Meno a priezvisko: .....

RT z predmetu AFJ

Na vypracovanie použite vlastný papier. Máte čas **90 minút**. **Všetky úlohy pozorne čítajte!**

1. (10b) Zistite, či uvedené slovo w má/nemá v gramatike G odvodenie - vyberte si ľubovoľný typ parsovacieho prístupu z týchto: CYK, LL(1), SLR(1), LR(1), ktorý riadne a kompletne popíšte. Množiny FIRST a FOLLOW gramatiky máte k dispozícii. Gramatika je LL(1), aj SLR(1), aj LR(1).

   Teda, podľa zvoleného prístupu uveďte:

   - pre CYK - konverziu do Chomského normálneho tvaru gramatiky a množiny $N_{i,j}$;
   - pre LL(1) - PREDICT, RT a výpočet LL(1) parsera;
   - pre SLR(1) - LR(0)-automat, ACTION, GOTO a výpočet SLR(1) parsera;
   - pre LR(1) - LR(1)-automat, ACTION, GOTO a výpočet LR(1) parsera.

   Slovo $w = baa$
   Gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$
   Pravidlá P :

   1. $S \to AB$
   2. $A \to Bb$
   3. $B \to aB$
   4. $B \to \varepsilon$

	S	A	B
FIRST	$\{a, b\}$	$\{a, b\}$	$\{a, \varepsilon\}$
FOLLOW	$\{\varepsilon\}$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$

Riešenie:

Keďže v roku 2024/2025 sme neprebrali LR parsery, očakávalo by sa od Vás, že si v úlohe vyberiete alebo CYK alebo LL(1)-prístup. Najprv spravíme CYK prístup. Pre použitie CYK algoritmu je nutné najprv gramatiku previesť do Chomského normálneho tvaru. Tento postup sa počítal na cvičení a bol na prednáške. To znamená:
Odstrániť ε-pravidlá. Množina $N_\varepsilon = \{B\}$. Gramatika po odstránení ε-pravidiel:

- $S \to AB$
- $S \to A$
- $A \to Bb$
- $A \to b$
- $B \to aB$
- $B \to a$

V ďalšom kroku *odstrániť jednoduché pravidlá.* Množiny $N_S = \{S, A\}, N_A = \{A\}, N_B = \{B\}$. Po odstránení jednoduchých pravidiel:

- $S \to AB$
- $S \to Bb$
- $S \to b$
- $A \to Bb$
- $A \to b$
- $B \to aB$
- $B \to a$

V ďalšom kroku vykonáme *redukciu gramatiky.* Množina $N_T = \{S, A, B\}$. Neodstránime nič. Ďalej množina $V_D = \{S, A, B, b, a\}$, neodstránime nič. Redukovaná gramatika:

- $S \to AB$
- $S \to Bb$
- $S \to b$

- $A \to Bb$
- $A \to b$
- $B \to aB$
- $B \to a$

V ďalšom kroku konverzie na CHNT by sme vykonali skracovanie pravých strán gramatiky - ak by niektorá pravá strana boli dlžky 3 alebo viac. Keďže táto gramatika má všetky pravé strany dĺžky najviac 2, skracovanie nerobíme.

Posledný krok úpravy na CHNT, náhrada terminálov v pravých stranách dĺžky 2 za neterminály. Po úprave:

- $S \to AB$
- $S \to BV_b$
- $S \to b$
- $A \to BV_b$
- $A \to b$
- $B \to V_aB$
- $B \to a$
- $V_a \to a$
- $V_b \to b$

Tým dostávame gramatiku v Chomského normálnom tvare. Následne môžeme pristúpiť k realizácii CYK algoritmu pre reťazec $w = baa$. Postupne teda budeme hľadať množiny N_{ij} neterminálov, z ktorých je možné odvodiť časti slova baa od i-teho po j-ty symbol.

- Časti w dlžky 1:
- $N_{11} = \{S, A, V_b\}$
- $N_{22} = \{B, V_a\}$
- $N_{33} = \{B, V_a\}$
- Časti w dĺžky 2:
- $N_{12} = \{S\}$
- $N_{23} = \{B\}$
- Časť w dĺžky 3, teda celý reťazec:
- $N_{13} = \{S\}$

Na záver konštatujeme, že keďže počiatočný neterminál S je prvkom poslednej množiny N_{13}, to znamená že $S \Rightarrow^* baa$, a teda baa **má v gramatike deriváciu**.

Alternatívne je možné úlohu riešiť pomocou LL(1)-parsera. Najprv by sme gramatiku redukovali - táto je však už redukovaná. Ďalej by sme ako prvé našli množiny $FIRST$ a $FOLLOW$ pre neterminály gramatiky - tie sú už však dané, rovno prejdeme na konštrukciu množín $PREDICT$ pre pravidlá gramatiky:

- $PREDICT(S \to AB) = \{a, b\}$
- $PREDICT(A \to Bb) = \{a, b\}$
- $PREDICT(B \to aB) = \{a\}$
- $PREDICT(B \to \varepsilon) = \{b, \varepsilon\}$

Následne rozkladová tabuľka:

	a	b	ε
S	1	1	
A	2	2	
B	3	4	4

Keďže rozkladová tabuľka obsahuje pre každú kombináciu najviac jedno pravidlo, gramatika je LL(1) gramatikou a môžeme pristúpiť k parsovaniu.

VSTUP:	Zásobník (vrch vľavo)	Akcia
baa	S	E1
baa	AB	E2
baa	BbB	E4
baa	bB	P
aa	B	E3
aa	aB	P
a	B	E3
a	aB	P
ε	B	E4
ε	ε	A

Keďže analyzátor prečítal celý vstup a vyprázdnil zásobník, vstup akceptuje. Reťazec *baa* teda **má v gramatike deriváciu**.

2. (20b) Uvažujme nasledovnú bezkontextovú gramatiku G:

1. <UCET> → <POLOZKA><UCET>
2. <UCET> → ε
3. <POLOZKA> → <TOVAR><JEDN_CENA><VAHA>
4. <TOVAR> → `id`
5. <JEDN_CENA> → `cenakg`
6. <VAHA> → `gramaz`

	FIRST	FOLLOW
<UCET>	{ε, `id`}	{ε}
<POLOZKA>	{`id`}	{ε, `id`}
<TOVAR>	{`id`}	{`cenakg`}
<JEDN_CENA>	{`cenakg`}	{`gramaz`}
<VAHA>	{`gramaz`}	{ε, `id`}

kde neterminály sú ohraničené značkami <,>, počiatočný neterminál je <UCET>, terminálne symboly (lexémy) gramatiky sú `id, cenakg, gramaz`, pričom tieto lexémy reprezentujú nasledovné regulárne výrazy:

(a) `id`: $(a|b|...|z)(a|b|...|z)^*$ (neprázdny textový reťazec malých písmen anglickej abecedy)

(b) `cenakg`: $(0|1|...|9)(0|1|...|9)^*(.)(0|1|...|9)(0|1|...|9)$ (kladné desatinné číslo s 2 ciframi za desatinnou čiarkou)

(c) `gramaz`: $(0|1|...|9)(0|1|...|9)^*(g)$ (kladné celé číslo zreťazené s písmenom g)

Sú dané reťazce:

i. `ceresne 7.00 200g cucoriedky 8.50 100g`

ii. `cernice 14.32 125 jahody 7.50 50g`

Pre reťazce ozn. (i), (ii) vykonajte:

- lexikálnu analýzu - stačí uviesť **výslednú postupnosť lexém**. V prípade, že **nastala lexikálna chyba**, uveďte túto skutočnosť a označte časť reťazca, ktorú sa nepodarilo tokenizovať. Pre jednoduchosť uvažujte, že medzera oddeľuje jednotlivé lexémy. **Dané reťazce POZORNE čítajte!**
- syntaktickú analýzu - zostrojte syntaktický analyzátor - **iný, než ste spravili v úlohe č. 1** - typu LL(1), alebo SLR(1), alebo LR(1) pre uvedenú gramatiku G (gramatika je aj LL(1), aj SLR(1), aj LR(1)). Popíšte **všetky podstatné časti analyzátora** (pozri úloha č. 1) a nad platnou postupnosťou lexém vykonajte syntaktickú analýzu. Ak má reťazec deriváciu, nakreslite jeho **derivačný strom**.

Riešenie: Ako prvé vykonáme lexikálnu analýzu. Podľa prednášky o lexikálnej analýze to znamená, že potrebujeme rozdeliť príslušný reťazec na postupnosť lexém (tokenov), teda rozpoznať jednotlivé lexémy - v tomto prípade `id, cenakg, gramaz` v danom reťazci podľa príslušných regulárnych výrazov.

Výsledkom lexikálnej analýzy prvého reťazca `ceresne 7.00 200g cucoriedky 8.50 100g` by bola postupnosť lexém: `id cenakg gramaz id cenakg gramaz`, pretože časti reťazca `ceresne`, resp. `cucoriedky` patria do jazyka popísaného regulárnym výrazom pre lexému `id`, reťazce `7.00` a `8.50` patria do lexémy `cenakg` (teda postupnosť cifier dĺžky aspoň 1 za ktorou je bodka za ktorou sú presne 2 cifry) a reťazce `200g` a `100g` patria do lexémy `gramaz` (postupnosť cifier dĺžky aspoň 1 za ktorou je písmeno g).

Lexikálna analýza druhého reťazca by rozpoznala reťazec `cernice` ako lexému `id`, reťazec `14.32` ako lexému `cenakg` a pri spracovaní reťazca `125` by došlo k **lexikálnej chybe**, pretože časť `125` **nepatrí** do žiadnej lexémy. Nie je to `id`, ani `cenakg` a ani `gramaz`, pretože reťazce rozpoznané ako lexéma `gramaz` **musia končiť písmenom g.** Pre reťazec ii. je teda potrebné uviesť, že obsahuje lexikálnu chybu, pretože jeho časť `125` sa nedá rozpoznať ako žiadna z uvedených lexém.

Ako druhé vykonáme syntaktickú analýzu nad **platnou postupnosťou lexém**. Ako sme ukázali vyššie, druhý reťazec **nie je platná postupnosť lexém**, preto má zmysel robiť syntaktickú analýzu len pre **lexikálnu reprezentáciu prvého reťazca**. Pre reťazec `ceresne 7.00 200g cucoriedky 8.50 100g` teda nerobíme syntaktickú analýzu priamo na tomto reťatci, ale na korešpondujúcej postupnosti lexém, teda `id cenakg gramaz id cenakg gramaz`.

Keďže tento rok sme LR-parsery nepreberali, zostrojíme LL(1) parser (CYK algoritmus nie je vhodný pre účely syntaktickej analýzy, keďže nevie vrátiť postupnosť pravidiel, ktoré dávajú deriváciu daného reťazca). Množiny PREDICT:

- $PREDICT($<UCET> → <POLOZKA><UCET>$) = \{$`id`$\}$
- $PREDICT($<UCET> → $\varepsilon) = \{\varepsilon\}$

- $PREDICT(<\text{POLOZKA}> \rightarrow <\text{TOVAR}><\text{JEDN\_CENA}><\text{VAHA}>) = \{\texttt{id}\}$
- $PREDICT(<\text{TOVAR}> \rightarrow \texttt{id }) = \{\texttt{id}\}$
- $PREDICT(<\text{JEDN\_CENA}> \rightarrow \texttt{cenakg}) = \{\texttt{cenakg}\}$
- $PREDICT(<\text{VAHA}> \rightarrow \texttt{gramaz}) = \{\texttt{gramaz}\}$

Rozkladová tabuľka:

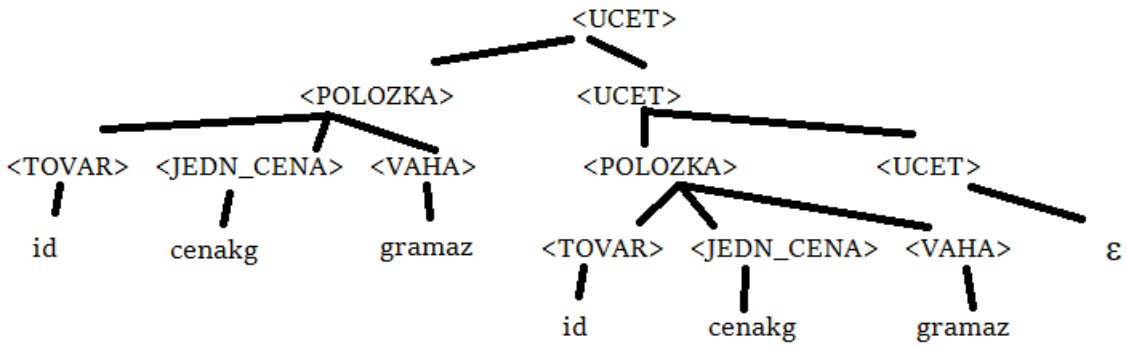
	`id`	`cenakg`	`gramaz`	ε
<UCET>	1			2
<POLOZKA>	3			
<TOVAR>	4			
<JEDN_CENA>		5		
<VAHA>			6	

Následne vykonáme syntaktickú analýzu reťazca `id cenakg gramaz id cenakg gramaz`, teda výpočet LL(1) parsera:

VSTUP:	Zásobník (vrch vľavo)	Akcia
`id cenakg gramaz id cenakg gramaz`	<UCET>	E1
`id cenakg gramaz id cenakg gramaz`	<POLOZKA><UCET>	E3
`id cenakg gramaz id cenakg gramaz`	<TOVAR><JEDN_CENA><VAHA><UCET>	E4
`id cenakg gramaz id cenakg gramaz`	`id`<JEDN_CENA><VAHA><UCET>	P
`cenakg gramaz id cenakg gramaz`	<JEDN_CENA><VAHA><UCET>	E5
`cenakg gramaz id cenakg gramaz`	`cenakg`<VAHA><UCET>	P
`gramaz id cenakg gramaz`	<VAHA><UCET>	E6
`gramaz id cenakg gramaz`	`gramaz`<UCET>	P
`id cenakg gramaz`	<UCET>	E1
`id cenakg gramaz`	<POLOZKA><UCET>	E3
`id cenakg gramaz`	<TOVAR><JEDN_CENA><VAHA><UCET>	E4
`id cenakg gramaz`	`id`<JEDN_CENA><VAHA><UCET>	P
`cenakg gramaz`	<JEDN_CENA><VAHA><UCET>	E5
`cenakg gramaz`	`cenakg`<VAHA><UCET>	P
`gramaz`	<VAHA><UCET>	E6
`gramaz`	`gramaz`<UCET>	P
ε	<UCET>	E2
ε	ε	A

Keďže vstup bol celý prečítaný a syntaktický analyzátor skončil s prázdnym zásobníkom - vstup `id cenakg gramaz id cenakg gramaz` **má deriváciu v gramatike**. LL(1) parser zistil, že **ľavá derivácia** vznikne postupnou aplikáciou pravidiel 1,3,4,5,6,1,3,4,5,6,2.

V zadaní úlohy je ďalej napísané, že ak má postupnosť lexém deriváciu - a my sme zistili, že má - je potrebné nakresliť jej derivačný strom. Takže na základe pravidiel, ktoré vrátil LL(1), kreslíme:



3. (5b) Uveďte príklad redukovanej bezkontextovej gramatiky s aspoň 2 neterminálmi a aspoň 2 terminálmi, ktorá **nie je LR(0)** gramatikou.

Riešenie: Toto riešiť nebudem, lebo LR(0) gramatiky sme tento rok nepreberali.

4. (5b) Uveďte príklad redukovanej bezkontextovej gramatiky s aspoň 2 neterminálmi a aspoň 2 terminálmi, ktorá **nie je LL(1)** gramatikou.

Riešenie: Ak máme nájsť gramatiku, ktorá nie je LL(1) gramatikou, potrebujeme zostrojiť takú gramatiku, ktorá bude mať s istotou konflikt v rozkladovej tabuľke. Potrebujeme teda zostrojiť gramatiku, ktorá bude mať neterminál, ktorý bude obsahovat 2 pravidlá, ktoré budú mať prienik v ich množinách PREDICT. Keďže základom množiny PREDICT je počítanie množiny FIRST pravej strany, zostrojíme 2 pravidlá, ktoré budú mať určite kolíziu v množinách FIRST - obe pravidlá budú mať **spoločnú predponu!**. Napríklad gramatiku s 2 neterminálmi S, A, 2 terminálmi a, b a pravidlami - rovno uvediem aj ich množiny PREDICT:

- $S \to A, PREDICT(S \to A) = \{a\}$,
- $A \to aA, PREDICT(A \to aA) = \{a\}$,
- $A \to ab, PREDICT(A \to ab) = \{a\}$.

Pravidlá 2 a 3 majú na pravej strane spoločnú predponu - terminál a. Teda 2. a 3. pravidlo budú mať spoločný prvok v množinách PREDICT a v rozkladovej tabuľke $RT[A, a]$ by boli uvedené aj pravidlo 2, aj pravidlo 3, teda vznikne konflikt a gramatika nie je LL(1) gramatikou.

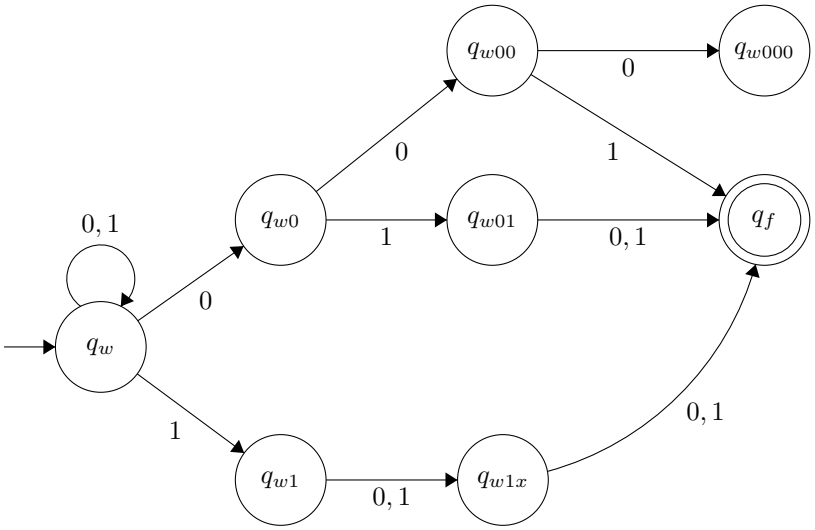
5. (10b) Nájdite konečný automat (nechám na Vás či DKA alebo NKA), ktorý akceptuje jazyk $L = \{w \in \{0,1\}^* \mid$ aspoň 1 z posledných 3 symbolov slova w je jednotka$\}$.

Riešenie: Do jazyka patria práve tie reťazce, ktoré majú na jednom z posledných 3 miest aspoň jednu jednotku. Teda inými slovami:

- Reťazec musí byť dĺžky aspoň 3
- Reťazec nesmie končiť 000, teda bude končiť 001, 010, 011, 100, 101, 110, 111.

Je teda vhodné navrhnúť NKA/DKA tak, aby sledoval, aké sú posledné 3 symboly aktuálne prečítanej časti vstupu a ak sú posledné 3 symboly čokoľvek iné, než 000, tak bude aktuálne signalizovaná akceptácia vstupu. Zároveň treba zaistiť, aby automat neakceptoval reťazec kratší ako 3.

Prvé riešenie, ktoré ukážem, je pomocou NKA:



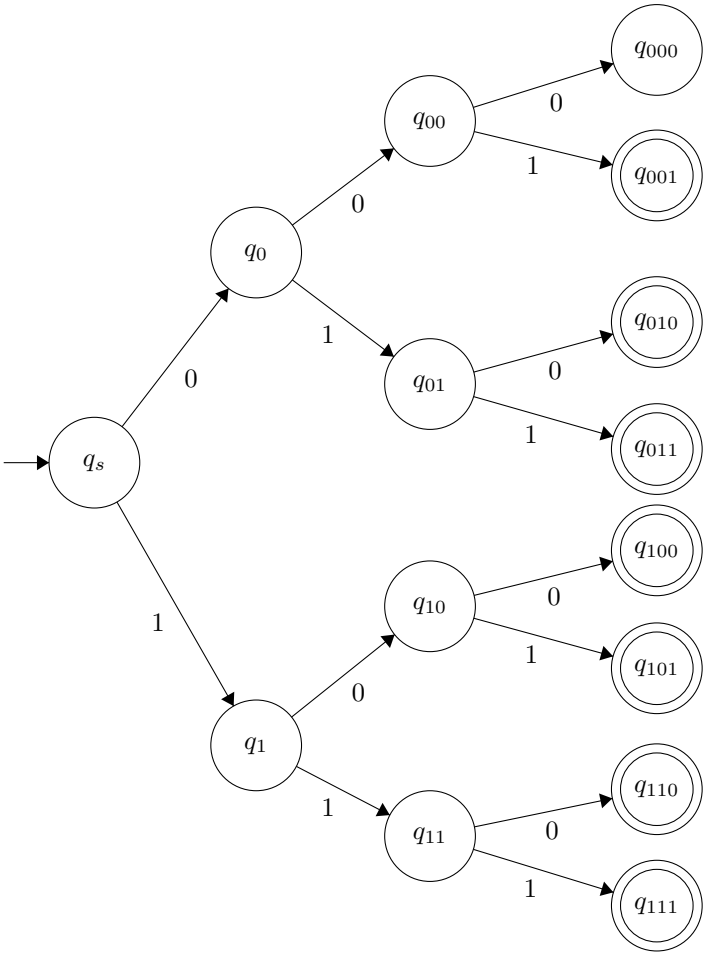
Vysvetlenie logiky NKA

- Uvedený jazyk je možné popísať regulárnym výrazom: $(0 \mid 1)^*(001 \mid 010 \mid 011 \mid 100 \mid 101 \mid 110 \mid 111)$ teda že doň patria všetky reťazce končiace niektorou trojicou z 001,010,011,100,101,110,111
- NKA pracuje tak, že tú časť vstupu ktorá **netvorí posledné 3 symboly**, spracuje v stave q_w - označím ju premennou w.
- Následne sa NKA vyberie tou cestou, ktorá zodpovedá posledným 3 symbolom. Ak je prvý symbol zo záverečnej trojice 1, teda vstup končí 100,101,110,111, automat sa prepne do stavu q_{w1}. Následne je **irelevantné**, aké budú ďalšie 2 symboly, teda predposledný a posledný symbol, reťazec určite **patrí do jazyka**, preto sa dostane automat po ich spracovaní do q_f.
- Ak je prvý symbol z poslednej trojice 0, automat sa prepne do q_{w0}. Následne sa rozlíši, či je ďalší symbol z poslednej trojice 0 alebo 1, podľa toho sa prepne do q_{w00} alebo w_{w01}.
- Ak je automat v stave q_{w01}, znamená to, že predposledný symbol zo záverečnej trojice bol 1. Teda reťazec končí určite 010 alebo 011, a teda patrí do jazyka L. Preto zo stavu q_{w01} vedie prechod na 0 alebo 1 do akceptačného stavu q_f.
- Ak je automat v stave q_{w00}, znamená to, že prvý a druhý symbol zo záverečnej trojice boli nuly. Je teda na poslednom symbole vstupu, či celý reťazec automat bude alebo nebude akceptovať. Ak je posledný symbol vstupu 0, znamená to, že vstup končí trojicou 000. Automat sa prepne do stavu q_{w000}, ktorý **nie je akceptačný**.
- Ak je automat v stave q_{w00} a posledný symbol na vstupe je 1, znamená to, že reťazec končí trojicou 001, reťazec patrí do jazyka L a automat sa prepne do stavu q_f.
- Stav q_f je teda akceptačným stavom, ktorý signalizuje len to, že vstup určite končil príponou 001, 010, 011, 100, 101, 110, 111, a teda že sa má akceptovať.

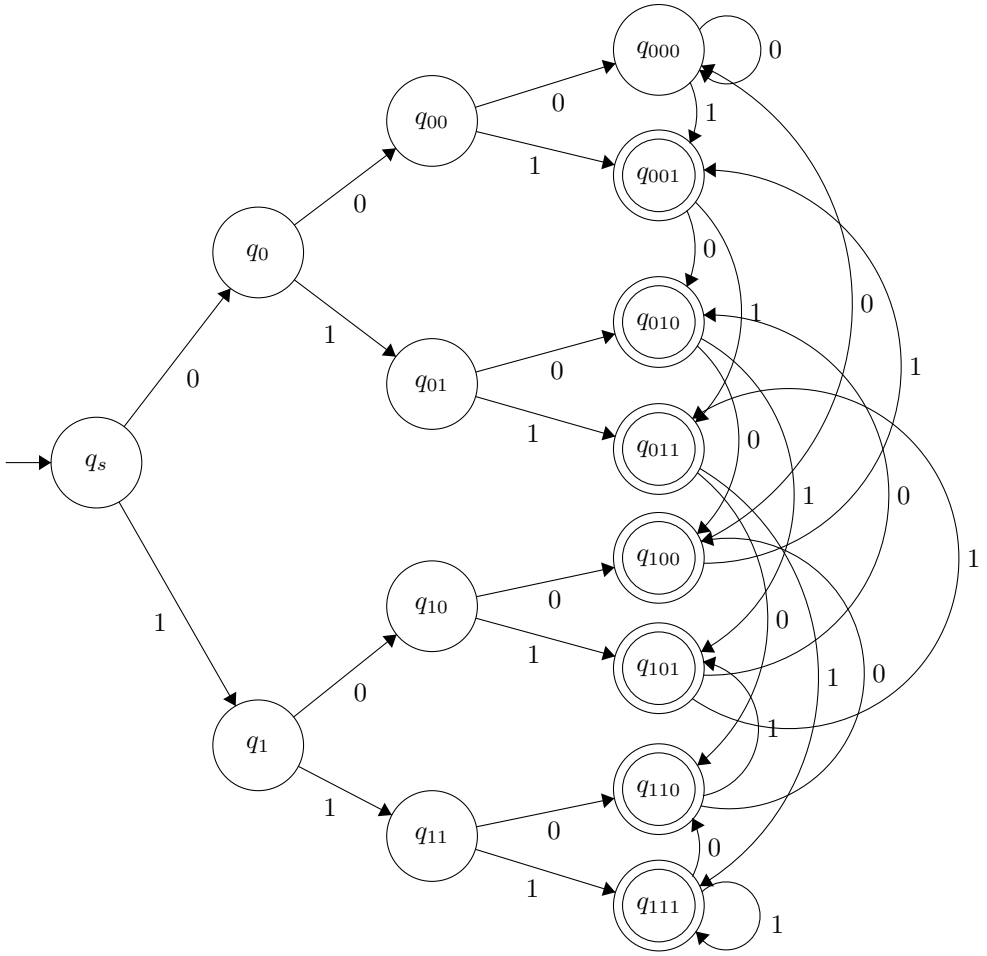
Druhé riešenie, ktoré ukážem, bude pomocou DKA (nie je to najjednoduchšie riešenie z hľadiska počtu stavov, ale ide o riešenie, ktoré demonštruje, ako deterministicky rozmýšľať pri konštrukcii DKA).

V podstate spravíme automat veľmi podobný NKA, teda tiež budeme sledovať posledné 3 vstupné symboly. Rozdiel je v tom, že v NKA sa viete "spoľahnúť" na nedeterministické správanie, pretože vám stačí, aby existoval akceptačný výpočet - v uvedenom NKA to znamená, že ak sa práve číta tretí symbol od konca, tak to je ten symbol, ktorým by sa mal NKA prepnúť zo stavu q_w do alebo q_{w0} alebo q_{w1}.

V DKA je nutné všetky prechody mať deterministické! Všimnite si, ako je navrhnutý nasledovný DKA - **toto ešte nie je finálne riešenie!!!**. Stavy q_0, q_1 predstavujú **prvý prečítaný symbol**. Stavy $q_{00}, q_{01}, q_{10}, q_{11}$ predstavujú **prvé dva prečítané symboly**. Stavy $q_{000}, .., q_{111}$ rozlišujú 8 rôznych situácií podľa toho, aké boli prvé 3 prečítané symboly! Všimnite si, ktoré stavy sú akceptačné!



Výsledné riešenie z tohto DKA získame tak, že sa teraz na stavy $q_{000}, ..., q_{111}$ budeme dívať aj ako na stavy, ktoré hovoria, aké boli **doposiaľ posledné 3 prečítané symboly**. Ak sa napríklad automat nachádza v stave q_{000}, teda na vstupe bol taký reťazec, že jeho posledné 3 doposiaľ prečítané symboly boli 000 a ďalší vstup bude 1, automat sa prepne do q_{001}. Alebo ak je automat v stave q_{010}, teda doposiaľ bola prečítaná taká časť vstupu, ktorá končila 010 a ďalší symbol na vstupe by bol 0, automat sa prepne do q_{100}. Ak by v stave q_{010} bol ďalší vstupný symbol 1, znamená to, že nová doposiaľ prečítaná posledná trojica vstupu bude 101 a automat sa prepne do q_{101}. V podobnom duchu realizujeme prechody z každého stavu $q_{000}, ..., q_{111}$, čím dostaneme výsledné riešenie:



Samozrejme pri NKA a DKA platí, že **akékoľvek vaše riešenie**, ktoré akceptuje ten istý jazyk, je z hľadiska skúšky **rovnako dobré**.

6. (10b) Navrhnite bezkontextovú gramatiku popisujúcu blok deklarácie premenných. Terminály (lexémy) máte uvedené, je nutné určiť neterminály, počiatočný neterminál a pravidlá. Chceme popísať syntax bloku deklarácie premenných v nejakom programovacom jazyku podľa nasledovných pokynov:

   (a) Blok deklarácie premenných pozostáva z deklarácie jednej premennej za ktorou nasleduje blok deklarácie premenných.

   (b) Blok deklarácie premenných môže byť prázdny.

   (c) Deklarácia jednej premennej môže byť realizovaná 2 spôsobmi:

   - Deklarácia premennej obsahuje typ premennej, za ktorým nasleduje identifikátor premennej, za ktorým nasleduje `;` (bodkočiarka). (t.j. ide o deklaráciu premennej s typom)
   - Deklarácia premennej obsahuje typ premennej, za ktorým nasleduje identifikátor premennej, za ktorým nasleduje znak priradenia `=` (rovná sa), za ktorým nasleduje konštanta, za ktorou nasleduje `;` (bodkočiarka). (t.j. ide o deklaráciu premennej s typom a s inicializáciou).

   (d) Typ premennej môže byť:

   - lexéma `Char` - táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz $(Char)$
   - lexéma `Int` - táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz (Int)
   - lexéma `Double` - táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz $(Double)$

   (e) Identifikátor premennej je reprezentovaný lexémou `id`. Táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz $(a|b|...|z)(a|b|...|z)^*$

   (f) Konštanta môže byť:

   - lexéma `konst_char` - znaková konštanta, táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz $(')(a|b|...|z)(')$
   - lexéma `konst_int` - celočíselná konštanta, táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz $(0|1|...|9)(0|1|...|9)^*$
   - lexéma `konst_double` - desatinná konštanta, táto lexéma by následne v lexikálnom analyzátore rozpoznávala reťazce spĺňajúce regulárny výraz
     $(0|1|...|9)(0|1|...|9)^*(.)(0|1|...|9)(0|1|...|9)^*$

Príklad: Ukážka reťazca, z ktorého odvodená postupnosť lexém po tokenizácii (lexikálnej analýze) patrí do uvedeného jazyka (pri tokenizácii ignorujeme whitespace znaky):

```
Int i;
Char c = 'a';
Double d = 10.5;
```

Riešenie:

Táto úloha má za cieľ zistiť, či máte aspoň tušenia, o čom do ryže vôbec bol tento predmet a načo slúžia o.i. gramatiky. V tejto úlohe pomocou bezkontextovej gramatiky formálne popíšeme štruktúru deklarácie bloku premenných podľa pokynov - body (a) - (f) korešpondujú s bodmi zo zadania úlohy. Pravidlá gramatiky by boli (nová strana):

(a) <BLOK_DEKLARACII> → <DEKLARACIA> <BLOK_DEKLARACII>

(b) <BLOK_DEKLARACII> → ε

(c) <DEKLARACIA> → <TYP> <IDENTIFIKATOR> ;
<DEKLARACIA> → <TYP> <IDENTIFIKATOR> = <KONSTANTA> ;

(d) <TYP> → `Char`
<TYP> → `Int`
<TYP> → `Double`

(e) <IDENTIFIKATOR> → `id`

(f) <KONSTANTA> → `konst_char`
<KONSTANTA> → `konst_int`
<KONSTANTA> → `konst_double`

Počiatočný neterminál S = <BLOK_DEKLARACII>. Neterminály N = { <BLOK_DEKLARACII>, <DEKLARACIA>, <TYP>, <IDENTIFIKATOR>, <KONSTANTA> }.

7. (Bonus 5b) Ku gramatike z predchádzajúceho bodu navrhnite sémantické podprogramy, ktoré sa realizujú v prípade redukcie podľa pravidiel gramatiky. Úlohou sémantických podprogramov je **výpočet miesta, ktoré budú premenné zaberať v pamäti**. Uvažujte, že premenná typu `Char` zaberá 1 bajt, premenná typu `Int` zaberá 4 bajty a premenná typu `Double` zaberá 8 bajtov. Sémantické podprogramy zabezpečia, že koreň derivačného stromu bude mať priradený sémantický atribút, ktorý popisuje celkovú pamäťovú náročnosť bloku deklarácie premenných.

Príklad: Pre reťazec z minulého príkladu by v derivačnom strome bol v koreni uložený sémantický atribút s hodnotou 13, pretože celý blok deklarácií obsahuje deklaráciu premennej typu `Int` (t.j. o veľkosti 4 bajtov), `Char` (t.j. o veľkosti 1 bajt) a `Double` (t.j. o veľkosti 8 bajtov).

Riešenie: Toto riešiť nebudem, sémantické podprogramy sme tento rok nepreberali.