řetězec jazyka lze odvodit z počátečního symbolu.

Gramatiky G_1 a G_2 prohlásíme za ekvivalentní právě tehdy, jestliže generují stejný jazyk, tedy jestliže $L(G_1) = L(G_2)$.

Množiny neterminálních symbolů a přepisovací pravidla mohou být u ekvivalentních gramatik naprosto odlišné. Není rozhodující jak (a z čeho) jsou terminální řetězce odvozovány, ale zda jsou si množiny vygenerovaných řetězců rovny.

Příklad: $G = (N, T, S, P)$, $N = \{S\}$, $T = \{0,1\}$, S, $P = \{S \to 0S1, S \to 01\}$.

Použitý způsob zápisu gramatiky vychází důsledně z definice. Vzhledem k výše uvedeným konvencím ovšem tento způsob zápisu není obvyklý, dále budeme používat už jen zkrácený zápis, který tutéž gramatiku popíše úsporněji:

$$G: \quad S \to \overset{1}{0}S\overset{2}{1} \mid 01$$

Symbol S představuje neterminální symbol (je to velké písmeno latinky), 0 a 1 představují terminální symboly (jsou to číslice), počáteční symbol je S (je to první neterminální symbol

na levé straně pravidel). Gramatika má dvě přepisovací pravidla se stejnou levou stranou, proto je lze ve zkráceném zápisu sloučit do jednoho řádku, v němž jsou pravé strany odděleny svislou čárou. Pro názornost dalších úvah jsme nad každou pravou stranu přepisovacího pravidla napsali jeho číslo, obvykle se to ale nedělá.

Zkusme z naší gramatiky odvodit několik řetězců (číslice nad operátorem přímého přepsání říkají, které přepisovací pravidlo jsme v daném kroku použili):

$$S \overset{2}{\Rightarrow} 01$$
$$S \overset{1}{\Rightarrow} 0S1 \overset{2}{\Rightarrow} 0011$$
$$S \overset{1}{\Rightarrow} 0S1 \overset{1}{\Rightarrow} 00S11 \overset{2}{\Rightarrow} 000111$$
……

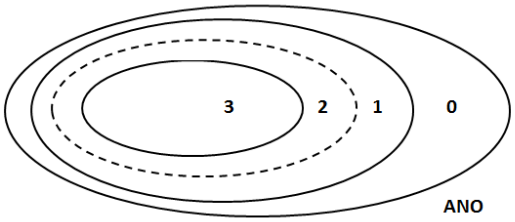
Z odvození tří nejkratších řetězců jazyka je zřejmé, že použijeme-li pravidlo 2, dojdeme k terminálnímu řetězci a odvozování tedy končí. Každé odvození se tedy skládá z konečného počtu použití pravidla 1 a je zakončeno použitím pravidla 2. Množinu terminálních řetězců, které mohou být odvozeny z počátečního symbolu, lze popsat jako $L = \{w \mid w = 0^n 1^n ; n \geq 1\}$, je to tedy jazyk, u něhož jsme v kapitole 2.4 dokázali, že není rozpoznatelný konečným automatem.

Přestože gramatika popisující tento jazyk vypadá velice jednoduše (jediný neterminální symbol a pouhá dvě přepisovací pravidla), konečný automat k rozpoznání jazyka nestačí. V dalším si ukážeme, že o tom, jaký syntaktický analyzátor je k rozpoznání jazyka zapotřebí, nerozhoduje počet přepisovacích pravidel ani počet neterminálních či terminálních symbolů, ale výhradně jen tvar přepisovacích pravidel.

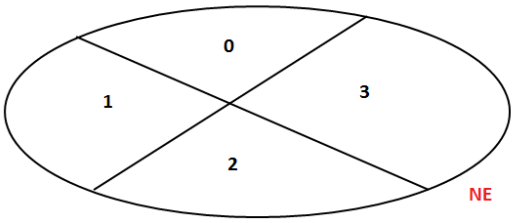
2.7. Chomského hierarchická klasifikace gramatik

Americký lingvista Noem Chomski zavedl na přelomu padesátých a šedesátých let minulého století klasifikaci gramatik podle tvaru jejich pravidel. Gramatiky jsou klasifikovány do čtyř tříd. POZOR! Na rozdíl od klasifikace objektů (řetězců) do disjunktních tříd, jak jsme ji poznali například u klasifikačního automatu, je Chomského klasifikace hierarchická, je tedy založena na *vnořování tříd*. Znamená to, že každá další (vyšší) třída gramatik je definována tak, že se zpřísní formální požadavky na tvar přepisovacích pravidel (obecný tvar přepisovacích pravidel konkrétní třídy gramatik je speciálním případem tvaru pravidel nižší třídy; uvidíme ale, že to neplatí absolutně). Požadavkům na tvar gramatik typu 0 vyhovují všechny gramatiky (třída 0 je nejobecnější), naopak nejužší třídou gramatik jsougramatiky typu3 (později ukážeme, že tyto gramatiky generují jazyky rozpoznatelné konečnými automaty).

Následující obrázek ilustruje oba přístupy ke klasifikaci.



Princip Chomského klasifikace



Princip klasifikace do disjunktních tříd

Gramatiky typu 0 (neomezené) (dále jen G0) mají všechna pravidla v nejobecnějším tvaru, jenž definice gramatiky umožňuje, tedy

$\alpha \to \beta$, kde $\alpha \in (N \cup T)^* N (N \cup T)^*$ a $\beta \in (N \cup T)^*$.

Gramatiky typu 1 (kontextové, CSG, nevypouštěcí) (G1) mají všechna pravidla ve tvaru

$\alpha X \beta \to \alpha \gamma \beta$, kde $\alpha, \beta \in (N \cup T)^*$, $X \in N$ a $\gamma \in (N \cup T)^+$.

Výjimka: v gramatice může být pravidlo $S \to e$, pak se ale symbol S nemůže vyskytnout na pravé straně žádného přepisovacího pravidla.

Gramatiky typu 2 (bezkontextové, BKG, CFG) (G2) mají všechna pravidla ve tvaru

$X \to \gamma$, kde $X \in N$ a $w \in (N \cup T)^*$.

Gramatiky typu 3 *(regulární) pravé lineární* (G3R) mají v všechna pravidla ve tvaru

$X \to wY$ nebo $X \to w$, kde $X, Y \in N$ a $w \in T^*$.

Gramatiky typu 3 *(regulární) levé lineární* (G3L) mají v všechna pravidla ve tvaru

$X \to Yw$ nebo $X \to w$, kde $X, Y \in N$ a $w \in T^*$.

Poznámky k Chomského klasifikaci:

1. Gramatika je typu *i* jsou-li všechna pravidla typu *i* nebo vyššího. Jinak řečeno – o typu gramatiky rozhoduje „nejhorší“ pravidlo (tedy pravidlo typu s nejnižším číslem).
2. POZOR! Vyskytují-li se v gramatice současně pravidla typu G3P a G3L, jedná se o gramatiku typu G2!!! Gramatiky typu G3P a G3L označujeme souhrnně jako typu G3.
3. S klesajícím typem gramatiky roste její „vyjadřovací sila“ – nižší třída je schopna generovat širší skupinu formálních jazyků než třída vyšší. Nejširší třídu jazyků generují gramatiky G0, nejužší třídu G3.
4. Je zřejmé, že tvar pravidel gramatik typu u G1 je takový, že existují gramatiky G2 a G3, které mu nevyhovují (na rozdíl od G1 mohou G2 a G3 obsahovat *e-pravidla* tvaru $\alpha \to e$, kde $\alpha \in N$). Proto je ve výše uvedeném ilustrativním obrázku třída G1 znázorněna čárkovaně, protože její zakreslení není zcela korektní.

5. Přívlastek *nevypouštěcí*, kterým jsou označeny G1, vypovídá o tom, že se (až na výjimku pravidla $S \to e$) neterminální symbol X nemůže v průběhu odvozování „ztratit bez náhrady“ (nemůže být vypuštěn).
6. *Kontextová gramatika x bezkontextová gramatika* – při prvním pohledu na tvar pravidel G1 a G2 se může zdát, že efekt obou pravidel je stejný – neterminální symbol X může být nahrazen řetězcem γ. Zásadní rozdíl je ovšem v tom, že u G1 k této náhradě může dojít pouze tehdy, je-li levým kontextem neterminálu X řetězec α a jeho pravým kontextem řetězec β. U G2 (bezkontextové gramatiky) k náhradě symbolu X řetězcem γ může dojít kdykoli, tedy v jakémkoli kontextu.
7. Zkratky používané pro gramatiky typu 2 – BKG (bezkontextová gramatika), resp. CFG (context free grammar).
8. Zkratky používané pro gramatiky typu 1 – KG (kontextová gramatika), resp. CSG (context sensitive grammar).

Příklad: Jakého typu jsou následující gramatiky:

$$G_1: \quad S \overset{3P\ 3}{\to} abA \mid ab$$
$$A \overset{3P\ 3}{\to} aaB \mid ba$$
$$B \overset{2\ 3}{\to} AB \mid e$$

$$G_2: \quad S \overset{3P\ 3}{\to} abA \mid ab$$
$$A \overset{3L\ 3}{\to} Bab \mid e$$
$$B \overset{3P\ 3}{\to} bbB \mid b$$

Čísla nad pravými stranami pravidel ukazují „nejpřísnější“ typ gramatiky, kterému konkrétní pravidlo vyhovuje. U pravidel označených typem 3 je zřejmé, že vyhovují jak gramatikám G3P, tak G3L.

U gramatiky G_1 je zřejmé, že „nejhorším pravidlem“ je pravidlo $B \to AB$, které je typu 2, proto je gramatika G_1 typu G2.

U gramatiky G_2 jsou „smíchána“ pravidla typu G3P a G3L, proto je gramatika G_2 také typu G2.

2.8. Hierarchické uspořádání tříd jazyků

Podobně, jako jsme klasifikovali gramatiky, se budeme snažit hierarchicky klasifikovat i jazyky. Naším cílem bude objasnit uspořádání mezi třídami jazyků různých typů. Na jedno úskalí jsme již narazili (viz poznámka 4 k Chomského klasifikaci), další si ukážeme v následujícím příkladu.

Příklad: Je dána gramatika G. Popište jazyk $L(G)$ množinovým popisem.

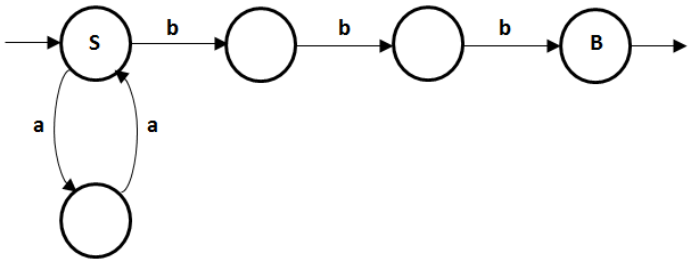
$$G: \quad S \xrightarrow{1} AB$$
$$A \xrightarrow{2} aaA \mid \xrightarrow{3} e$$
$$B \xrightarrow{4} bbb$$

Znaky nad pravými stranami tentokrát neoznačují typ pravidla, ale jeho číslo, na které se budeme odvolávat při odvozování. Vzhledem k pravidlu 1 je gramatika G typu G2.

Zkusme z naší gramatiky odvodit několik řetězců:

$S \overset{1}{\Rightarrow} AB \overset{3}{\Rightarrow} B \overset{4}{\Rightarrow} bbb$
$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aabbb$
$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{2}{\Rightarrow} aaaaAB \overset{3}{\Rightarrow} aaaaB \overset{4}{\Rightarrow} aaaabbb$
$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{2}{\Rightarrow} aaaaAB \overset{2}{\Rightarrow} aaaaaaAB \overset{3}{\Rightarrow} aaaaaaB \overset{4}{\Rightarrow} aaaaaabbb$
……

Je zřejmé, že každé odvození začíná použitím pravidla 1, poté následuje konečný počet použití pravidla 2 a odvození je zakončeno použitím pravidel 3 a 4. Jiné odvození z této gramatiky neexistuje. Jazyk $L(G)$ lze tedy popsat jako $L(G) = \{w \mid w = (aa)^n bbb \ \wedge \ n \geq 0\}$. Tento jazyk ovšem dokážeme rozpoznat konečným automatem:



Výše uvedený přechodový graf nás dovede k (zatím jen intuitivnímu) závěru, že jazyk $L(G)$ lze generativně popsat také gramatikou G': $S \to aaS \mid bbb$, která je typu G3P (obecné souvislosti mezi gramatikami typu G3P a rozpoznávacími konečnými automaty probereme později v kapitole 2.9).

Příklad nám ukázal, že jeden jazyk může být generován ekvivalentními gramatikami různých typů, a že tedy nelze mechanicky konstatovat „každá gramatika typu *i* generuje jazyk typu *i*.“

Typ jazyka je typ nejvyšší gramatiky (tedy gramatiky s nejpřísnějšími pravidly), která jazyk generuje. Jazyk z našeho příkladu tedy bude jazykem typu 3, přestože jsme jej vygenerovali z gramatiky typu G2.

K určení relací mezi třídami jazyků různých typů nám pomůže důležitý poznatek z teorie bezkontextových jazyků, který zde uvedeme bez vysvětlení (zájemce odkazuji na bohaté zdroje týkající se bezkontextových jazyků a transformací BKG gramatik):

Ke každé bezkontextové gramatice G existuje ekvivalentní bezkontextová *gramatika bez e-pravidel* G' taková, že buď neobsahuje žádné e-pravidlo, nebo obsahuje jediné e-pravidlo $S \to e$, pak se ale symbol S nevyskytuje na žádné pravé straně přepisovacího pravidla.

Uvědomíme-li si, že výše uvedené tvrzení platí i pro regulární gramatiky (gramatiky G3 jsou bez výjimek zvláštním případem gramatik G2), můžeme přistoupit k formulaci uspořádání mezi třídami jazyků.

Označíme-li symbolem $\mathcal{L}_i$ třídu všech jazyků typu *i*, platí relace $\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$.
Vysvětlení: inkluze $\mathcal{L}_0 \supseteq \mathcal{L}_1$ a $\mathcal{L}_2 \supseteq \mathcal{L}_3$ jsou zřejmé (gramatiky G1 jsou zvláštním případem G0, gramatiky G3 jsou zvláštním případem G2), inkluze $\mathcal{L}_1 \supseteq \mathcal{L}_2$ je zřejmá poté, co si uvědomíme, že bezkontextová gramatika bez e-pravidel (na níž podle výše uvedeného tvrzení můžeme transformovat každou gramatiku G2) je zvláštním případem gramatiky G1.

V následující tabulce jsou pro všechny třídy jazyků uvedeny standardně používané názvy a modely jejich syntaktických analyzátorů:

Třída jazyků	**Používaný název**	**Model syntaktického analyzátoru**
0	rekurzivně vyčislitelné jazyky	Turingův stroj
1	kontextové jazyky	lineárně omezený Turingův stroj
2	bezkontextové jazyky	nedeterministický zásobníkový automat
3	regulární jazyky	konečný automat

Největší praktické použití mají bezkontextové jazyky, protože mají vysoce rozpracované metody automatického překladu. Pro vhodně navržené jazyky je syntaktická analýza deterministická (např. u jazyků generovaných LL(k) gramatikami). Současné programovací a specifikační jazyky jsou vesměs jazyky třídy 2.

V omezené míře nacházejí praktické použití i regulární jazyky, a to např. v jednodušších úlohách umělé inteligence (příznaková anylýza, rozpoznávání objektů) nebo při lexikální analýze (popis a rozpoznávání klíčových slov, identifikátorů a konstant v překladačích programovacích jazyků). S regulárními jazyky úzce souvisí zpracovávání regulárních výrazů **regexp** ve „skriptovacích jazycích“ jako jsou např. PHP, AWK, PERL (viz kapitola).

V dalším textu se budeme zabývat pouze jazyky typu 3 a nástroji pro jejich syntaktickou analýzu. Bezkontextovými jazyky se zabývá předmět KIV/FJP v další etapě studia, s Turingovými stroji se zájemci mohou setkat zejména v souvislosti s modelováním algoritmů z hlediska výpočetní složitosti, např. v předmětu KMA/TGD1.

2.9. Vztah mezi jazyky typu 3 a konečnými automaty

V předcházejím textu jsme několikrát konstatovali, že jsou regulární jazyky rozpoznatelné konečnými automaty, aniž bychom toto tvrzení nějak blíže objasnili. V této kapitole se budeme věnovat právě souvislostem mezi gramatikami G3 a konečnými rozpoznávacími automaty a výše uvedené tvrzení dokážeme.

Nejprve připomeneme mechanismus, kterým konečný automat rozpoznává jazyk (viz kapitoly 1.2 a 2.4): Pokud vstupní řetězec převede automat z počátečního stavu do některého ze stavů z množiny koncových stavů, vydává tím automat o zpracovaném řetězci informaci „ano, tento řetězec akceptuji“, pokud řetězec převede automat z počátečního stavu do některého ze stavů mimo množinu koncových stavů, vydává tím automat rozhodnutí „ne, tento řetězec zamítám“. Každý rozpoznávací automat tak jednoznačně definuje množinu řetězců (*jazyk rozpoznávaný automatem*) jako množinu všech řetězců, jež akceptuje.

Exaktně můžeme *jazyk rozpoznávaný konečným automatem* $A = (Q, \Sigma, \delta, q_0, F)$ vyjádřit jako $L(A) = \{w \mid w \in \Sigma^* \wedge \delta^*(q_0, w) \in F\}$.

Symbol δ^* představuje zobecněnou přechodovou funkci, definovanou a vysvětlenou v kapitole 1.4.

Pokud dva rozpoznávací konečné automaty budou rozpoznávat stejný jazyk, budeme je považovat za ekvivalentní. Exaktněji:

Automaty $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ a $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ jsou *ekvivalentní* právě tehdy, když platí $L(A_1) = L(A_2)$.

POZOR! Ekvivalentní mohou být i takové automaty, které nemají stejný počet stavů. *Redukovaný automat (minimální automat)* je takový reprezentant třídy ekvivalentních automatů, který má minimální počet stavů. Jsou známy algoritmy pro redukci automatu (tj. pro minimalizaci počtu jeho stavů).

Příklad: Je dána gramatika G:

$$G: \quad S \xrightarrow{1} 0A \mid 1S \mid \overset{2}{\overset{3}{e}}$$
$$A \xrightarrow{4} 0B \mid \overset{5}{1}A$$
$$B \xrightarrow{6} 0S \mid \overset{7}{1}B$$

Gramatika G je pravá lineární gramatika, je tedy gramatikou typu G3P.

Zkonstruujeme „automat“ tímto formálním postupem:

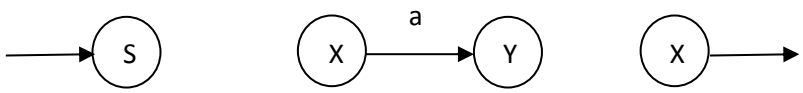
- stavy „automatu“ budou odpovídat neterminálním symbolům gramatiky

- vstupní symboly „automatu“ budou odpovídat terminálním symbolům gramatiky
- počáteční stav „automatu“ bude odpovídat počátečnímu symbolu gramatiky
- množinu koncových stavů „automatu“ určíme tak, že v ní budou všechny stavy X, pro které je pravidlo $X \to e$ v přepisovacích pravidlech gramatiky G
- přechodovou funkci „automatu“ zkonstruujeme tak, že každému pravidlu typu $X \to aY$ z gramatiky G bude odpovídat hrana ze stavu X do stavu Y ohodnocená vstupním symbolem a

Výše uvedené analogie mezi gramatikou a „automatem“ názorně:

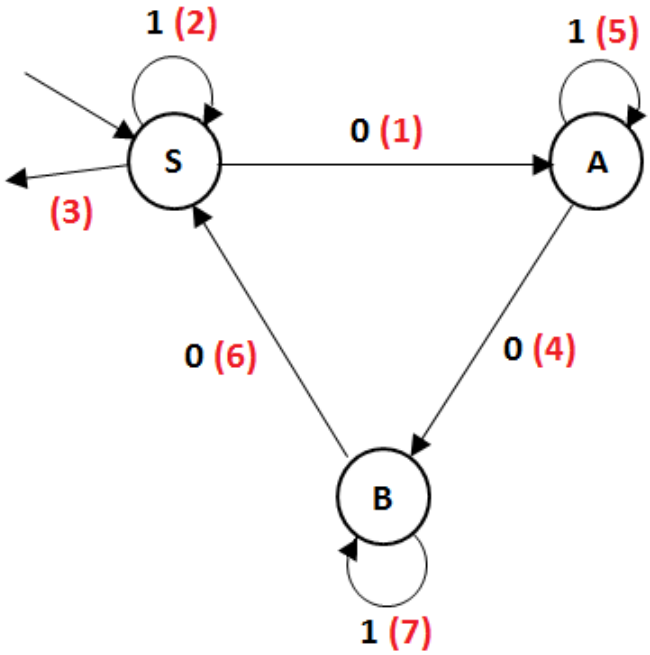
Gramatika: počáteční symbol S pravidlo $X \to aY$ pravidlo $X \to e$

„Automat“:



Na následujícím obrázku je vidět graf, který vznikl na základě výše uvedených analogií z gramatiky G. Symboly u orientovaných hran chápejme takto:

- černé číslice představují vstupní symbol
- červené číslice v závorkách označují přechody v grafu (a koncový stav); toto označení respektuje označení pravidel výchozí gramatiky, kterému odpovídají ve výše uvedených anologiích



Při bližším zkoumání tohoto grafu zjistíme, že v každém stavu je pro vstupní symboly 0 a 1 jednoznačně definován následující stav, tento graf je tedy přechodovým grafem rozpoznávacího konečného automatu, který pro následující úvahy označíme A. POZOR! To, že graf získaný na základě analogií je přechodovým grafem automatu, je pouze shodou okolností, která je dána vhodně zvolenou výchozí gramatikou (proto jsme v předchozích odstavcích psali „au-

tomat“) . Popsaný postup obecně k přechodovému grafu konečného automatu ve smyslu naší definice nevede, jak bude ukázáno později.

Zkusíme z gramatiky G odvodit řetězec, například

$$S \overset{1}{\Rightarrow} 0A \overset{4}{\Rightarrow} 00B \overset{7}{\Rightarrow} 001B \overset{6}{\Rightarrow} 0010S \overset{2}{\Rightarrow} 00101S \overset{3}{\Rightarrow} 00101$$

Nyní budeme zkoumat, jakým způsobem tento řetězec zpracuje automat A (nad operátorem přechodu mezi konfiguracemi je uvedeno označení příslušné hrany v přechodovém grafu):

$$(S,00101) \overset{1}{\mapsto} (A,0101) \overset{4}{\mapsto} (B,101) \overset{7}{\mapsto} (B,01) \overset{6}{\mapsto} (S,1) \overset{2}{\mapsto} (S,e)$$

Je vidět, že (díky analogiím, podle nichž jsme konstruovali přechodový graf) posloupnost přechodů v automatu jednoznačně odpovídá posloupnosti přímých přepsání v odvození řetězce (samozřejmě až na použití pravidla 3 $S \to e$, kterému v analogii neodpovídá přechodová hrana, ale koncový stav). Je tedy zřejmé, že

- každému řetězci vygenerovanému gramatikou G v automatu A jednoznačně odpovídá posloupnost přechodů, která začíná v počátečním a končí v koncovém stavu
- každé posloupnosti přechodů A, která začíná v počátečním a končí v koncovém stavu, jednoznačně odpovídá odvození terminálního řetězce v gramatice G

Platí tedy $L(A) = L(G)$

Na základě uvedených analogií lze sestrojit přechodový graf rozpoznávacího konečného automatu pouze ke gramatikám typu G3P, které splňují navíc ještě přísnější požadavky na tvar pravidel : $X \to aY$ nebo $X \to e$, kde $X,Y \in N$ a $a \in T$ a kde navíc pro žádný neterminální symbol neexistuje více pravidel se stejným terminálním symbolem na pravé straně. Dúvody pro oba požadavky jsou zřejmé:

- analogie jsme měli formulovány právě jen k pravidlům tvaru $X \to aY$ a $X \to e$
- kdyby v gramatice existovala např. pravidla $S \to 0A$ a $S \to 0S$, nebyl by přechod ze stavu S vstupním symbolem 0 určen jednoznačně, graf by tedy nebyl přechodovým grafem konečného automatu ve smyslu naší definice z kapitoly 1.2.

Naše snaha o nalezení obecného postupu, který ke <u>každé</u> gramatice typu G3P dokáže zkonstruovat rozpoznávací konečný automat, povede k následujícím krokům:

- ukážeme, že ke každé gramatice typu G3P existuje ekvivalentní *gramatika v regulárním tvaru* G3R, která bude mít pravidla právě ve tvaru $X \to aY$ nebo $X \to e$ kde $X,Y \in N$ a $a \in T$
- pojem konečného automatu zobecníme na *nedeterministický konečný automat*, v němž připustíme i nejednoznačně definované přechody
- ukážeme, že ke každému nedeterministickému konečnému automatu lze sestrojit ekvivalentní deterministický konečný automat, tedy automat ve smyslu naší původní definice z 1.2
- nalezené postupy uplatníme i pro rozpoznávání jazyků generovaných gramatikami G3L

2.10. Převod gramatiky typu G3P na regulární tvar

Tvrzení: Ke každé gramatice $G = (N,T,S,P)$ typu G3P existuje gramatika $G' = (N',T,S,P')$ s pravidly v regulárním tvaru $X \to aY$ nebo $X \to e$, kde $X,Y \in N$ a $a \in T$, taková, že $L(G) = L(G')$.

Toto tvrzení má pro další postup zásadní význam, proto je i dokážeme. Důkaz je konstruktivní, takže poskytuje návod, jak gramatiku v regulárním tvaru vytvořit.

Důkaz (= postup pro konstrukci ekvivalentní gramatiky v regulárním tvaru) :

1. Množina terminálních symbolů T a počáteční symbol S jsou v G' stejné jako v G.
2. Množinu přepisovacích pravidel P' zkonstruujeme takto:
   a. Do P' zařadíme všechna pravidla z P, která jsou v požadovaném regulárním tvaru $X \to aY$ nebo $X \to e$, kde $X,Y \in N$ a $a \in T$.
   b. Za každé pravidlo $X \to x_1x_2....x_{n-1}x_nY$ z P ($X,Y \in N$, $x_i \in T$) přidáme do P' soustavu pravidel $X \to x_1X_1$, $X_1 \to x_2X_2$, ……, $X_{n-2} \to x_{n-1}X_{n-1}$, $X_{n-1} \to x_nY$ ($X_i \in N$) .
   c. Za každé pravidlo $X \to z_1z_2....z_n$ z P ($X \in N$, $z_i \in T$) přidáme do P' soustavu pravidel $X \to z_1Z_1$, $Z_1 \to z_2Z_2$, ……, $Z_{n-1} \to z_nZ_n$, $Z_n \to e$ ($Z_i \in N$) .
   d. Místo pravidel tvaru $X \to Y$ z P přidáme do P' soustavu pravidel ve tvaru $Z' \to z\ Z''$ $\forall\ Z' \in U(Y)$ $\forall\ Y \to z\ Z'' \in P$ kde $U(Y) = \{X \mid X \overset{*}{\Rightarrow} Y\}$
3. Množina neterminálních symbolů N' vznikne obohacením množiny N o všechny nové neterminální symboly vytvořené v bodech 2a, 2b a 2c.

Vysvětlení k některým bodům konstrukce:

ad 2b) Následující dvě odvození ukazují, že náhrada pravidla $X \to x_1x_2....x_{n-1}x_nY$ soustavou pravidel $X \to x_1X_1$, $X_1 \to x_2X_2$, ……, $X_{n-2} \to x_{n-1}X_{n-1}$, $X_{n-1} \to x_nY$ je korektní:

Odvození v G : $X \Rightarrow x_1x_2 \ldots . x_nY$

Odvození v G' : $X \Rightarrow x_1X_1 \Rightarrow x_1x_2X_2 \Rightarrow \ldots . \Rightarrow x_1x_2 \ldots . x_{n-1}X_{n-1} \Rightarrow x_1x_2 \ldots . x_{n-1}x_nY$

V obou odvozeních lze z neterminálního symbolu X odvodit jediný řetězec složený ze symbolů gramatiky G, a to $x_1x_2....x_{n-1}x_nY$.

Konkrétní příklad nahrazení:
Pravidlo v G $A \to bbaS$
Náhrada v G' : $A \to bA_1$, $A_1 \to bA_2$, $A_2 \to aS$

ad 2c) Podobně i náhrada pravidla $X \to z_1 z_2 \ldots . z_n$ soustavou pravidel $X \to z_1 Z_1$, $Z_1 \to z_2 Z_2$, ……., $Z_{n-1} \to z_n Z_n$, $Z_n \to e$:

Odvození v G : $X \Rightarrow z_1 z_2 \ldots . z_n$

Odvození v G' : $X \Rightarrow z_1 Z_1 \Rightarrow z_1 z_2 Z_2 \Rightarrow \ldots . \ldots . \Rightarrow z_1 z_2 \ldots . z_n Z_n \Rightarrow z_1 z_2 \ldots . z_n$

V obou odvozeních lze z neterminálního symbolu X odvodit jediný terminální řetězec, a to $z_1 z_2 \ldots . z_n$.

Konkrétní příklad nahrazení:
Pravidlo v G $A \to abba$
Náhrada v G' : $A \to aA_1$, $A_1 \to bA_2$, $A_2 \to bA_3$, $A_3 \to aA_4$, $A_4 \to e$

ad 2d) Náhradu pravidel tvaru $X \to Y$ už nelze provádět tak jednoduše mechanicky. Princip náhrady ukážeme na jednoduchém příkladu:

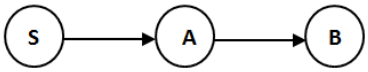
Pravidla tvaru $X \to Y$ v G : $S \to A$, $A \to B$
Všechna A a B pravidla, která jsou v této fázi zpracování v G' : $A \to aC \mid e$, $B \to aB \mid bA$

Je zřejmé, že pravidla tvaru $X \to Y$ v G umožňují tato odvození: $S \Rightarrow A \Rightarrow B$, $A \Rightarrow B$.
Z neterminálu S tedy lze odvodit neterminály A a B , z neterminálu A lze odvodit neterminál B. Zapsáno pomocí operátoru přepsání: $S \overset{*}{\Rightarrow} A$, $S \overset{*}{\Rightarrow} B$, $A \overset{*}{\Rightarrow} B$.
Jestliže má být gramatika G' ekvivalentní s G , musíme přidat do G' taková pravidla, která umožní z neterminálů vyskytujících se v pravidlech tvaru $X \to Y$ odvodit všechny řetězce, které z nich lze odvodit v gramatice G .
Množina $U(B) = \{ X \mid X \overset{*}{\Rightarrow} B \}$ je množina všech neterminálních symbolů ze kterých lze odvodit řetězec B , v našem případě tedy $U(B) = \{S, A\}$, $U(A) = \{S\}$, $U(S) = \emptyset$. Abychom dosáhli ekvivalence, musíme pro každou množinu $U(X)$ do G' přidat taková pravidla, která prokombinují na levé straně všechny netrminály z množiny $U(X)$ se všemi pravými stranami X pravidel z G' . V našem příkladu to konkrétně znamená doplnit do G' tato pravidla: $S \to aB$, $S \to bA$, $A \to aB$, $A \to bA$, $S \to aC$, $S \to e$.

Poznámky k praktickému provedení transformce:

- Provedení bodů 2b a 2c je triviální (postup lze popsat jako „postupné odřezávání“ terminálních symbolů)
- K provedení bodu 2d je u složitějších gramatik vhodné nakreslit pomocný graf, který bude zobrazovat pravidla tvaru $X \to Y$, a z něj vyčíst množiny $U(X)$
- Důležitým faktorem je udržet si přehled v tom, která pravidla jsem již zpracovali (tj. nahradili ekvivalentními soustavami pravidel), je tedy vhodné si zpracovaná pravidla v zadání označovat (odškrtávat)

Pro pravidla $S \to A$, $A \to B$ v G bude pomocný graf vypadat takto:



Za každé pravidlo $X \to Y$ v G je v pomocném grafu orientovaná hrana z vrcholu X do vrvholu Y. Množiny $U(X)$ pak snadno vytvoříme jako množiny všech (cizích) vrcholů, z nichž vede cesta do vrcholu X. Z teorie grafů víme, že z každého vrcholu X vede také cesta délky 0 do vrcholu X, v našem případě cesty délky 0 nepřinášejí potřebu nových přepisovacích pravidel, proto sám vrchol X prvkem množiny $U(X)$ nebude. Z pomocného grafu snadno zjistíme, že $U(S) = \emptyset$, $U(A) = \{S\}$, $U(B) = \{S, A\}$.

Příklad: Gramatiku G transformujte na regulární tvar:

$$G: \quad S \xrightarrow{1} A \mid \xrightarrow{2} B \mid \xrightarrow{3} aaS$$
$$A \xrightarrow{4} C \mid \xrightarrow{5} abB$$
$$B \xrightarrow{6} bB \mid \xrightarrow{7} c$$
$$C \xrightarrow{8} cC \mid \xrightarrow{9} bc$$

Množinu přepisovacích pravidel začneme vytvářet od pravidel, která už v gramatice G v regulárním tvaru $X \to aY$ nebo $X \to e$ jsou (bod 2a postupu). To jsou pravidla 6 a 8, takže do P' zařadíme pravidla $B \to bB$ a $C \to cC$.

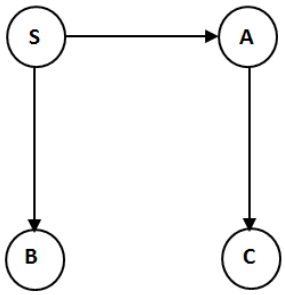
Podle bodu 2b pak transformujeme všechna pravidla ve tvaru $X \to x_1x_2....x_{n-1}x_nY$. To jsou pravidla 3 a 5, takže do P' zařadíme pravidla $S \to aS_1$, $S_1 \to aS$, $A \to aA_1$ a $A_1 \to bB$.

Podle bodu 2c pak transformujeme všechna pravidla ve tvaru $X \to z_1z_2....z_{n-1}z_n$, ted pravidla 7 a 8. Do P' proto zařadíme pravidla $B \to cB_1$, $B_1 \to e$, $C \to bC_1$, $C_1 \to cC_2$. $C_2 \to e$.

Pro názornost si v této fázi pravidla dosud zařazená do P' uspořádáme:

$$S \to aS_1, \quad S_1 \to aS$$
$$A \to aA_1, \quad A_1 \to bB$$
$$B \to bB \mid cB_1, \quad B_1 \to e,$$
$$C \to cC \mid bC_1, \quad C_1 \to cC_2 . \; C_2 \to e$$

V gramatice G v této fázi zbyla nezpracovaná už jen pravidla tvaru $X \to Y$, tedy $S \to A$, $S \to B$, $A \to C$. Nakreslíme pomocný graf a z něho vyčteme množiny vrcholů $U(X)$:



$U(S) = \emptyset$
$U(A) = \{S\}$,
$U(B) = \{S\}$
$U(C) = \{S, A\}$.

Protože už jsou v G' pravidla

$A \to aA_1$, $B \to bB \mid cB_1$ a $C \to cC \mid bC_1$,

přidáme do G' pravidla

$S \to aA_1$ (protože $S \in U(A)$)

$S \to bB \mid cB_1$ (protože $S \in U(B)$)

$S \to cC \mid bC_1$ (protože $S \in U(C)$) a

$A \to cC \mid bC_1$ (protože $A \in U(C)$)

Kompletní zápis ekvivalentní gramatiky G' po uspořádání a očíslování pravidel:

G':

1 2 3 4 5 6

$S \to aS_1 \mid aA_1 \mid bB \mid cB_1 \mid cC \mid bC_1$

7

$S_1 \to aS$

8 9 10

$A \to aA_1 \mid cC \mid bC_1$

11

$A_1 \to bB$

12 13

$B \to bB \mid cB_1 \mid$

14

$B_1 \to e$

15 16

$C \to cC \mid bC_1$

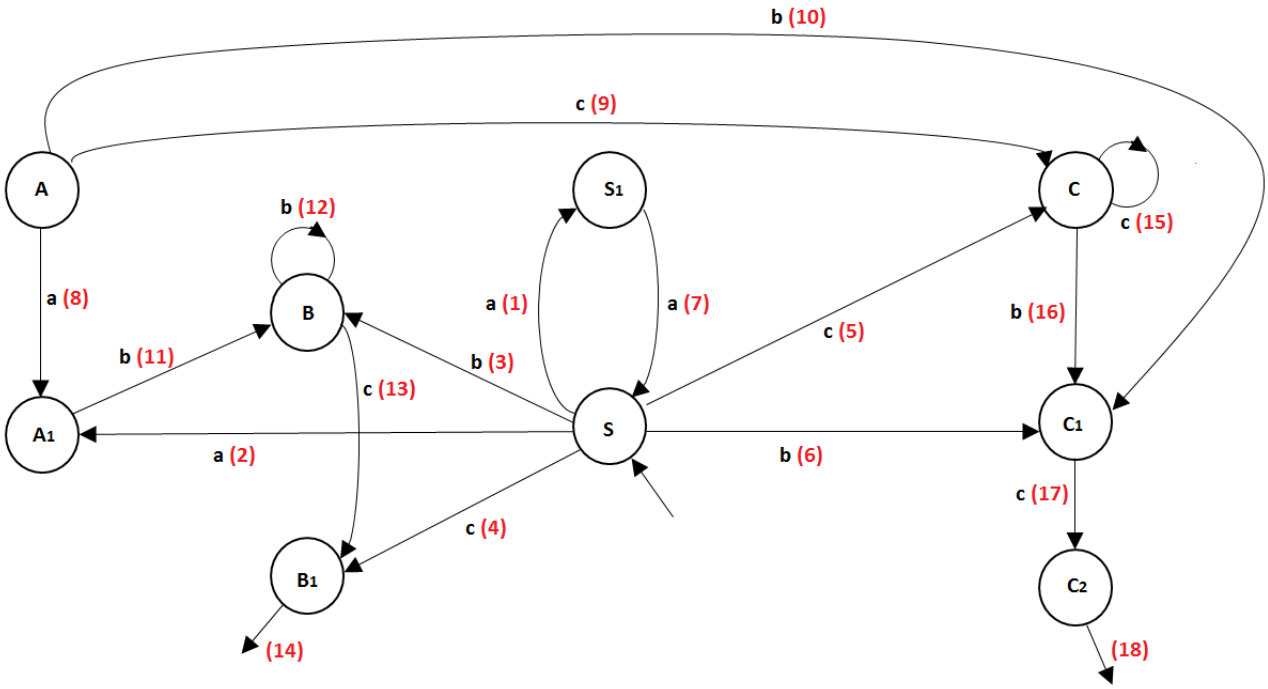
17

$C_1 \to cC_2$

18

$C_2 \to e$

Podle pravidel v regulárním tvaru nakreslíme na základě dříve uvedených analogií graf:



Je zřejmé, že přechody ze stavu S nejsou jednoznačné, tento graf tedy není grafem konečného automatu ve smyslu definice z kapitoly 1.2.

2.11. Nedeterministický rozpoznávací konečný automat

V této kapitole zobecníme model rozpoznávacího konečného automatu v tom smyslu, že bude v přechodovém grafu z jednoho stavu připouštět existenci více výstupních hran označených stejným symbolem. Tyto *nejednoznačné přechody* budeme chápat tak, že automat v takové situaci může vykonat libovolný z možných přechodů, a to nezávisle na svém okolí. Hovoříme o nedeterminismu (nejednoznačnosti), tento model budeme nazývat *nedeterministický rozpoznávací konečný automat*. Stejně jako (deterministický) automat definovaný v 1.2 se i nedeterministický automat v libovolný okamžik své existence nachází právě v jednom stavu. U deterministického automatu jsme ovšem ze znalosti přechodové funkce a vstupního řetězce jednoznačně schopni určit posloupnost stavů, kterými automat při zpracování řetězce prošel, u nedeterministického automatu tomu tak být nemusí.

Je zřejmé, že při výše uvedeném zobecnění můžeme konstatovat, že ke každé gramatice typu 3 v regulárním tvaru lze (na základě dříve uvedených formálních analogií) sestrojit nedeterministický konečný automat.

Z důvodů, které budou zřejmé z dalších kapitol, zavedeme do modelu ještě další dva typy nedeterminismu: možnost nejednoznačně určeného počátečního stavu a možnost samovolných přechodů, které mohou v automatu proběhnout, aniž by automat zpracoval vstupní znak.

Nedeterministický rozpoznávací konečný automat (NKA) je uspořádaná pětice

$A = (Q, \Sigma, \delta, S, F)$,

kde Q je konečná neprázdná množina stavů
Σ je konečná neprázdná množina vstupních symbolů
$S \subseteq Q$ je množina počátečních stavů
$\delta : Q \times (\Sigma \cup \{e\}) \to 2^Q$ je přechodová funkce
$F \subseteq Q$ je množina koncových stavů

Poznámky k definici nedeterministického rozpoznávacího konečného automatu:

1. Na začátku své činnosti se automat nachází v některém ze stavů z množiny počátečních stavů S.
2. V definičním oboru přechodové funkce δ je (oproti deterministickému KA) množina vstupních symbolů Σ „formálně obohacena“ o symbol prázdného řetězce e. Znamená to, že se v přechodovém grafu mohou vyskytovat hrany označené symbolem e Tyto *e-hrany* vyjadřují, že automat může (ale nemusí) provést *e-hranou* samovolný přechod, aniž by zpracoval symbol ze vstupního řetězce.
3. Obor hodnot přechodové funkce 2^Q představuje potenční množinu (tedy množinu všech podmnožin) množiny stavů Q. Zobrazení množiny všech dvojic (stav, vstupní symbol) do 2^Q (a nikoli jen do Q jako u (D)KA) souvisí s tím, že přechod v NKA nemusí být jednoznačný, tedy pro konkrétní stav a vstupní písmeno může být hodnotou přechodové funkce více než jeden stav.

Tento model NKA tedy umožňuje tři typy nedeterminismu:

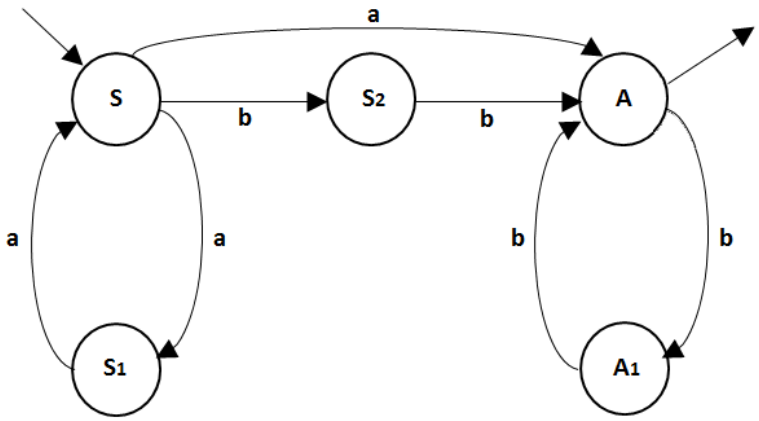
- nejednoznačně určený počáteční stav
- nejednoznačné přechody
- spontánní přechody (e-hrany)

Je zřejmé, že na rozdíl od (D)KA, kde jsme byly schopni vždy jednoznačně určit, v jakém stavu se automat nachází, jsme u NKA schopni určit pouze to, v jakém stavu se automat může nacházet. Budeme ale nuceni najít nový pohled na to, co u NKA je a co není akceptovaný řetězec. To, co bylo u (D)KA jednoznačné, totiž zda řetězec převedl či nepřevedl automat z počátečního do některého ze stavů koncových, díky nedeterminismu u NKA jednoznačné není. Ukážeme si to na příkladě:

Příklad:

$$G: \quad S \xrightarrow{1} aaS \mid \xrightarrow{2} bbA \mid \xrightarrow{3} aA$$
$$A \xrightarrow{4} bbA \mid \xrightarrow{5} e$$

Gramatika obsahuje jen několik málo pravidel, na pravých stranách se nevyskytují nijak dlouhé řetězce a gramatika neobsahuje pravidla tvaru $X \to Y$, přechodový graf je tedy snadné nakreslit rovnou, tj. bez „mezipřevodu“ gramatiky na regulární tvar:



Ukážeme si dva způsoby, kterými může být v našem NKA zpracován řetězec a :

$$(S,a) \overset{3}{\mapsto} (A,e) \qquad A \overset{(5)}{\in} F$$
$$(S,a) \overset{1}{\mapsto} (S_1,e) \qquad S_1 \notin F$$

Jeden ze dvou možných výpočtů končí v koncovém stavu, druhý ve stavu nekoncovém. Je zřejmé, že k definici toho, co je akceptovaný řetězec, nemůžeme použít přístup, který jsme zavedli u (D)KA. Uvědomíme-li si ale, že jsme model NKA vytvořili proto, abychom dokázali rozpoznávat jazyky generované regulárními gramatikami, je přirozené, že by akceptování řetězce mělo být definováno tak, aby byl řetězec akceptován právě tehdy, když je generován výchozí gramatikou.

V gramatice z našeho příkladu platí

$S \overset{3}{\Rightarrow} aA \overset{5}{\Rightarrow} a$, tedy $a \in L(G)$; je zřejmé, že toto odvození odpovídá prvnímu z uvedených zpracování řetězce a automatem.

Druhé uvedené zpracování řetězce a, které nekončí v koncovém stavu, odpovídá odvození řetězce, jenž není terminální, tedy $S \overset{1}{\Rightarrow} aS_1$; toto odvození sice „může pokračovat dál“, ale nikoli tak, že by se symbol S_1 přepsal na prázdný řetězec.

Výše uvedené úvahy nás přivádí k následujícímu pojetí akceptování/zamítání řetězců v NKA: Řetězec je nedeterminstickým konečným automatem akceptován, pokud v přechodovém grafu existuje alespoň jedna cesta , jejíž hrany jsou ohodnoceny znaky řetězce (nebo symboly *e*), která automat převádí z některého z počátečních stavů do některého z koncových stavů.

Přesnějsi formulace pro NKA bez e-hran:

Slovo $w = w_1 w_2 \ldots\ldots w_n$ je nedeterministickým konečným automatem akceptováno právě tehdy, existuje-li posloupnost stavů $q_0, q_1, q_2, \ldots\ldots, q_n$ taková, že $q_0 \in S$, $q_n \in F$ a $q_{i+1} \in \delta(q_i, w_i) \quad \forall i = 1,2,\ldots,n-1$.

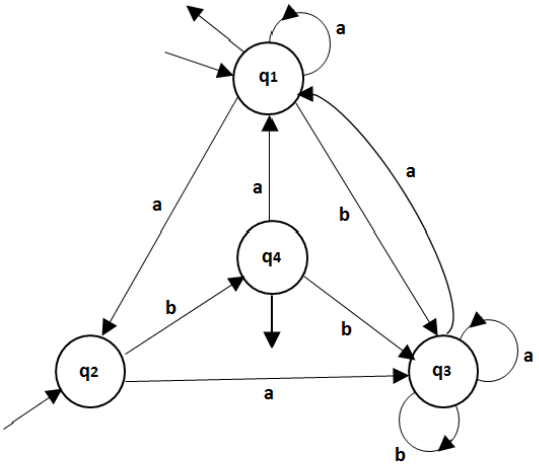
Kdy NKA akceptuje prázdné slovo e ? Jestliže existuje stav, který je současně počáteční i koncový, tj. jestliže $S \cap F \neq \emptyset$, nebo pokud v přechodovém grafu existuje cesta složená výhradně z e-hran z některého z počátečních stavů do některého z koncových stavů.

Jiná formulace: Řetězec je akceptován, jestliže existuje alespoň jeden výpočet, který začíná v některém z počátečních a končí v některém z koncových stavů. Různé možnosti zpracování konkrétního řetězce nedeterministickým automatem chápeme jako různé výpočty. Je zřejmé, že u deterministického automatu pro každý řetězec existuje právě jedno zpracování, tedy právě jeden výpočet.

Příklad: NKA bez e-hran je zadán tabulkou:

		a	b
↔	**q_1**	$\{q_1,q_2\}$	$\{q_3\}$
→	**q_2**	$\{q_3\}$	$\{q_4\}$
	q_3	$\{q_1,q_3\}$	$\{q_3\}$
←	**q_4**	$\{q_1\}$	$\{q_3\}$

Pro názornost si jej znázorníme i přechodovým grafem, abychom mohli snáze zkoumat, jak mohou být zpracovány různé vstupní řetězce .



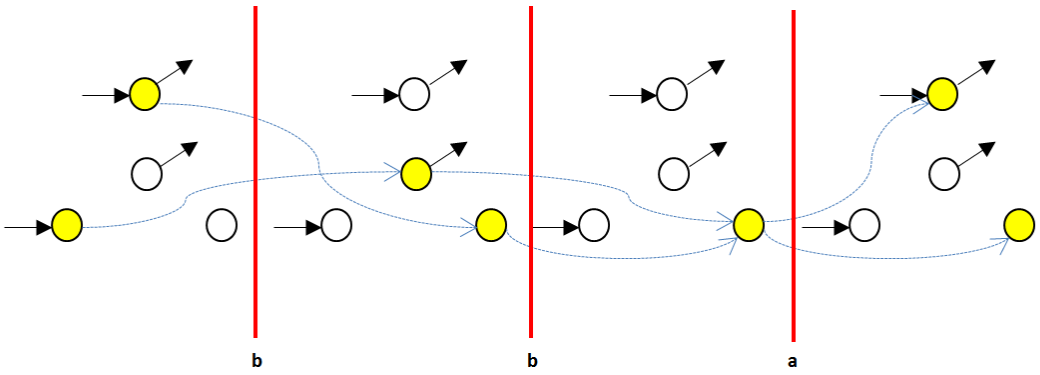
Jaké různé výpočty mohou proběhnout při zpracování vstupního řetězce *abaa* ?

$$(q_1, abaa) \mapsto (q_1, baa) \mapsto (q_3, aa) \mapsto (q_1, a) \mapsto (q_3, e) \qquad q_3 \notin F$$
$$(q_1, abaa) \mapsto (q_1, baa) \mapsto (q_3, aa) \mapsto (q_3, a) \mapsto (q_3, e) \qquad q_3 \notin F$$
$$(q_1, abaa) \mapsto (q_2, baa) \mapsto (q_4, aa) \mapsto (q_1, a) \mapsto (q_3, e) \qquad q_3 \notin F$$
$$(q_2, abaa) \mapsto (q_3, baa) \mapsto (q_3, aa) \mapsto (q_1, a) \mapsto (q_3, e) \qquad q_3 \notin F$$
$$(q_2, abaa) \mapsto (q_3, baa) \mapsto (q_3, aa) \mapsto (q_3, a) \mapsto (q_3, e) \qquad q_3 \notin F$$
$$(q_1, abaa) \mapsto (q_1, baa) \mapsto (q_3, aa) \mapsto (q_1, a) \mapsto (q_1, e) \qquad q_1 \in F$$
$$(q_1, abaa) \mapsto (q_1, baa) \mapsto (q_3, aa) \mapsto (q_3, a) \mapsto (q_1, e) \qquad q_1 \in F$$
$$(q_1, abaa) \mapsto (q_2, baa) \mapsto (q_4, aa) \mapsto (q_1, a) \mapsto (q_1, e) \qquad q_1 \in F$$
$$(q_2, abaa) \mapsto (q_3, baa) \mapsto (q_3, aa) \mapsto (q_1, a) \mapsto (q_1, e) \qquad q_1 \in F$$
$$(q_2, abaa) \mapsto (q_3, baa) \mapsto (q_3, aa) \mapsto (q_3, a) \mapsto (q_1, e) \qquad q_1 \in F$$

To, že řetězec *abaa* patří do jazyka akceptovaného naším automatem, jsme dokázali nalezením prvního výpočtu, který skončil v koncovém stavu. Všechny ostatní výpočty už byly nadbytečné.

Pokud bychom chtěli dokázat, že nějaký konkrétní řetězec nepatří do jazyka akceptovaného NKA, museli bychom výše uvedeným způsobem pro řetězec provést všechny možné výpočty a všechny by musely skončit v některém z „nekoncových“ stavů. To by v rozsáhlejších přechodových grafech s větším počtem nedeterministických přechodů bylo zdlouhavé a proto si ukážeme metodu, která umožní vyšetřovat všechny možné výpočty současně.

V následujících schématickém obrázku budeme současně zkoumat všechna možná zpracování řetězce *bba* :



Čtyři „pozice“ v obrázku znázorňují stavy našeho automatu. Topologické uspořádání je stejné jako ve výše uvedeném přechodovém grafu, černými šipkami jsou označeny počáteční a koncové stavy. Žlutě zvýrazněné jsou stavy, v jednom z nichž se v daném kroku zpracování vstupního řetězce automat musí nacházet (bez ohledu na to, v jakém výpočtu se do toho stavu mohl dostat). Modré šipky ukazují přechody, díky nimž automat do zvýrazněného stavu mohl přejít.

V první (počáteční) „pozici“ jsou zvýrazněny počáteční stavy q_1 a q_2.

Druhá „pozice“ má zvýrazněny všechny stavy, ve kterých se automat může nacházet po zpracování znaku b, tedy stavy q_3 a q_4. Modré šipky naznačují, že znakem b mohl automat přejít do stavu q_3 ze stavu q_1 nebo přešel do stavu q_4 ze stavu q_2.
Třetí „pozice“ ukazuje, že přestože je automat nederministický, po zpracování řetězce bb bude jednoznačně ve stavu q_3, protože hodnoty přechodové funkce $\delta(q_3,b)$ a $\delta(q_4,b)$ jsou jednoznačné a jsou rovny $\{q_3\}$.
Ze čtvrté „pozice“ je zřejmé, že se zpracováním řetězce bba automat může dostat do stavu q_1 nebo q_3 (protože $\delta(q_3,a)=\{q_1,q_3\}$).

O tom, zda je řetezec akceptován, vypovídá to, zda je zvýrazněn některý z koncových stavů. V tom případě existuje požadová posloupnost stavů začínající počátečním a končící koncovým stavem. Tuto posloupnost dokážeme určit z cest složených z modrých šipek.

Z obrázku je tedy zřejmé, že

- prázdný řetězec e je akceptován (existuje stav q_1, který je zároveň počáteční i koncový)
- řetězec b je akceptován (díky stavu q_4)
- řetězec bb je zamítnut (stav q_3 není koncový)
- řetězec bba je akceptován (díky stavu q_1)

Zobecnění:

Zpracování libovolného řetězce nedeterminstickým konečným automatem lze popsat sekvencí „pozic“, která jednoznačně definuje, zda je řetězec akceptován či zamítnut. Těchto „pozic“ je konečný počet (liší se pouze tím, které stavy jsou zvýrazněny; může jich být maximálně 2^n-1, protože „pozice“, která by neměla zvýrazněný žádný stav, nemá smysl; n představuje počet stavů NKA). Přechody mezi „pozicemi“ jsou jednoznačné, jednoznačně je dána „počáteční pozice“. **To vše jsou ale vlastnosti deterministického konečného automatu.**

Závěr:

Deterministický a nedeterministický rozpoznávací konečný automat rozpoznávají tutéž třídu jazyků.

Jinak řečeno:

Ke každému nedeterministickému rozpoznávacímu konečnému automatu NKA existuje ekvivalentní deterministický konečný automat KA (tj. automat ve smyslu definice v 1.2).

2.12. Převod NKA na ekvivalentní KA

Z předchozího je zřejmé, že stavy ekvivalentního deterministického automatu budou odpovídat „pozicím“, tedy podmnožinám stavové množiny výchozího NKA, a že počáteční stav KA je jednoznačně určen množinou počátečních stavů NKA. Vrátíme se k automatu z předchozí kapitoly:

Příklad:

		a	b
↔	**q_1**	$\{q_1,q_2\}$	$\{q_3\}$
→	**q_2**	$\{q_3\}$	$\{q_4\}$
	q_3	$\{q_1,q_3\}$	$\{q_3\}$
←	**q_4**	$\{q_1\}$	$\{q_3\}$

Tabulku s přechodovu funkcí deterministického ekvivalentu začneme vytvářet od jeho počátečního stavu, předem ovšem nevíme, kolik řádků (tedy stavů) bude tabulka mít (víme pouze, že je počet stavů konečný). Tabulku vytváříme tak, že do ní postupně přidáváme řádky (tedy stavy deterministického ekvivalentu) tak, jak vznikají sjednocováním množin v příslušných řádcích tabulky NKA. Konkrétně

		a	b
→	**$\{q_1,q_2\}$**	$\{q_1,q_2,q_3\}$	$\{q_3,q_4\}$

Počáteční stav $\{q_1,q_2\}$ odpovídá množině počátečních stavů NKA, množina stavů $\{q_1,q_2,q_3\}$ je sjednocením množin $\{q_1,q_2\}$ a $\{q_3\}$ v řádcích q_1 a q_2 *a*-sloupce tabulky výchozího NKA, stejně tak $\{q_3,q_4\}$ je sjednocením množin $\{q_3\}$ a $\{q_4\}$ v řádcích q_1 a q_2 *b*-sloupce. Nové stavy $\{q_1,q_2,q_3\}$ a $\{q_3,q_4\}$ zapíšeme do záhlaví řádků jako nové stavy.

		a	b
→	**$\{q_1,q_2\}$**	$\{q_1,q_2,q_3\}$	$\{q_3,q_4\}$
	$\{q_1,q_2,q_3\}$		
	$\{q_3,q_4\}$		

…….. vytvoříme hodnoty v těchto řádcích tabulky jako sjednocení množin v řádcích q_1,q_2,q_3 (respektive q_3,q_4) tabulky NKA a „nově vzniklé“ stavy zapíšeme do záhlaví řádků:

		a	b
→	**$\{q_1,q_2\}$**	$\{q_1,q_2,q_3\}$	$\{q_3,q_4\}$
	$\{q_1,q_2,q_3\}$	$\{q_1,q_2,q_3\}$	$\{q_3,q_4\}$
	$\{q_3,q_4\}$	$\{q_1,q_3\}$	$\{q_3\}$
	$\{q_1,q_3\}$		
	$\{q_3\}$		

V tomto postupu pokračujeme do té doby, dokud „vznikají“ nové stavy, tedy množiny, které ještě nemají ve vytvářené tabulce „svůj řádek“:

	a	b
→ **$\{q_1,q_2\}$**	$\{q_1,q_2,q_3\}$	$\{q_3,q_4\}$
$\{q_1,q_2,q_3\}$	$\{q_1,q_2,q_3\}$	$\{q_3,q_4\}$
$\{q_3,q_4\}$	$\{q_1,q_3\}$	$\{q_3\}$
$\{q_1,q_3\}$	$\{q_1,q_2,q_3\}$	$\{q_3\}$
$\{q_3\}$	$\{q_1,q_3\}$	$\{q_3\}$

Zbývá označit koncové stavy. Budou jimi všechny stavy, které „v sobě obsahují“ některý z koncových stavů výchozího NKA, tedy q_1 nebo q_4 . Pro zjednodušení bývá ještě zvykem stavy (tj. podmnožiny množiny Q) přejmenovat jednopísmennými názvy:

	a	*b*
↔ ***A*** $\{q_1,q_2\}$	*B* $\{q_1,q_2,q_3\}$	*C* $\{q_3,q_4\}$
← ***B*** $\{q_1,q_2,q_3\}$	*B* $\{q_1,q_2,q_3\}$	*C* $\{q_3,q_4\}$
← ***C*** $\{q_3,q_4\}$	*D* $\{q_1,q_3\}$	*E* $\{q_3\}$
← ***D*** $\{q_1,q_3\}$	*B* $\{q_1,q_2,q_3\}$	*E* $\{q_3\}$
E $\{q_3\}$	*D* $\{q_1,q_3\}$	*E* $\{q_3\}$

Přesně můžeme obecný vztah mezi NKA a ekvivalentním KA popsat takto:

$A=(Q,\Sigma,\delta,S,F)$ je nedeterministický rozpoznávací konečný automat

$A'=(Q',\Sigma,\delta',S,F')$ je ekvivalentní deterministický rozpoznávací konečný automat

Pak platí: $Q'\subseteq 2^Q$,

$$\delta':Q'\times\Sigma\to Q',\quad \delta'(K,x)=\bigcup_{q\in K}\delta(q,x)\quad \forall K\in Q',\ \forall x\in\Sigma$$

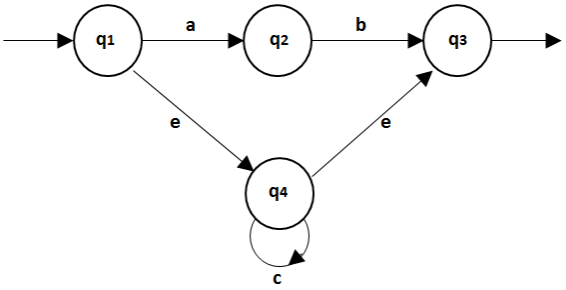
$$F'=\{K\mid K\in Q'\ \wedge\ K\cap F\neq\emptyset\}$$

Uvedený postup pro převod NKA ma (D)KA je nutné modifikovat v případě, že se v přechodovém grafu NKA vyskytnou e-hrany. Nejdříve je třeba rozšířit přechodovou tabulku NKA v tom smyslu, že se hodnoty přechodové funkce (tj. podmnožiny stavové množiny v tabulce) rozšíří o všechny *e-následníky*, tj. o další stavy, do nichž se automat může dostat samovolnými přechody cestami složenými z e-hran.

Příklad:

		a	b	C	e
→	**q_1**	$\{q_2\}$	∅	∅	$\{q_4\}$
	q_2	∅	$\{q_3\}$	∅	∅
←	**q_3**	∅	∅	∅	∅
	q_4	∅	∅	$\{q_4\}$	$\{q_3\}$

Pro názornost znázorníme i přechodový graf:



Jaké řetězce jsou tímto NKA akceptovány:

- prázdný řetězec *e*
- řetězec *ab*
- všechny řetězce c^n kde $n > 0$

Z grafu je zřejmé, že e-následníky vrcholu q_1 jsou vrcholy q_4 a q_3 (cestou složenou výhradně z e-hran může automat samovolně přejít z q_1 přes q_4 až do q_3), e-následníkem vrcholu q_4 je vrchol q_3. U automatu se složitějším přechodovým grafem je výhodnější spočítat matici relace „x *má za e-následníka y*“ jako reflexivně tranzitivní uzávěr relace „*z x vede e-hrana do y*“.

Reflexivně tranzitivním uzávěrem relace $\boldsymbol{E}$ rozumíme relaci $\boldsymbol{E}^* = \bigcup_{i=0,..,\infty} \boldsymbol{E}^i$, kde relaci $\boldsymbol{E}^i$ interpretujeme jako $xE^iy \Leftrightarrow$ *z x do y vede cesta délky i složená výhradně z e-hran.* Je zřejmé, že v grafu automatu o n stavech je délka nejdelší neuzavřené cesty $< n$. Proto v našem příkladu stačí spočítat matici relace $\boldsymbol{E}^*$ jako $\boldsymbol{E}^* = \boldsymbol{I} \cup \boldsymbol{E} \cup \boldsymbol{E^2} \cup \boldsymbol{E^3}$:

E	q_1	q_2	q_3	q_4
q_1	0	0	0	1
q_2	0	0	0	0
q_3	0	0	0	0
q_4	0	0	1	0

E^2	q_1	q_2	q_3	q_4
q_1	0	0	1	0
q_2	0	0	0	0
q_3	0	0	0	0
q_4	0	0	0	0

E^3	q_1	q_2	q_3	q_4
q_1	0	0	0	0
q_2	0	0	0	0
q_3	0	0	0	0
q_4	0	0	0	0

Pak z řádků matice relace E^* snadno pro každý stav určíme množinu e-následníků:

E^*	q_1	q_2	q_3	q_4
q_1	1	0	1	1
q_2	0	1	0	0
q_3	0	0	1	0
q_4	0	0	1	1

	e-následníci
q_1	$\{q_1,q_3,q_4\}$
q_2	$\{q_2\}$
q_3	$\{q_3\}$
q_4	$\{q_3,q_4\}$

Množiny e-následníků nyní použijeme

- k vytvoření ekvivalentní přechodové tabulky NKA bez e-hran tím způsobem, že stavy v množinách, určujících hodnoty přechodové funkce nahradíme všemi jejich e-následníky
- k vytvoření počátečního stavu deterministického ekvivalentu, kam pro každý počáteční stav výchozího NKA dáme všechny jeho e-následníky

Ekvivalentní přechodová tabulka NKA bez e-hran:

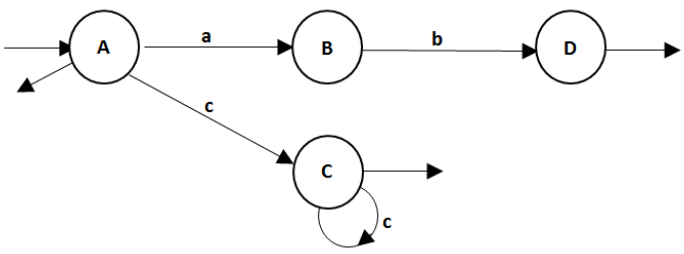
		a	b	c
→	q_1	$\{q_2\}$	∅	∅
	q_2	∅	$\{q_3\}$	∅
←	q_3	∅	∅	∅
	q_4	∅	∅	$\{q_3,q_4\}$

Počáteční stav deterministického ekvivalentu: $\{q_1,q_3,q_4\}$

Dál postupujeme stejně jako v případě NKA bez e-hran s tím, že místo s výchozí tabulkou NKA pracujeme s ekvivalentní přechodovou tabulkou bez e-hran. Výsledný (D)KA:

		A	b	c
↔	A $\{q_1, q_3, q_4\}$	B $\{q_2\}$	∅	C $\{q_3, q_4\}$
	B $\{q_2\}$	∅	D $\{q_3\}$	∅
←	C $\{q_3, q_4\}$	∅	∅	C $\{q_3, q_4\}$
←	D $\{q_3\}$	∅	∅	∅

Pro názornost nakreslíme i přechodový graf tohoto automatu:



Je zřejmé, že tento KA akceptuje právě tyto řetězce:

- prázdný řetězec *e*
- řetězec *ab*
- všechny řetězce c^n kde $n > 0$,

tedy stejný jazyk jako výchozí NKA.

2.13. Využití (D)KA a NKA při návrhu syntaktického analyzátoru regulárního jazyka

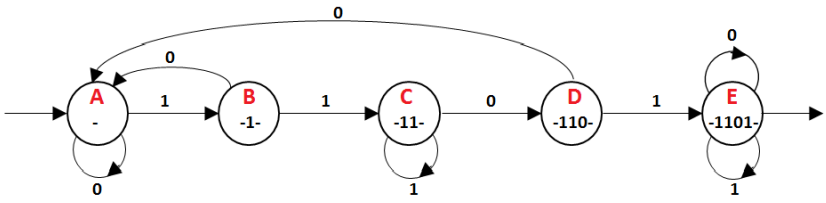
Příklady návrhu analyzátorů regulárních jazyků definovaných společnou vlastností řetězců jsme uvedli v kapitole 2.4. V případech, kdy je společnou vlastností řetězců jazyka přítomnost/nepřítomnost daného podřetězce ve specifikované části řetězce, lze přechodový graf deterministického automatu zkonstruovat intuitivně. Pokud je společná vlastnost formulována jako splnění určitého počtu z několika podmínek, je standardním postupem vytvoření několika „parciálních" automatů, z nichž každý rozhodne o splnění/nesplnění jedné z podmínek, a postupné vytvoření jejich kartézského součinu. Postup si ukážeme nejprve na příkladu.

Příklad: Vytvořte (D)KA, který bude rozpoznávat množinu všech řetězců nad abecedou {0,1}, které splňují současně následující podmínky

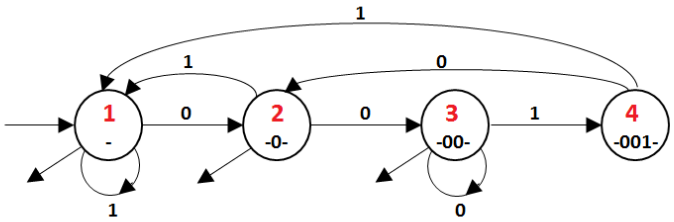
- řetězec obsahuje podřetězec 1101
- řetězec nekončí podřetězcem 001

Nejprve vytvoříme dva „parciální" automaty. Každý z nich bude rozpoznávatjazyk vyhovující jedné z podmínek. Z důvodů, které budou zřejmé později, pojmenujeme stavy jednoho automatu písmeny, druhého číslicemi.

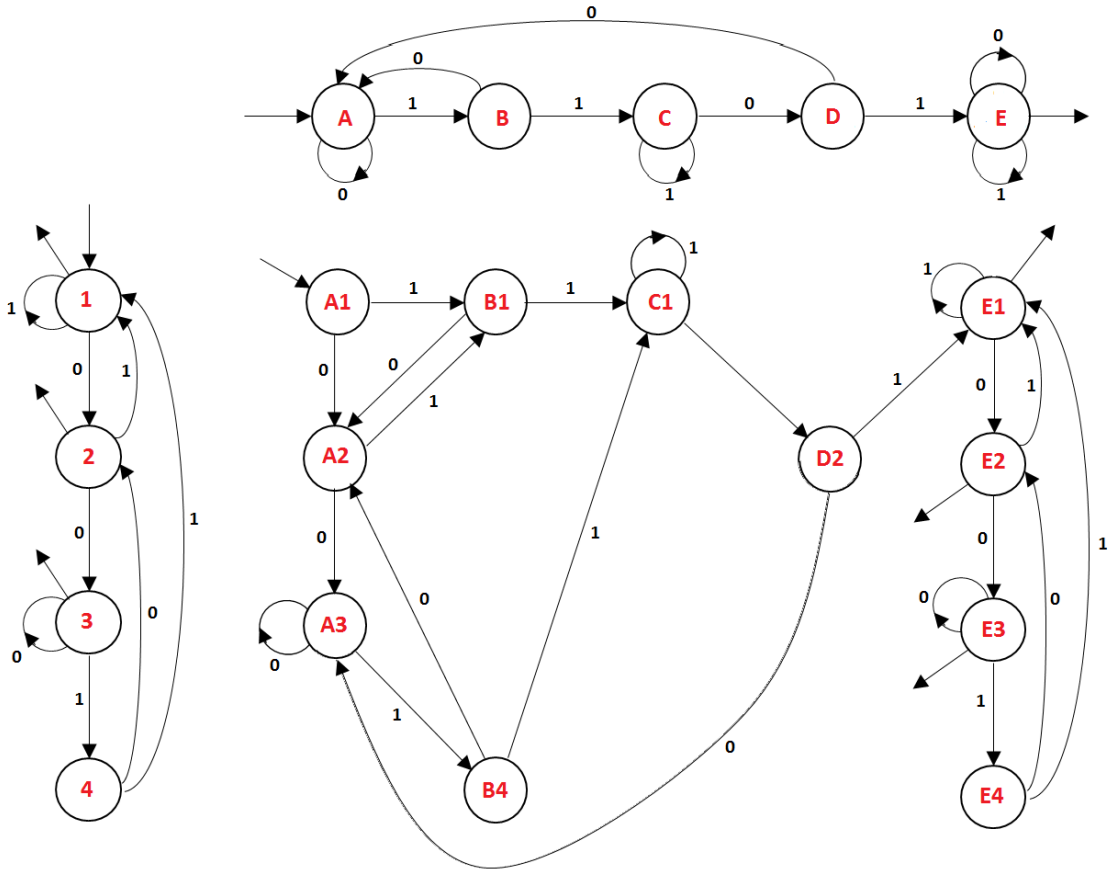
Automat, rozpoznávající řetězce splňující první podmínku („obshauje podřetězec 1101"):



Automat, rozpoznávající řetězce splňující druhou podmínku („nekončí 001“):



Kartézský součin obou automatů zakreslíme do ortogonálního uspořádání obou přechodových grafů. Níže zobrazený automat současně sleduje stavy obou parciálních automatů.



Kartézským součinem automatů $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ a $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ rozumíme automat $A = (Q, \Sigma, \delta, q_0, F)$ takový, že

- množina stavů $Q \subseteq Q_1 \times Q_2$, tedy $\forall q \in Q$ platí $q = (q_1, q_2)$, $q_1 \in Q_1, q_2 \in Q_2$
- počáteční stav $q_0 = (q_{01}, q_{02})$
- přechodová funkce $\delta(q, x) = (\delta_1(q_1, x), \delta_2(q_2, x))$ $\forall q \in Q$, $\forall x \in \Sigma$
- množina koncových stavů $F \subseteq Q$ taková, že

- $q = (q_1, q_2) \in F \Leftrightarrow q_1 \in F_1 \wedge q_2 \in F_2$, automat A pak rozpoznává průnik jazyků $L(A_1) \cap L(A_2)$
  nebo
- $q = (q_1, q_2) \in F \Leftrightarrow q_1 \in F_1 \vee q_2 \in F_2$, automat A pak rozpoznává sjednocení jazyků $L(A_1) \cup L(A_2)$

Praktický postup při vytváření kartézského součinu dvou automatů:

1. Přechodové grafy obou dílčích automatů nakreslíme tak, že všechny stavy umístíme do jedné přímky (stavy prvního automatu do přímky vodorovné, druhého automatu do přímky svislé) tak, aby vzniklo místo na zakreslení stavů výsledného automatu „v průsečících souřadnic" stavů parciálních automatů.
2. Do pozice odpovídající „průsečíku souřadnic" počátečních stavů obou dílčích automatů zakreslíme počáteční stav (ve výše uvedeném příkladu stav A1).
3. Pro každé písmeno vstupní abecedy zakreslíme do „průsečíků souřadnic" následujících stavů dílčích automatů následující stav automatu výsledného.
4. Bod 3 opakujeme tak dlouho, dokud v přechodovém grafu vznikají nové stavy. Je zřejmé, že tento postup někdy skončí (počet stavů výsledného automatu je konečný).

Tento postup lze samozřejmě analogicky a se stejným výsledkem provádět i nad tabulkami přechodových funkcí.

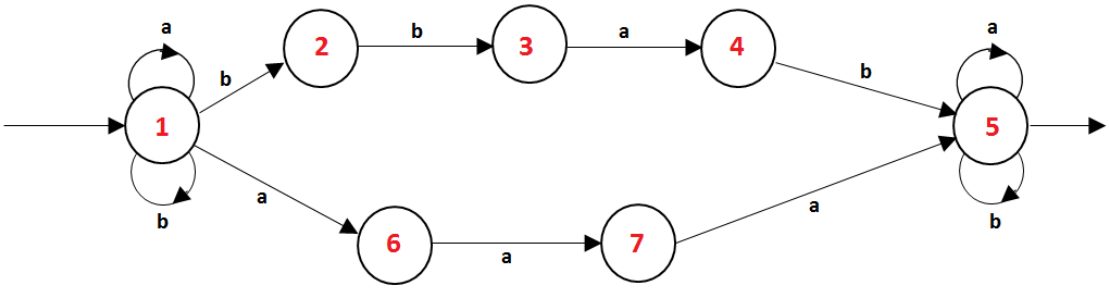
Je zřejmé, že výsledný automat získaný tímto postupem nemusí mít minimální počet stavů (jinak řečeno – může existovat ekvivalentní automat s menším počtem stavů). To není na závadu, existují algoritmy, které umožňují nalézt *redukovaný (minimální) automat*, tj. automat s minimálním počtem stavů, který daný jazyk rozpoznává.

Obecně bývá snazší navrhnout syntaktický analyzátor konkrétního regulárního jazyka jako nedeterministický konečný automat s tím, že poté bude nutné věnovat jisté úsilí jeho převodu na automat deterministický.

Příklad: Vytvořte NKA, který bude rozpoznávat množinu všech řetězců nad abecedou $\{a, b\}$, jež splňují alespoň jednu z následujících podmínek

- řetězec obsahuje podřetězec $bbab$
- řetězec obsahuje podřetězec aaa

Následující přechodový graf je založen na úvaze, že z počátečního do koncového stavu mohou vést pouze cesty ohodnocené řetězci $bbab$ nebo aaa . Hledaný podřetězec se může vyskytnout kdekoli ve vstupním řetězci, proto jsou v počátečním stavu 1 i koncovém stavu 5 smyčky pro oba vstupní symboly:

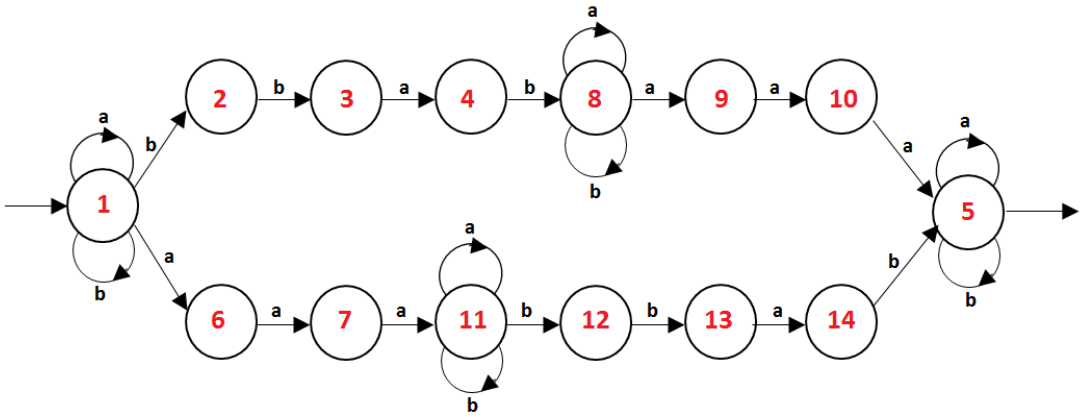


Modifikace tohoto přechodového grafu pro různé varianty zadání:

řetězec začíná podřetězcem $bbab$ nebo aaa - v přechodovém grafu nebudou dvě smyčky ve stavu 1

řetězec končí podřetězcem $bbab$ nebo aaa - v přechodovém grafu nebudou dvě smyčky ve stavu 5

řetězec splňuje obě podmínky , obsahuje tedy oba podřetězce $bbab$ i aaa :



- v přechodovém grafu jsou dvě paralelní cesty; cesta 1-2-3-4-8-9-10-5 se ve výpočtu uplatní v případě, že se podřetězec $bbab$ vyskytne dříve než podřetězec aaa, díky cestě 1-6-7-11-12-13-14-5 budou akceptovány řetězce, ve kterých se aaa vyskytne před $bbab$. Protože oba podřetězce nemusí následovat bezprostředně za sebou, musí být ve stavech 8 a 11 smyčky pro oba vstupní symboly.

Je zřejmé, že kdybychom automaty tohoto typu navrhovali jako deterministické, vyžadovalo by to mnohem více tvůrčího úsilí. Návrh ve formě nedeterministického automatu je zjevně jednodušší, nevyžaduje příliš kreativity, to je ale vykoupeno úsilím, jež je nutno věnovat následnému převodu na (D)KA. To lze ale řešit algoritmicky, resp. softwarovými nástroji.

2.14. Syntaktická analýza jazyků generovaných gramatikami G3L

V kapitole 2.7 jsme konstatovali, že regulární gramtiky jsou dvojího druhu: pravé lineární (G3P) a levé lineární (G3L). Zatím jsem si ukázali, jak vytvořit syntaktický analyzátor ke gramatikám G3P. Ke konstrukci analyzátoru jazyků generovaných gramatikami typu G3L využijeme reverzních gramatik a reverzních automatů.

Pod pojmem *reverzní gramatika* (k nějaké výchozí gramatice) budeme rozumět gramatiku, která má na levé i pravé straně přepisovacích pravidel (zrcadlově) převrácené řetězce z přepisovacích pravidel výchozí gramatiky. Reverzní gramatika bude generovat reverzní jazyk.

Přesněji:

Nechť $G = (N,T,S,P)$ je gramatika. Reverzní gramatikou $G^R = (N,T,S,P^R)$ ke gramatice G budeme rozumět takovou gramatiku, pro kterou platí $\alpha \to \beta \in P^R \Leftrightarrow \alpha^R \to \beta^R \in P$.
Pak $L(G^R) = \{w \mid w^R \in L(G)\}$. Platí $(G^R)^R = G$.

Příklad:

G_1: $S \to aS$
$aS \to bA \mid bba$
$A \to abb$

G^R: $S \to Sa$
$Sa \to Ab \mid abb$
$A \to bba$

Odvození řetězců z G :

$S \Rightarrow aS \Rightarrow bA \Rightarrow babb$

$S \Rightarrow aS \Rightarrow bba$

Odvození řetězců z G^R :

$S \Rightarrow Sa \Rightarrow Ab \Rightarrow bbab$

$S \Rightarrow Sa \Rightarrow abb$

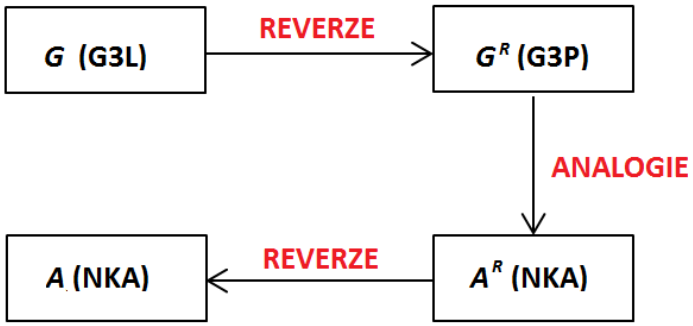
Příklad ilustruje, že reverze obou stran přepisovacích pravidel vede k tomu, že reverzní gramatika generuje reverzní jazyk.

Je zřejmé, že reverzí gramatiky typu G3L získáme gramatiku typu G3P. K té umíme sestrojit NKA, který bude rozpoznávat jazyk generovaný gramatikou G3P. Tento jazyk je ovšem reverzí jazyka generovaného výchozí gramatikou typu G3L (tedy reverzí jazyka, který máme rozpoznávat). Proto k vytvořenému NKA zkonstruujeme reverzní automat tím způsobem, že převrátíme orientaci všech šipek v přechodovém grafu, tj. změníme orientaci přechodových hran a navzájem zaměníme počáteční a koncové stavy.

Přesněji:

Nechť $A=(Q,\Sigma,\delta,S,F)$ je nedeterministický rozpoznávací konečný automat. Reverzním automatem $A^R=(Q,\Sigma,\delta^R,S^R,F^R)$ k automatu A budeme rozumět takový automat, pro který platí $q'\in\delta^R(q,x)\Leftrightarrow q\in\delta\ (q',x)\ \ \forall q,q'\in Q\ ,\ \forall x\in\Sigma;\ \ S^R=F\ ,\ \ F^R=S$.
Pak $L(A^R)=\{w\,|\,w^R\in L(A)\}$. Platí $(A^R)^R=A$.

Konstrukci syntaktického analyzátoru ke gramatice typu G3L pak lze znázornit takto:



2.15. Regulární množiny a regulární výrazy

Pojem regulární výraz je informatikům pracujícím v unixovském prostředí známý ze „skriptovacích“ jazyků, kde usnadňují zpracování textu (vyhledávání podřetězců, jejich extrakce či nahrazení jinými podřetězci, atd). V současnosti používané přístupy k regulárním výrazům jsou rozšířením konceptu, který publikoval v 50. letech 20. století americký matematik S. C. Kleene v práci o formalizaci popisu regulárních jazyků.

Kleeneho přístup k popisu regulárního jazyka (v jeho terminologii *regulární množiny*), je založen na principu, kdy definuje elementární regulární množiny jako základní stavební kameny, z nichž se regulárními operacemi vytvářejí další regulární množiny.

Regulární množina nad abecedou Σ je definována rekurzivně takto

1. $\emptyset$ je regulární množina
2. $\{e\}$ je regulární množina
3. $\{a\}$ je regulární množina $\forall\ a\ \in\ \Sigma$
4. Jsou-li P a Q regulární množiny, pak
   a. $P\cup Q$ je regulární množina
   b. $P\cdot Q$ je regulární množina
   c. P^* a Q^* jsou regulární množiny
5. Neexistují žádné jiné regulární množiny.

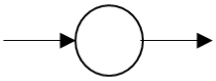
Poznámky k definici regulární množiny:

1. Z definice je zřejmé, že každá regulární množina je vytvořena z (elementárních) regulárních množin $\emptyset, \{e\}, \{a\}$ aplikací konečného počtu (regulárních) operací sjednocení $(\cup)$, zřetězení $(\cdot)$ a iterace $(*)$.
2. Ke každé regulární množině existuje konečný automat, jenž ji rozpoznává. Pojem regulární množina je tedy jen synonymem k pojmu regulární jazyk.

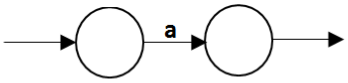
Jak vypadají konečné automaty rozpoznávající elementární regulární množiny?

$L(A) = \emptyset$

$L(A) = \{e\}$

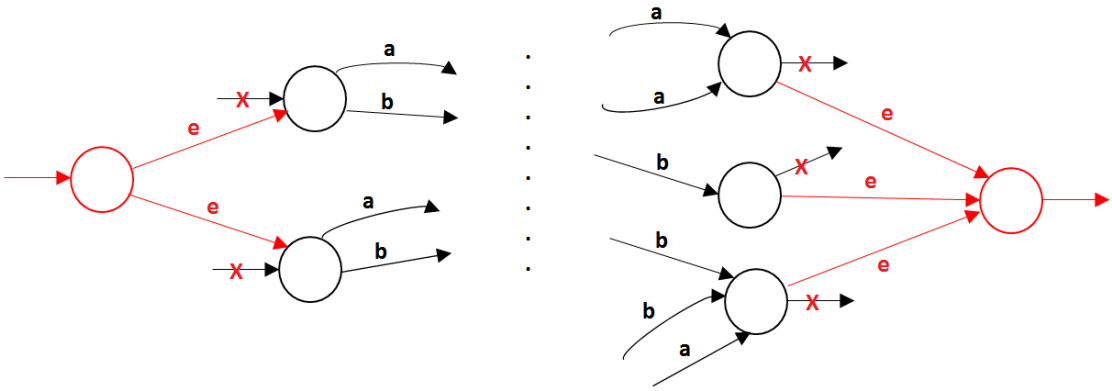


$L(A) = \{a\}$

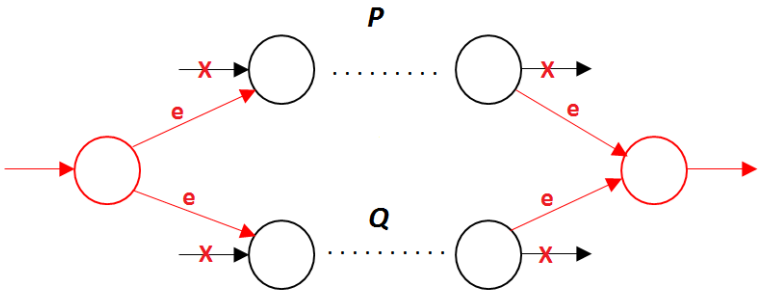


Jak konstruovat automaty rozpoznávající regulární mnořiny vytvořené regulárními operacemi?

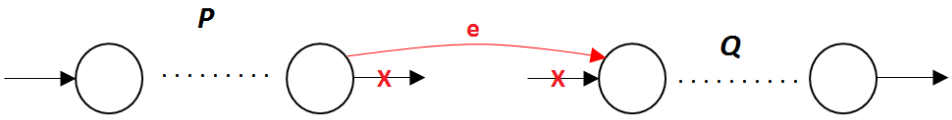
Bez újmy na obecnosti předpokládejme, že NKA má právě jeden počáteční a právě jeden koncový stav. Pokud tomu tak není, dosáhneme toho přidáním nového počátečního stavu, nového koncového stavu a e-hran, aniž by se tím změnila množina akceptovaných řetězců:
Ilustrace:



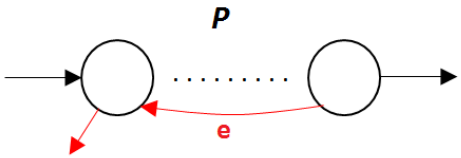
Přechodový graf automatu, akceptujícího sjednocení regulárních množin $P \cup Q$:



Přechodový graf automatu, akceptujícího zřetězení regulárních množin $P \cdot Q$:



Přechodový graf automatu, akceptujícího iteraci regulární množiny P^*:



Regulární výrazy *označují* (*reprezentují*, *popisují*) regulární množiny pomocí řetězců. Umožňují každou regulární množinu jednoznačně popsat jedním řetězcem tvořeným znaky z abecedy Σ, operátory $+$, $\cdot$ a * a závorkami (a) . Operátor $\cdot$ se vynechává, podobně jako operátor násobení symbolických aritmetických výrazů.

Regulární výraz nad abecedou Σ je definován rekurzivně takto

1. $\emptyset$ je regulární výraz označující regulární množinu $\emptyset$
2. e je regulární výraz označující regulární množinu $\{e\}$
3. a je regulární výraz je regulární množinu $\{a\}$ $\forall\, a \in \Sigma$
4. Jsou-li p a q regulární výrazy označující regulární množiny P a Q, pak
   a. $p + q$ je regulární výraz označující regulární množinu $P \cup Q$
   b. $p \cdot q$ je regulární výraz označující regulární množinu $P \cdot Q$
   c. p^* a q^* jsou regulární výrazy označující regulární množiny P^* a Q^*
5. Neexistují žádné jiné regulární výrazy.

Terminologie používaná pro vztah mezi regulárním výrazem a regulární množinou: Regulární výraz R *označuje* regulární množinu A, regulární množina A *je hodnotou* regulárního výrazu R. Značení: $A = \|R\|$

Příklady regulárních výrazů nad abecedou $\{a, b\}$:

Regulární výraz	hodnota regulárního výrazu (slovní popis)
baa^*	všechna slova začínající znakem b , následovým pouze znaky a
$a^*ba^*ba^*$	všechna slova obsahující právě dva znaky b
$(a+b)^*$	všechna slova
$(a+b)^*bb$	všechna slova končící dvěma znaky b

Na základě definice regulární množiny a vlastností kartézského součinu z kapitoly 2.13 můžeme zformulovat následující tvrzení:

Třída regulárních jazyků je uzavřená vůči operacím sjednocení, zřetězení, průniku a iterace.

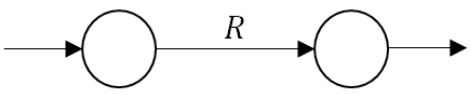
Jinak řečeno: Sjednocení, zřetězení, průnik libovolných dvou regulárních jazyků je regulární jazyk. Iterace libovolného regulárního jazyka je regulární jazyk.

2.16. Konstrukce konečného automatu z regulárního výrazu

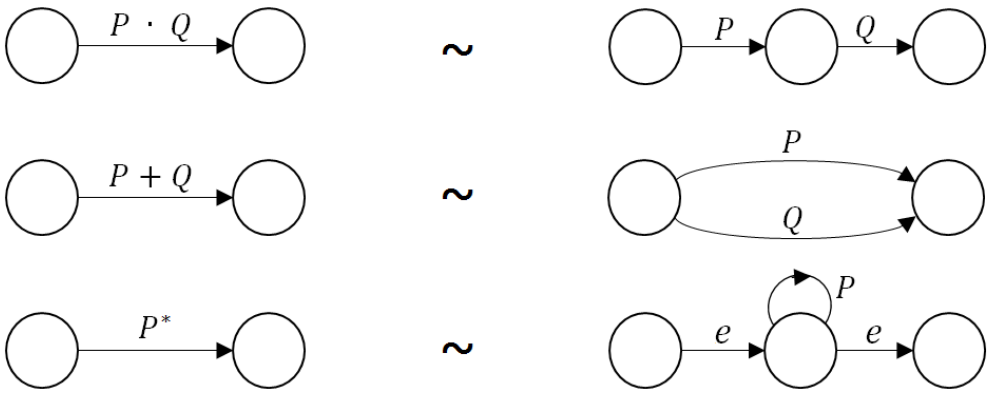
Jak z regulárního výrazu R vytvořit konečný automat A, který bude jazyk popsaný tímto výrazem akceptovat?

Řešením je postupný rozklad regulárního výrazu a související transformace *zobecněného přechodového grafu*. Pod pojmem *zobecněný přechodový graf* rozumíme přechodový graf, u něhož připustíme ohodnocení hran nejen symboly vstupní abecedy $a \in \Sigma$ a symbolem e ale i regulárními výrazy nad abecedou Σ.

Výchozí zobecněný přechodový graf bude mít pouze dva stavy - počáteční a koncový. Z počátečního stavu do koncového povede hrana ohodnocená regulárním výrazem R:



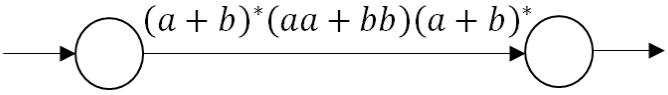
Zobecněný přechodový graf dále budeme transformovat podle těchto pravidel :



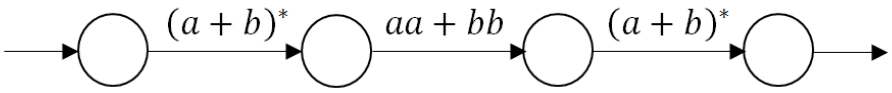
Protože je regulární výraz vytvořen pouze regulárními operacemi a pro každou regulární operaci je dáno transformační pravidlo, dojdeme tímto postupem k přechodovému grafu, v němž budou všechny hrany ohodnoceny symboly abecedy $a \in \Sigma$ nebo symbolem e , dojdeme k tedy k přechodovému grafu nedeterministického konečného automatu.

Příklad: Vytvořte konečný automat rozpoznávající jazyk $(a+b)^*(aa+bb)(a+b)^*$.

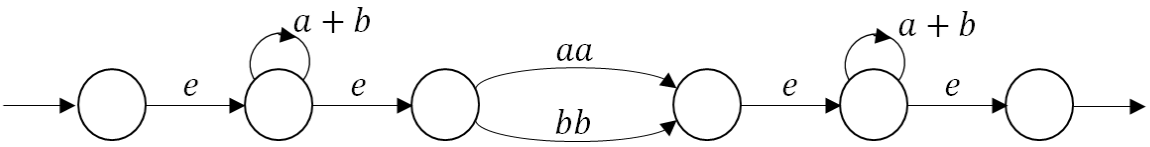
1. krok (vytvoření zobecněného přechodového grafu o dvou stavech)



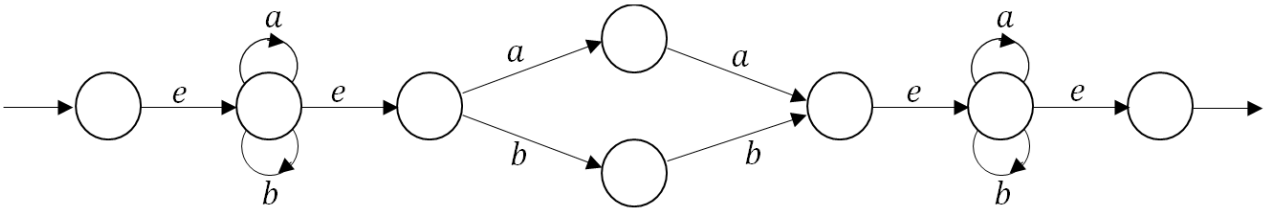
2. krok (transformace dvou zřetězení)



3. krok (transformace dvou iterací a součtu)



4. krok (transformace dvou součtů a dvou zřetězení)



Po 4. kroku je každá hrana ohodnocena písmenem vstupní abecedy nebo symbolem e . Transformace je tedy skončena, výsledkem je přechodový graf konečného automatu.

Automat vytvořený tímto postupem je obecně nedeterministický, lze vytvořit jeho deterministický ekvivalent (viz kapitola 2.12).

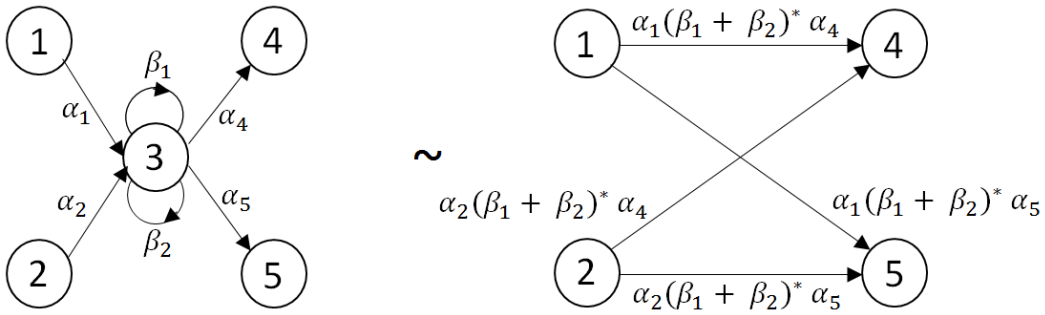
2.17. Popis jazyka akceptovaného automatem regulárním výrazem

Jak z konečného automatu A vytvořit regulární výraz R, který bude popisovat jazyk akceptovaný tímto automatem?

Bez újmy na obecnosti můžeme předpokládat automat s jedním počátečním a jedním koncovým stavem (viz kapitola 2.15). Postupně budeme eliminovat všechny hrany a vrcholy přechodového grafu kromě počátečního stavu, koncového stavu a hrany mezi nimi. Během procesu eliminace budou modifikovány hrany a jejich ohodnocení regulárními výrazy. Tímto způsobem postupně zkonstruujeme sekvenci zobecněných přechodových grafů, které akceptují stejnou množinu slov. Proces končí, když v grafu zbudou jen dva stavy (počáteční a koncový) a jedna hrana z počátečního do koncového stavu. Hledaný regulární výraz je pak ohodnocením této hrany.

Schéma eliminace hran:

Schéma eliminace vrcholů:



Levá část obrázku představuje výsek z přechodového grafu, který obsahuje eliminovaný vrchol, všechny jeho sousedy (tedy všechny vrcholy z nichž nebo do nichž vede hrana z eliminovaného vrcholu) a všechny hrany koincidující s eliminovaným vrcholem.

Při eliminaci vrcholu 3 musíme nejprve zmapovat všechny cesty mezi sousedními vrcholy, které přes eliminovaný vrchol vedou, a popsat je regulárními výrazy:

Cesty přes eliminovaný vrchol 3:

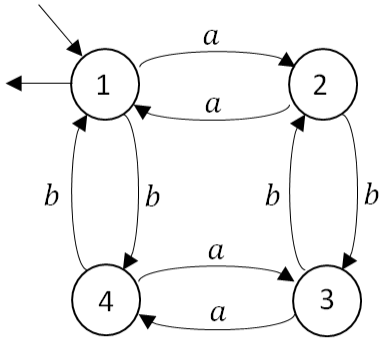
$1 \rightarrow 4$	$\alpha_1(\beta_1 + \beta_2)^* \alpha_4$
$1 \rightarrow 5$	$\alpha_1(\beta_1 + \beta_2)^* \alpha_5$
$2 \rightarrow 4$	$\alpha_2(\beta_1 + \beta_2)^* \alpha_4$
$2 \rightarrow 5$	$\alpha_2(\beta_1 + \beta_2)^* \alpha_5$

Žádné další orientované cesty přes eliminovaný vrchol 3 nevedou.

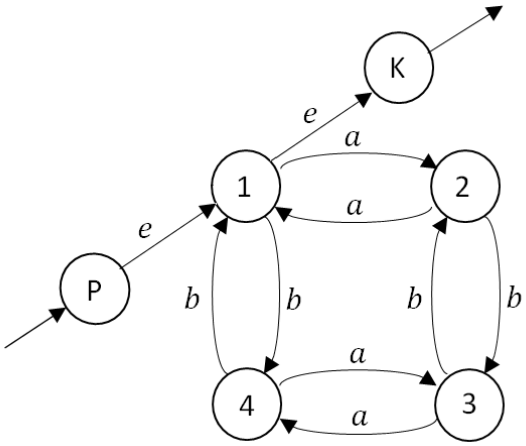
Pravá část obrázku představuje stejný výsek grafu po provedení eliminace vrcholu 3. Vrchol 3 se v něm tedy již nevyskytuje, ale doplněním nových hran ohodnocených výše uvedenými regulárními výrazy jsou „zachovány cesty“, které přes eliminovaný vrchol vedly. Množina řetězců akceptovaných oběma zobecněnými přechodovými grafy je stejná.

Příklad: Konečný automat A nad vstupní abecedou $\{a, b\}$ akceptuje řetězce obsahující sudý počet znaků a a sudý počet znaků b (nulu považujeme za sudé číslo). Popište jazyk $L(A)$ regulárním výrazem.

Přechodový graf automatu A :

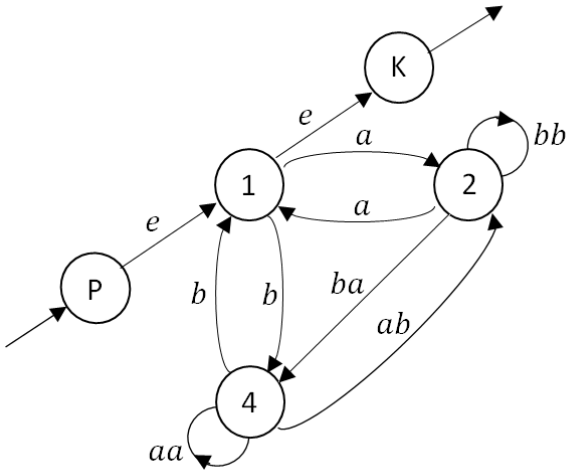


Krok 1 – vytvoření samostatného počátečního a samostatného koncového stavu :



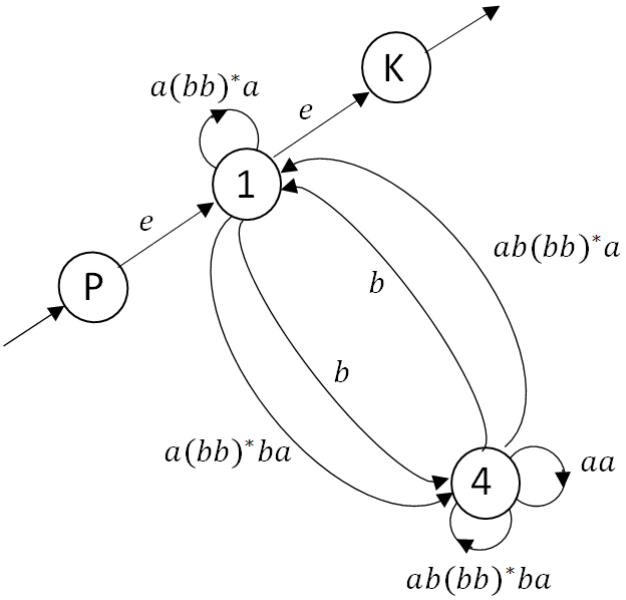
Krok 2 – eliminace vrcholu 3:

Cesty přes vrchol 3: 2 → 2: bb , 2 → 4: ba
4 → 2: ab , 4 → 4: aa



Krok 3 – eliminace vrcholu 2:

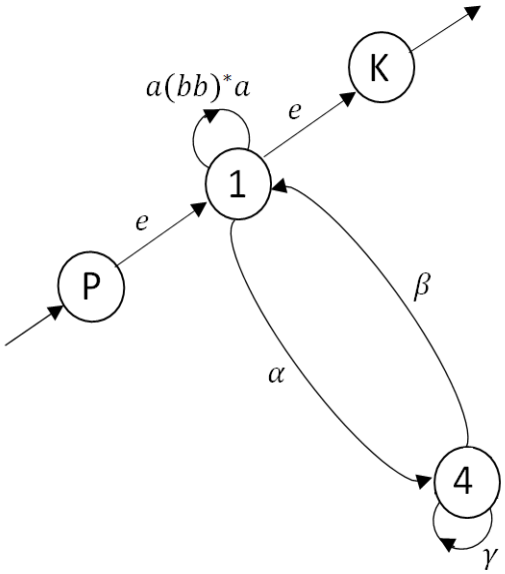
Cesty přes vrchol 2: $1 \rightarrow 1$: $a(bb)^*a$, $1 \rightarrow 4$: $a(bb)^*ba$,
$4 \rightarrow 1$: $ab(bb)^*a$, $4 \rightarrow 4$: $ab(bb)^*ba$



Krok 4 – eliminace paralelních hran a smyček:

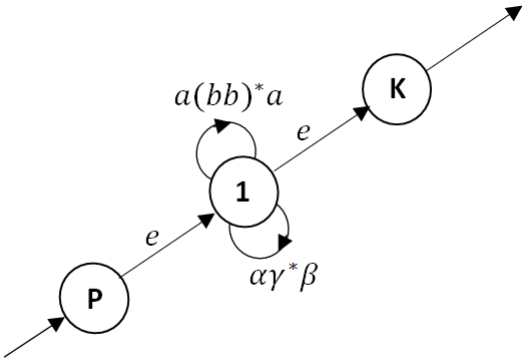
Hrany mezi vrcholy 4 a 1: $1 \rightarrow 4$: $b + a(bb)^*ba$ $(= \alpha)$
$4 \rightarrow 1$: $b + ab(bb)^*a$ $(= \beta)$
$4 \rightarrow 4$: $aa + ab(bb)^*ba$ $(= \gamma)$

Pro zjednodušení popisu hran jsme označili dílčí regulární výrazy symboly α, β, γ .



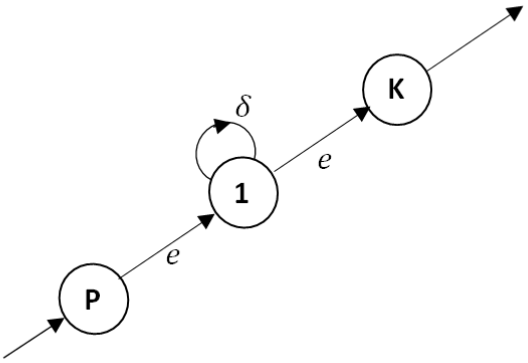
Krok 5 – eliminace vrcholu 4:

Cesty přes vrchol 4: 1 → 1: $\alpha\gamma^*\beta$



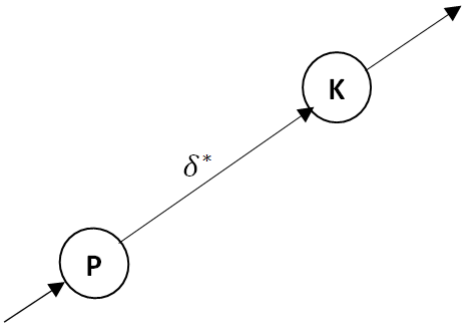
Krok 6 - eliminace paralelní smyčky v 1:

Smyčka ve vrcholu 1: 1 → 1: $a(bb)^*a + \alpha\gamma^*\beta = \delta$



Krok 7 - eliminace vrcholu 1:

Cesta přes vrchol 1: P → K: δ^* (viz kapitola 2.16).



Je zřejmé, že výsledný zobecněný přechodový graf akceptuje regulární množinu popsanou výrazem δ^* (viz kapitola 2.16).

Hledaný regulární výraz má tedy podobu

$$R = \delta^* = [a(bb)^*a + \alpha\gamma^*\beta)]^* =$$

$$= [a(bb)^*a + (b + a(bb)^*ba)(aa + ab(bb)^*ba)^*(b + ab(bb)^*a)]^*$$

Obecně platí, že tvar výsledného regulárního výrazu je ovlivněn pořadím eliminace vrcholů. Pokud zvolíme jiné pořadí eliminace vrcholů (a neuděláme při transformacích chybu), získáme regulární výraz jiný, ale ekvivalentní v tom smyslu, že bude popisovat stejný jazyk.

2.18. Regulární výrazy *regexp*

Kleeneho konceptu regulárních výrazů se úspěšně chopili tvůrci programovacích jazyků a zavedením matajazykových symbolů vytvořili z regulárních výrazů efektivní nástroj pro práci s textovými řetězci. Princip těchto regulárních výrazů (*regexp*, *regex*) zůstává stejný – popisují množiny řetězců. Lze je chápat jako vzor (resp. masku) pro textové řetězce, kterému programem zpracovávaný vstupní řetězec buď vyhovuje nebo nevyhovuje, od čehož se mohou odvíjet další akce. Nejčastější využití *regexp* je zjištění, zdali vstupní text vyhovuje zadanému regulárnímu výrazu, zjištění pozice ve vstupním textu, kde shoda s regulárním výrazem začíná nebo záměna textu v jednom z podvýrazů *regexp*. Regulární výrazy se například standardně používají ke kontrole dat zadávaných ve formulářích (e-mailová adresa, PSČ, bankovní spojení,…..) nebo k selekci informací z logových souborů při správě systémů.

Práci s regulárními výrazy umožňuje řada programovacích jazyků (C#, Java, Visual Basic .NET, Perl, PHP, Javascript aj.) i řada programů a funkcí v prostředí UNIX/LINUX.

Příklady metajazykových symbolů a operátorů používaných v *regexp* :

Metaznaky pro ukotvení pozice v řetězci				
Metaznak	**Význam metaznaku**	**Příklad regulárního výrazu**	**Příklady vyhovujících vstupních řetězců**	**Příklady nevyhovujících vstupních řetězců**
^	Pevný začátek řetězce	^Pos	Poslouchám Poseidon	Neposlouchej Porta
$	Pevný konec řetězce	rok$	nový rok Snížili úrok	Rok na vsi Dva roky poté Hrajte taroky

Metaznaky pro zástupné symboly a opakování				
Metaznak	**Význam metaznaku**	**Příklad regulárního výrazu**	**Příklady vyhovujících vstupních řetězců**	**Příklady nevyhovujících vstupních řetězců**
.	Libovolný znak (právě jeden)	t.p	step tipovat utopí	tis lep terpentýn
*	Předchozí znak se vyskytne 0x až ∞x	r*m	zarmoutit grrrrm! ataman	figura
+	Předchozí znak se vyskytne 1x až ∞x	s+t	pssst! kostival	blatník
?	Předchozí znak se vyskytne 0x až 1x	k?t	pakt port vektor	Vekktor
{*kolik*}	Předchozí znak se vyskytne právě *kolik x*	[0123456789]{3}	123 875 007	Abc 22 -135 1234
{*min,max*}	Předchozí znak se vyskytne *min x* až *max x*	Ps{*2,4*}t	Psst! Pssst! Psssst!	Pst! Pssssst!
Metaznaky pro výčtové množiny a řetězce				
Metaznak	**Význam metaznaku**	**Příklad regulárního výrazu**	**Příklady vyhovujících vstupních řetězců**	**Příklady nevyhovujících vstupních řetězců**
[]	Libovolný znak z výčtu znaků v závorce	[0123456789]	-16 1 ks máslo Peugeot 308	jedno máslo
[*d-h*]	Libovolný znak z množiny znaků definované rozsahem od znaku d až po znak h (rozsahy lze řetězit)	[A-M]	Dovolená Malaga 156A23 B007	-254 agent 007 okresní přebor Pražský výběr
[^d-h]	Libovolný znak **mimo** množinu znaků definovanou rozsahem od znaku d až po znak h (rozsahy lze řetězit)	[^a-z0-9]	ALFA C&K VOCAL	ALFa C&K F.J.1.
()	Celý řetězec v závorce	(slov)	oslovený slovenský máslový	složený úlovek solventní

Další metaznaky				
Metaznak	**Význam metaznaku**	**Příklad regulárního výrazu**	**Příklady vyhovujících vstupních řetězců**	**Příklady nevyhovujících vstupních řetězců**
|	Odděluje variantní podvýrazy	a(bec|port)	abeceda aportuje abecední	aerobik portace abertam
\	Následující metaznak není metaznak, ale datový znak	*	*Jakýkoli řetězec obsahující znak* ¨*¨	*Jakýkoli řetězec neobsahující znak* ¨*¨
Předdefinované množiny znaků				
\d \D \w \W \s \S	Libovolná desítková číslice [0–9] Libovolná „nečíslice" [^\d] Libovolný alfanumerický znak [a–zA–Z 0–9] Libovolný nealfanumerický znak [^\w] Libovolný „prázdný znak" [\ \t\n\r] Libovolný „neprázdný znak" [^\S]			

Výše uvedený přehled shrnuje nejběžnější metasymboly a operátory, které povětšinou fungují ve většině „skriptovacích jazyků", je ovšem nutné počítat s tím, že se jednotlivé implementace mohou v detailech lišit. Cílem této kapitoly je, aby si čtenář mohl o praktickém použití regulárních výrazů udělat jasnější představu; pro reálnou práci v konkrétním programovacím nástroji se zájemce musí obrátit ke zdrojům týkající se konkrétní implementace.

Příklady: Regulární výraz pro ověření správnosti syntaxe e-mailové adresy:

[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}

Regulární výraz pro ověření správnosti syntaxe poštovního směrovacího čísla (s možnou mezerou mezi trojčíslím a dvojčíslím):

[0-9]3} ?[0-9]{2}

Regulární výraz pro ověření správnosti syntaxe telefonního čísla z ČR nebo SR (s možnými mezerami mezi trojčíslími:

(\+420|\+421)?[0-9]{3} ?[0-9]{3} ?[0-9]{3}$

2.19. Nerodova věta, ekvivalentní reprezentace regulárních jazyků.

V kapitole 1.4 jsme konstatovali, že zobecněná přechodová funkce δ^* deterministického konečného automatu na množině všech vstupních řetězců Σ^* jednoznačně definuje rozklad o n blocích (kde n označuje počet stavů KA). Každý blok rozkladu odpovídá jednomu stavu automatu a patří do něj právě ty vstupní řetězce, které převádí automat z počátečního stavu do stavu odpovídajícího konkrétnímu bloku. Tento rozklad množiny všech vstupních řetězců jednoznačně indukuje relaci ekvivalence na množině všech vstupních řetězců. Ve stejné třídě ekvivalence jsou ty vstupní řetězce, které z počátečního stavu převedou automat do stejného stavu. Tuto ekvivalenci značíme operátorem $\approx$. V důsledku toho, že je funkce δ^* tranzitivním uzávěrem přechodové funkce δ, platí $(u \approx v) \Rightarrow (uw \approx vw)\ \forall u, v, w \in \Sigma^*$, ekvivalence $\approx$ je tedy pravou kongruencí a vzhledem ke konečnému počtu stavů automatu je pravou kongruencí konečného indexu (tj. má konečný počet tříd).

Nerodova věta: Formální jazyk L nad konečnou abecedou je regulární právě tehdy, pokud existuje pravá kongruence konečného indexu taková, že jazyk L je sjednocením vybraných tříd této kongruence.

Poznámka k Nerodově větě: Je zřejmé, že „vybranými třídami" jsou takové třídy ekvivalence, které odpovídají koncovým stavům konečného automatu, jenž jazyk L rozpoznává.

Nejdůležitější závěry z kapitol, týkajících se regulárních jazyků zakončíme důležitým konstatováním.

Následující tvrzení jsou ekvivalentní:

Tvrzení 1: Jazyk L nad abecedou Σ je regulární.

Tvrzení 2: Na množině Σ^* existuje pravá kongruence konečného indexu takové, že L je sjednocením některých tříd této kongruence.

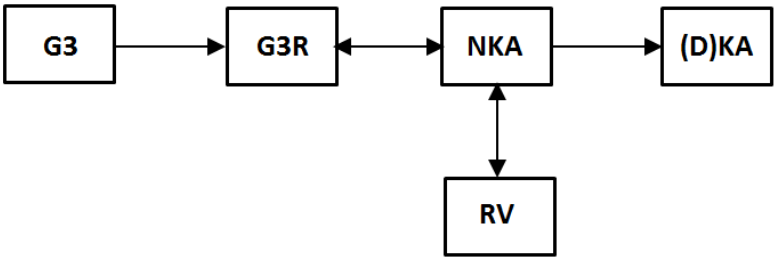
Tvrzení 3: Existuje nedeterministický konečný automat A, který jazyk L rozpoznává, tedy $L(A) = L$.

Tvrzení 4: Existuje deterministický konečný automat A', který jazyk L rozpoznává, tedy $L(A') = L$.

Tvrzení 5: Existuje regulární gramatika G, která jazyk L generuje, tedy $L(G) = L$.

Tvrzení 6: Existuje regulární výraz R, který jazyk L popisuje, tedy $\|R\| = L$.

Následující obrázek znázorňuje převody mezi různými reprezentaceni regulárních jazyků, které byly zmíněny v tomto textu:



Šipky ve sněru od G3P ke G3, respektive od (D)KA k NKA nemají smysl, neboť G3R je zvláštním případem G3 a (D)KA je zvláštním případem NKA.

Použitá literatura:

Demlová, M., Koubek, V.: Algebraická teorie automatů, SNTL 1990.
Friedmann, A. D., Menon, P. R.: Teorie a návrh logických obvodů, SNTL 1983.
Gries, D.: Kompilátory číslicových počítačov, ALFA SNTL, 1981
Chytil, M. : Gramatiky a automaty, SNTL 1983
Manna, Z.: Matematická teorie programů, SNTL 1981
Mareš, J. Jazyky, gramatiky a automaty. ČVUT Praha, 2004.
Melichar, B.; at al. Jazyky a překlady Cvičení. Praha, ČVUT, 2004.