Obsah

Předmluva

Tento materiál se snaží být v jistém smyslu „uceleným“ (ale určitě ne vyčerpávajícím) textem o *deterministických konečných automatech*, psaným sice „z pozic“ teoretické informatiky, ale pro studenty inženýrských, zejména informatických oborů. Texty pokrývající další partie přednášené v předmětu KIV/TI v podobné formě budou postupně následovat.

Text není pouze záznamem přednášek, rozsahem i podrobnostmi přesahuje objem, který obvykle bývá k daném tématu odpřednášen (a který také bývá zkoušen).

Při tvorbě tohoto textu jsem se snažil 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 řecká 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,
- v poslední kapitole jsem se snažil 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 a studenti se s touto látkou v dalším studiu ještě setkají; nicméně považoval jsem za vhodné, aby ti hloubavější nebo lépe motivovaní 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 prostudováním ovšem student může 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 některých teoretických partií, které mohou být pro posluchače při dnešní úrovni obecných matematických znalostí a dnešní frekvenci návštěv přednášek obtížnější.

Václav Vais
září 2015

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 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.

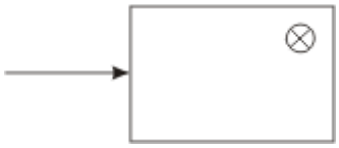
Základními prvky při popisu KA tedy budou *konečná množina stavů, konečná množina vstupních symbolů,* a *přechodová funkce,* která bude pro každý aktuální stav a každý vstupní symbol jednoznačně definovat následující stav. Je samozřejmé, že každé reálné zařízení musí svoji činnost začít v nějakém jednoznačně 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).

1.2. Matematický popis konečného automatu

1.2.1. Definice různých typů konečných automatů

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
$q_0 \in Q$ je počáteční stav
$\delta : Q \times \Sigma \to Q$ je přechodová funkce
$F \subseteq Q$ je množina koncových 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 nějaký rozklad množiny stavů

Rozkladem množiny stavů *{Qi}* v definici klasifikačního automatu se rozumí takový systém podmnožin množiny stavů *Q*, že se každý stav z *Q* nachází právě v jednom prvku *{Qi}* (prvky z *{Qi}* 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 existuje Moorův automat se stejným chováním a naopak. Existují algoritmy pro převod automatu jednoho typu na automat druhého 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“ všechnu 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 $\lambda : Q \times \Sigma \rightarrow O^*$, 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 aplika-

cích tento model ovšem uplatnění nenajde (problém synchronizace a časování).

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*. Později 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*.

1.2.2. 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 (Chytil):

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 $\Sigma = \{0,1\}$

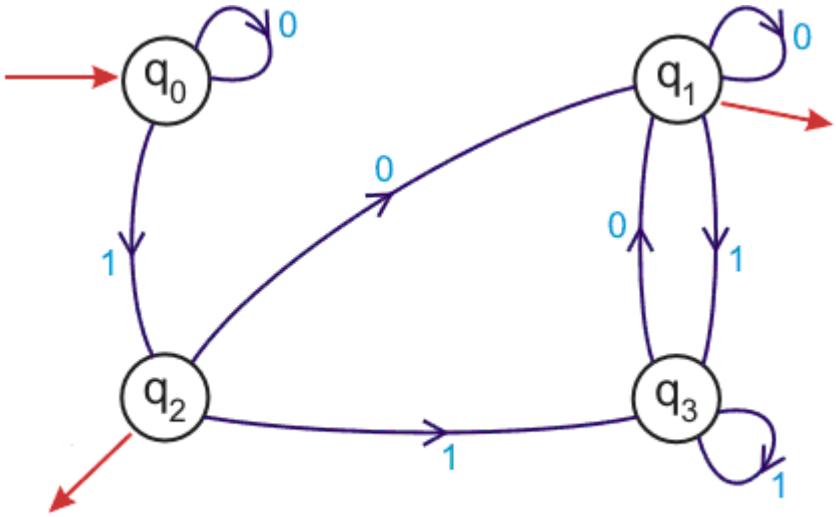
počáteční stav q_0 množina koncových stavů $F = \{q_1, q_2\}$

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 stavovým diagramem:

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.

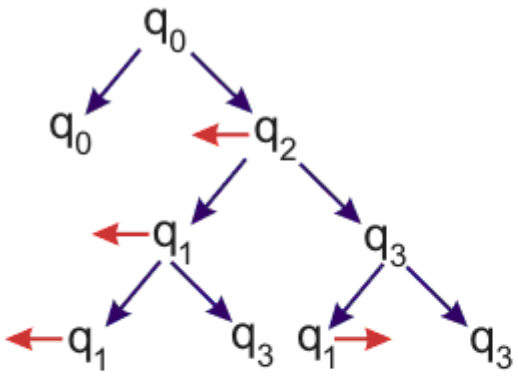
		0	**1**
→	**q_0**	q_0	q_2
←	**q_1**	q_1	q_3
←	**q_2**	q_1	q_3
	q_3	q_1	q_3

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. **Výstupní 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).

1.2.3.Chování konečného automatu

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

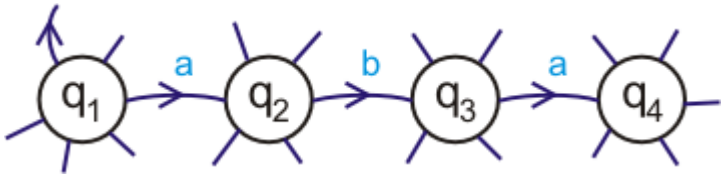
- zajímá nás, co je výsledkem zpracování konkrétního vstupního řetězce; k tomu lze přistoupit ze dvou různých hledisek a to
  - z pohledu vztahu mezi vstupním řetězcem a stavem, do něhož byl automat tímto řetězcem převeden (relace vstup – stav; tento pohled lze uplatnit u všech výše definovaných typů 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 celé vstupní řetězce, 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: $\delta : Q \times \Sigma \to Q$

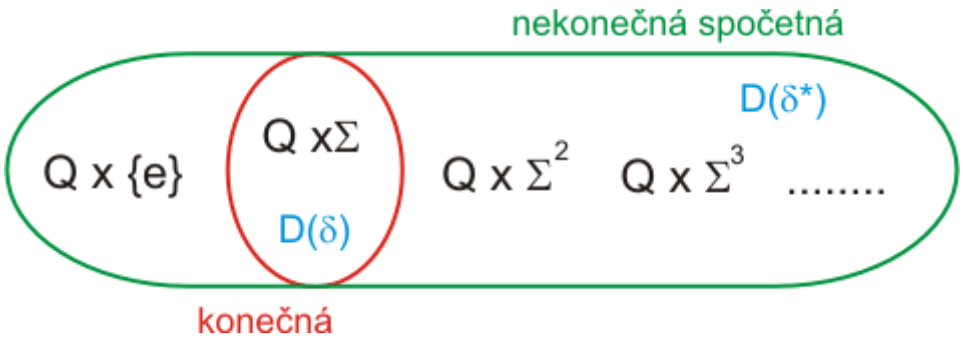
Zobecněná přechodová funkce: $\delta^* : Q \times \Sigma^* \to Q$

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). Nesmíme zapomenout na to, že i prázdný řetězec *e* je vstupním řetězcem 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 $\delta^*(q,a) = \delta(q,a) \quad \forall q \in Q, \forall a \in \Sigma$, 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 přecho-

dové 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 determinstický, 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:

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

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).

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

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

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
$T : \Sigma^* \rightarrow \{1,2,....,n\}$
kde $i \in N, i \leq n$ reprezentuje klasifikační třídu, do které je vstupní řetězec automatem zařazen.

Automat s výstupní funkcí transformuje vstupní řetězce na výstupní řetězce.
$T : \Sigma^* \rightarrow O^*$

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:

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

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

Překladová funkce β u Moorova automatu:

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

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

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 chování konečného automatu, konfigurace automatu

Chceme-li sledovat vývoj chování 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 chování 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.

1.3. Příklady z aplikačních oblastí

1.3.1. 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.

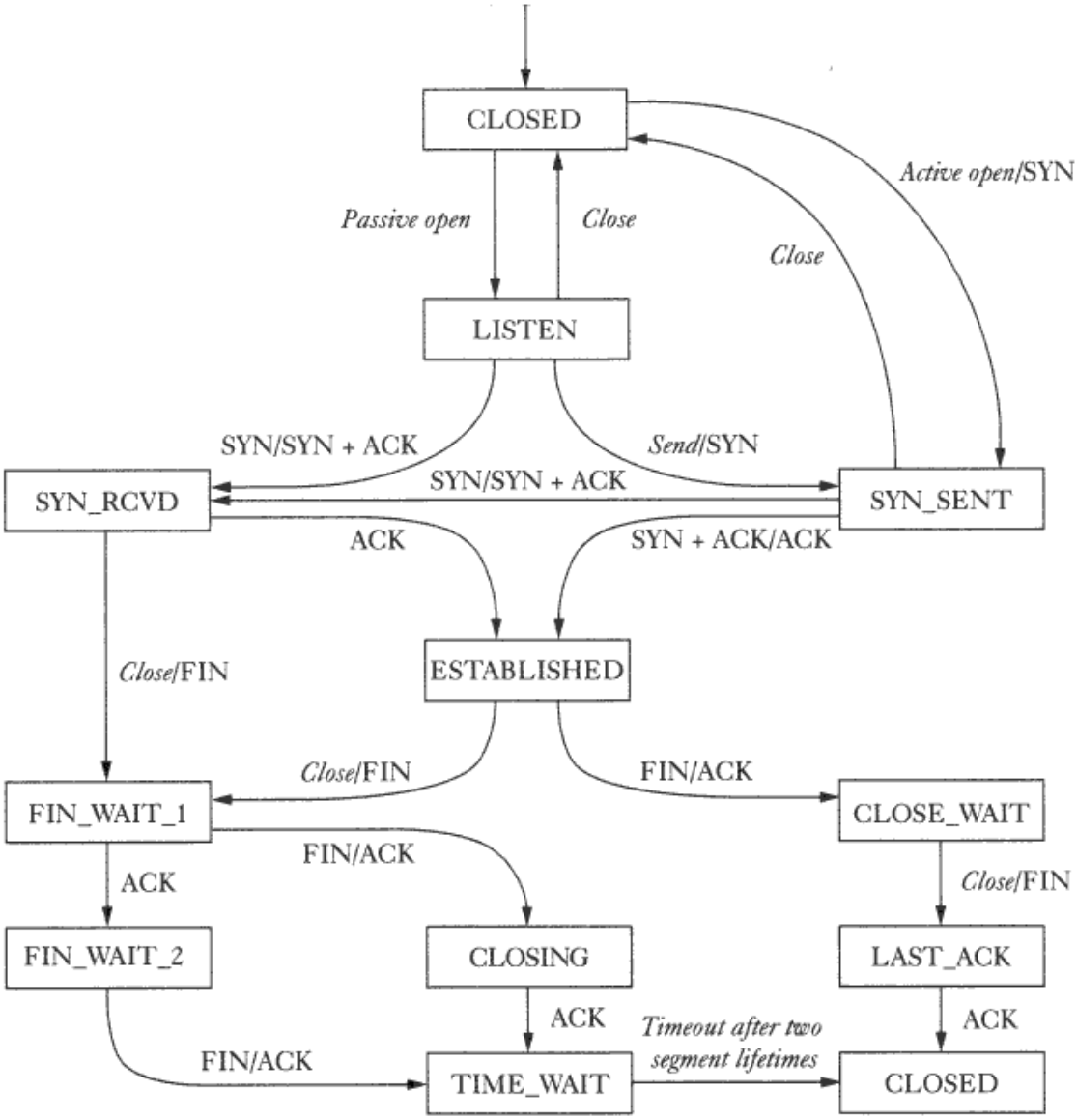
Přenosovou jednotkou v TCP je paket. Kromě datové části může mít paket nastaveno i několik příznaků v záhlaví paketu, které se využívají k řízení spojení (datová část paketu může i chybět, takový paket pak slouží pouze k řízení spojení). Řídícími příznaky jsou např.:

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

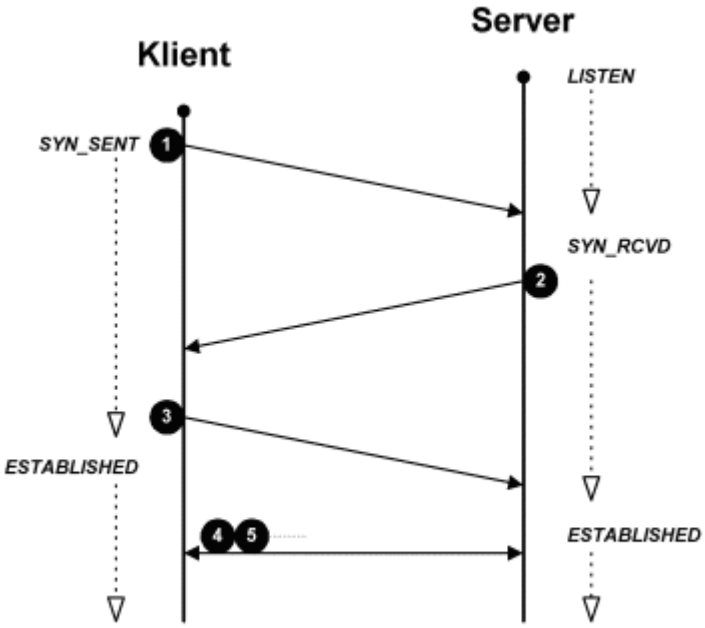
Nyní můžeme ze stavového diagramu odvodit, jak bude vypadat komunikace na úrovni TCP paketů při navazování spojení mezi klientem a serverem. Předpokládáme přitom, že je na serveru otevřen příslušný port, stavový diagram spojení z pohledu serveru je tedy ve stavu LISTEN, spojení na straně klienta neexistuje, 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.

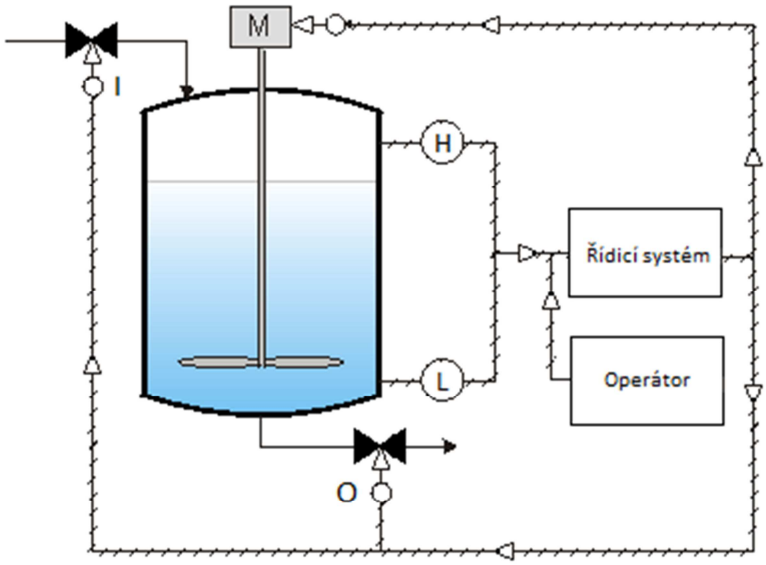
Vysvětlováním mechanizmu ukončování spojení se zabývat nebudeme. Čtenáře, kterým k pochopení nestačí stavový diagram protokolu, odkážeme na učební materiály k předmětům z oblasti počítačových sítí.

1.3.2. 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 ve 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

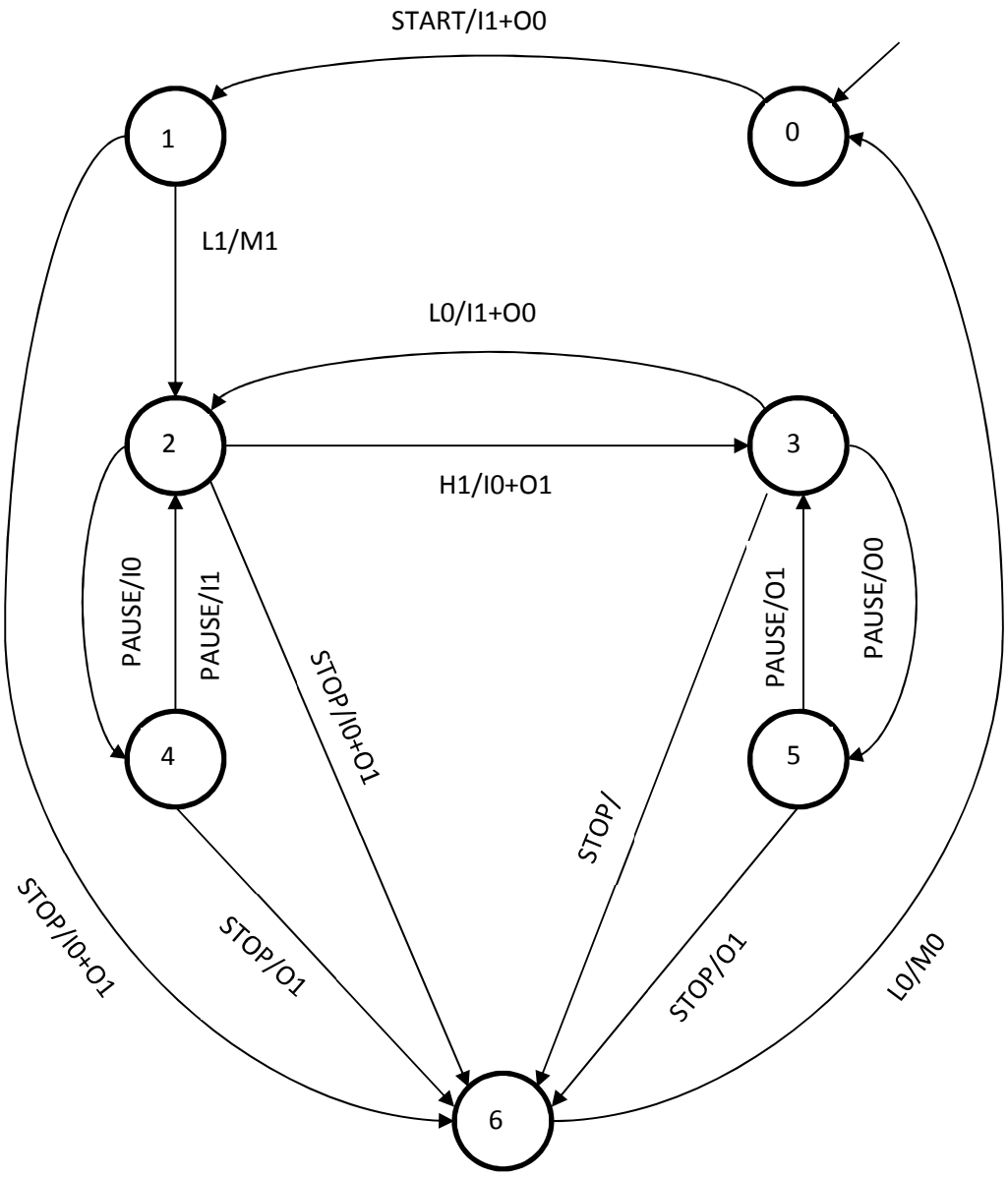
Řídicí systém míchací nádrže lze popsat stavovým diagramem na následující straně. Jednotlivé stavy modelu lze charakterizovat takto:

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 1.4.2. 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.

1.3.3. 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.

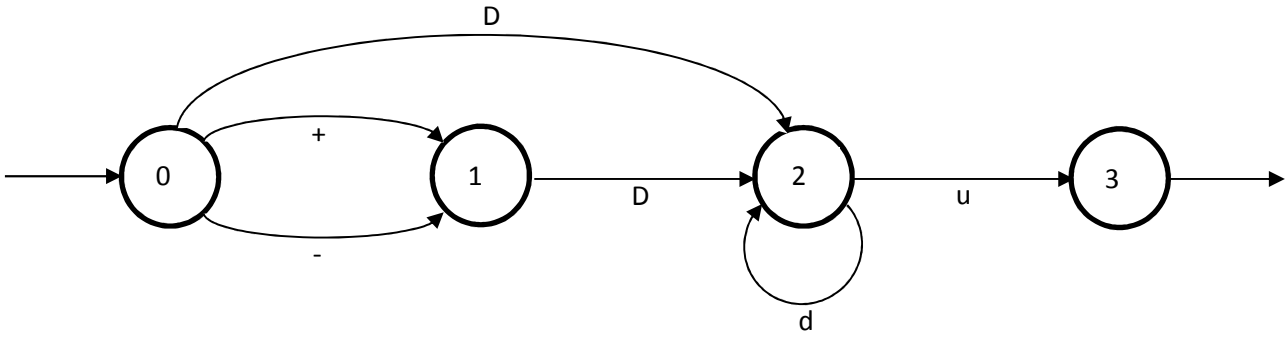
Pro ilustraci – reprezentace některých konstant ve znakové reprezentaci a ve formátu INTEGER s délkou slova 4 byty (obsahy jednotlivých bytů jsou zapsány hexadecimálně):

Konstanta	Reprezentace ve znakovém poli (kód 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* (např. mezera nebo aritmetický operátor).

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 řetězce 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.

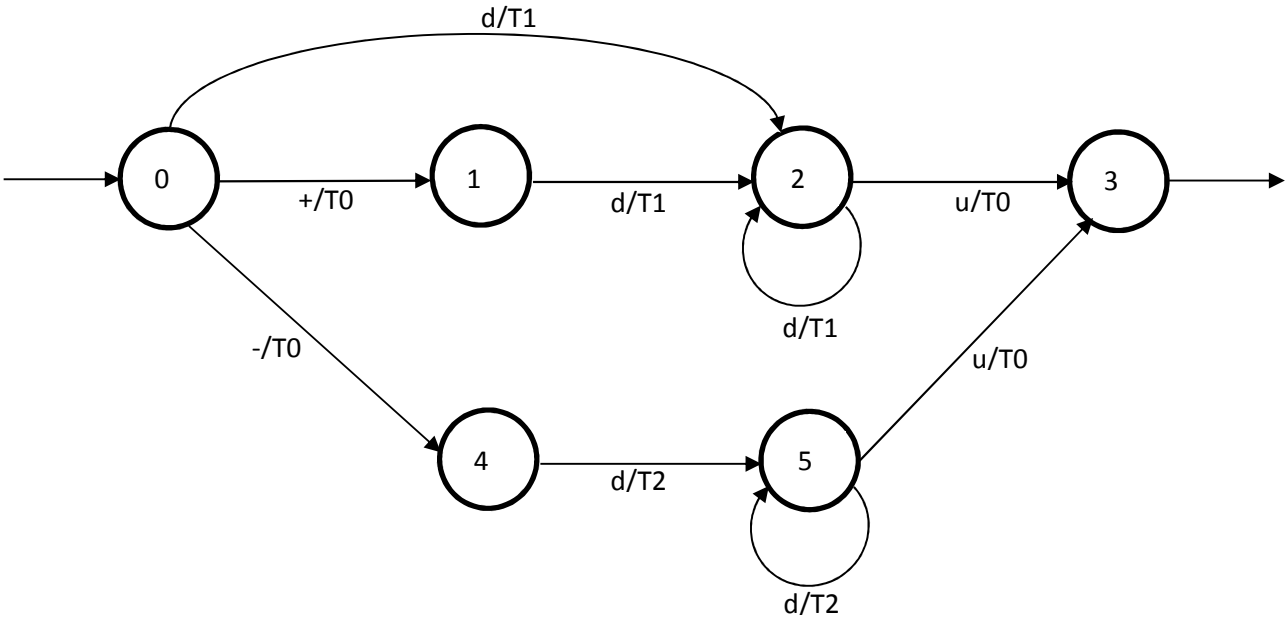
K popisu automatu, který bude provádět vstupní konverzi, můžeme využít přechodový graf rozpoznávacího automatu. Musíme si ale uvědomit, že aby si KA provádějící konverzi mohl

„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 1.4.1 .

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:

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

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

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

Q_p představuje kartézský součin množiny všech makrostavů a oborů hodnot všech pomocných stavových proměnných

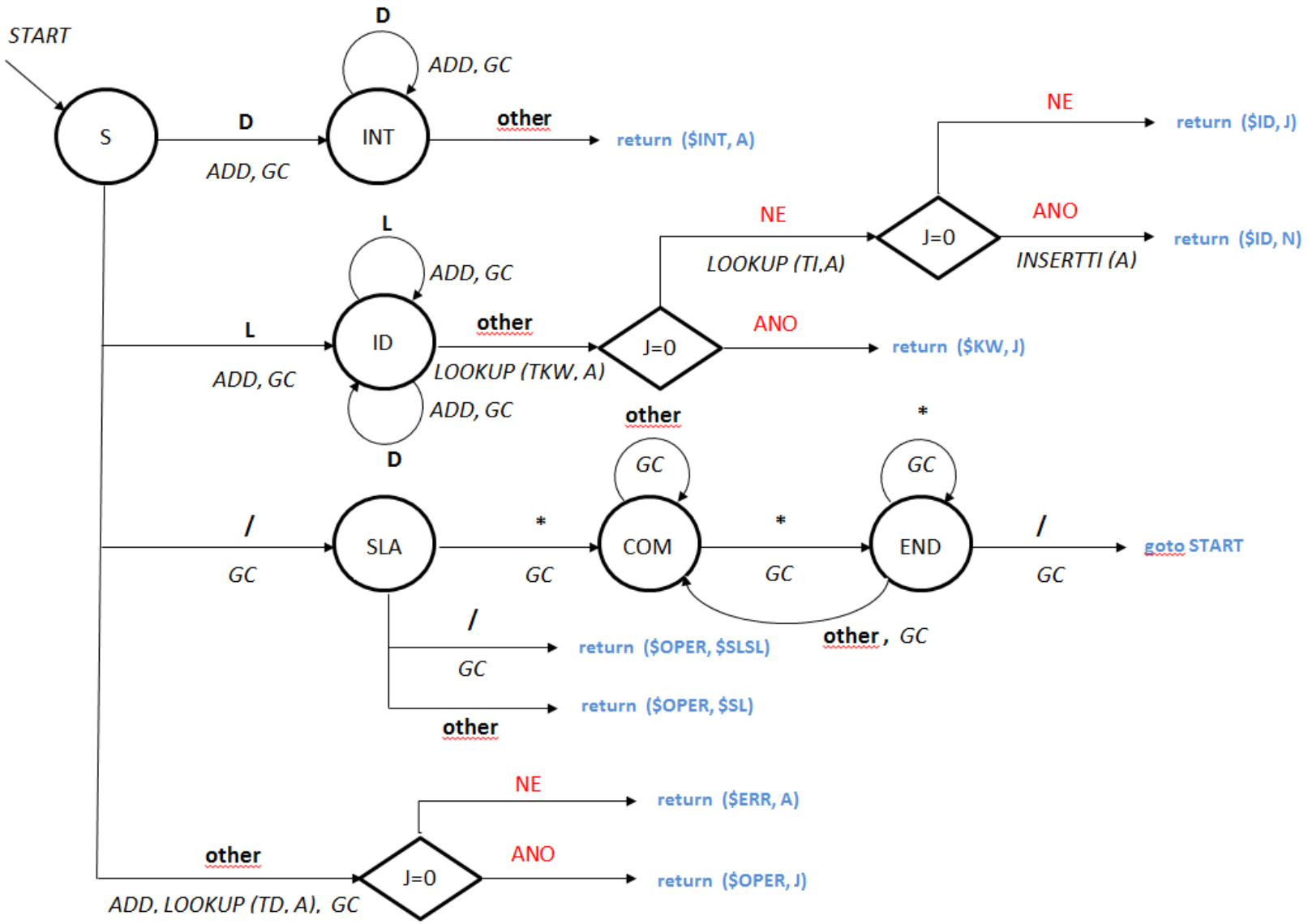
V našem příkladu má Q_m 6 prvků, pomocná proměnná *hodnota* může nabývat 2^{32} různých hodnot (INTEGER na 4 bytech), kartézský součin $Q_p = Q_m \times I_{hodnota}$ bude mít $6*2^{32}$ prvků, ovšem je zřejmé, že každý z makrostavů 0, 1 a 4 představuje jediný stav, takže množina stavů Q našeho automatu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

1.3.4. 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 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).


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

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 *

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, všech-

ny 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 // .

1.4. Základní principy implementace konečných automatů

1.4.1. Principy softwarové implementace

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 1.3.4).

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,

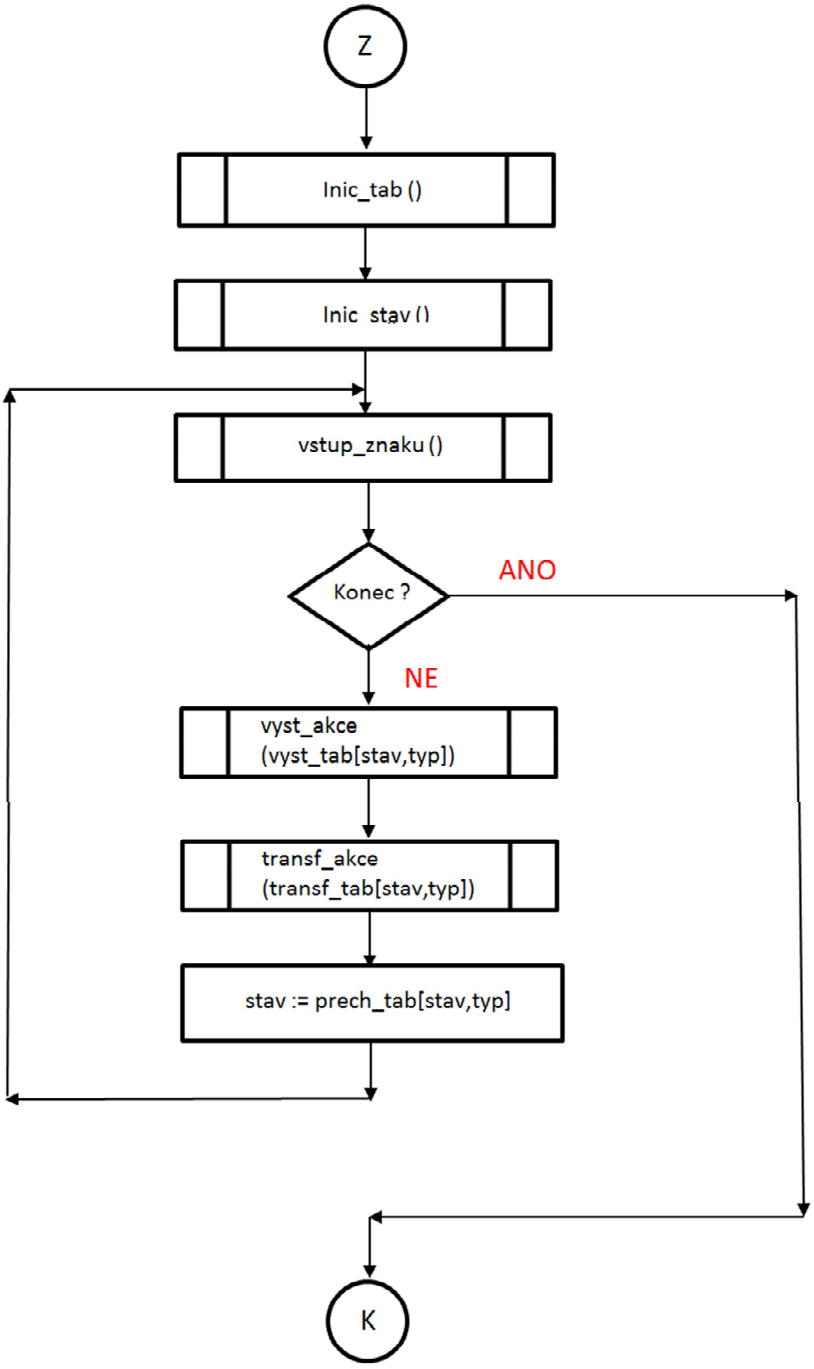
transf_akce ..... parametrem je identifikační kód transformace; provádí transformaci pomocných stavových proměnných podle předaného kódu,

vyst_akce .....parametrem je identifikační kód výstupní akce, kterou procedura provede.

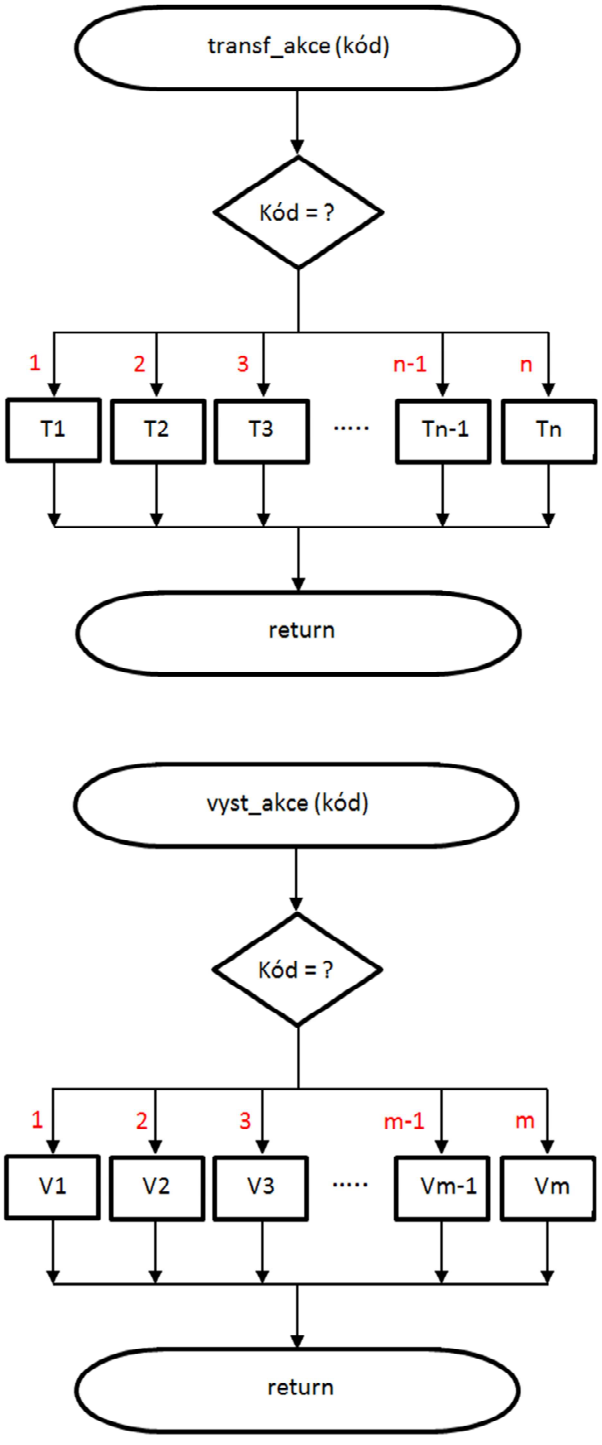
V procedurách nesmí existovat lokální proměnné, které by si uchovávaly svoji hodnotu mezi dvěma voláními procedury !! Jinak řečeno - pokud je potřeba použít v procedurách lokální proměnné, musí být při každém vstupu do procedury opětovně inicializovány. Důvod je zřejmý: roli stavu konečného automatu 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 1.3.3). 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]).

1.4.2. Principy hardwarové realizace

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 libovolně složitého 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

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

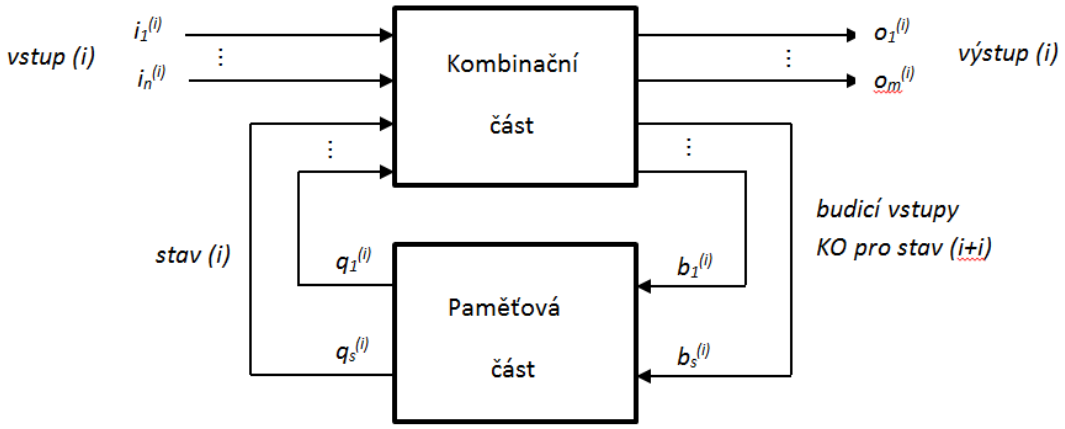
kde *n* je počet vstupních dvouhodnotových proměnných a *m* počet výstupních dvouhodnotových proměnných.

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 klopného 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 budící 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

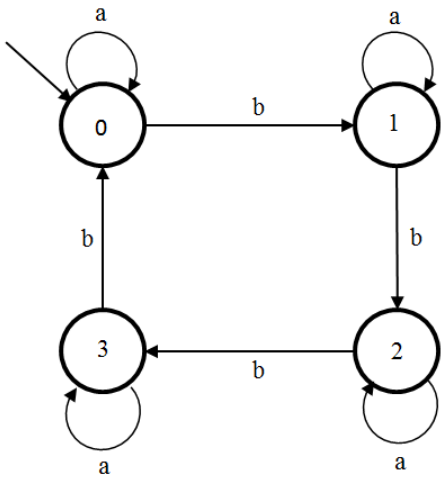
(pravá tabulka je pouze komprimací tabulky levé, neobsahuje žádnou informaci navíc)

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 s není celé číslo, pak samozřejmě nejbližší vyšší celé číslo.

Příklad: S využitím klopných obvodů typu J-K 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 nám ulehčuje generování výstupu (výstup bude přímo výstupem klopného obvodu KO1).

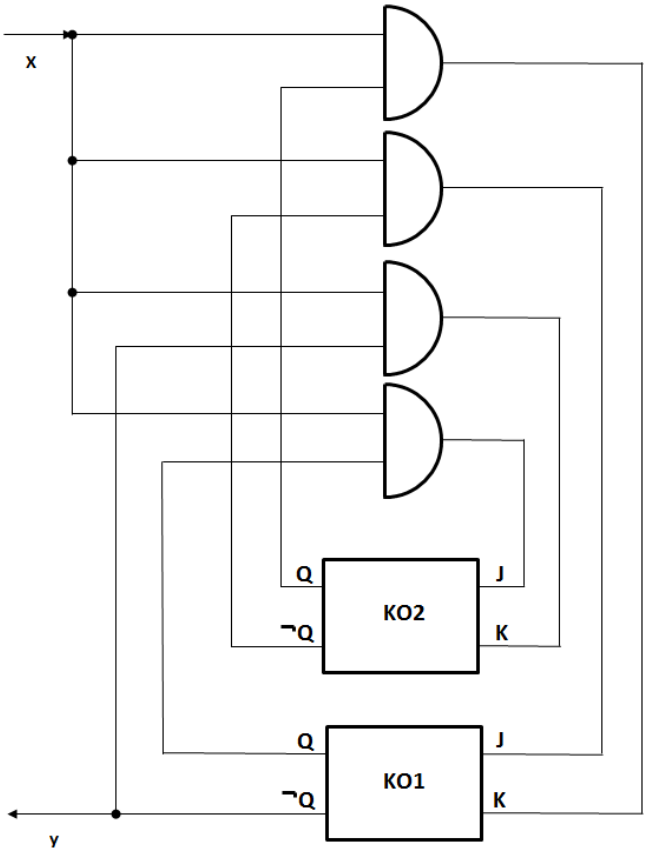
Zakódovaná přechodová tabulka KA + tabulka budicích funkcí klopných obvodů:

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:

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

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).



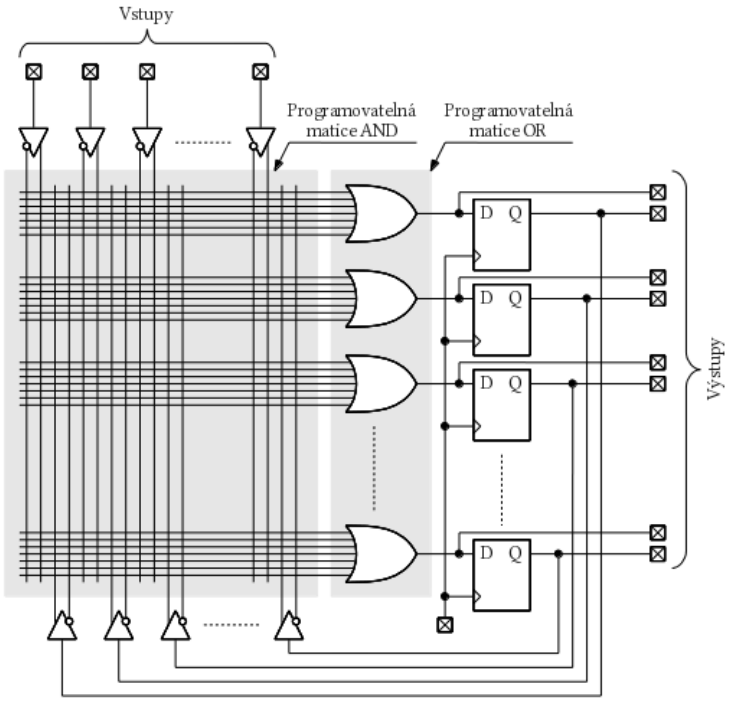
Obrázek znázorňuje schéma zapojení „logické části“ (nejsou nakresleny obvody související s generováním hodinových pulzů a s napájením), u KO nejsou znázorněny vstupy hodinových signálů a vstupy pro asynchronní nulování (tj. pro převedení obvodu do počátečního stavu).

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

Efektivním nástrojem pro realizaci rozsáhlých 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 naznačuje obrázek:



Každá vodorovná čára v programovatelné matici AND představuje vždy jedno součinové hradlo. Průsečík vodorovné a svislé čáry představuje „programovatelný spoj“ (při uživatelském programování obvodu lze ovlivnit, zda konkrétní bod bude/nebude propojen). Na vstupy každého součinového hradla lze tedy připojit libovolnou kombinaci vstupních signálů a zpětných vazeb z výstupů klopných obvodů (nebo 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, tj. pomocí speciálních jazyků pro popis hardware (např. ABEL nebo VHDL) nebo grafický (používají se editory schémat nebo stavových diagramů).