Automaty a gramatiky (BI-AAG)

10. Programová realizace NKA a DKA, obvodová realizace

Jan Holub

Katedra teoretické informatiky

Fakulta informačních technologií

ČVUT v Praze

Programová realizace

2 základní přístupy:

- Table driven – přechodová funkce δ uložena v proměnné (např. v dvourozměrném poli, zřetězeném seznamu, . . .), stav realizovaný proměnnou
- Hardcoded – přechodová funkce δ realizována příkazy programu, stav reprezentovaný místem v programu

+ speciální případy

DKA

Příklad:

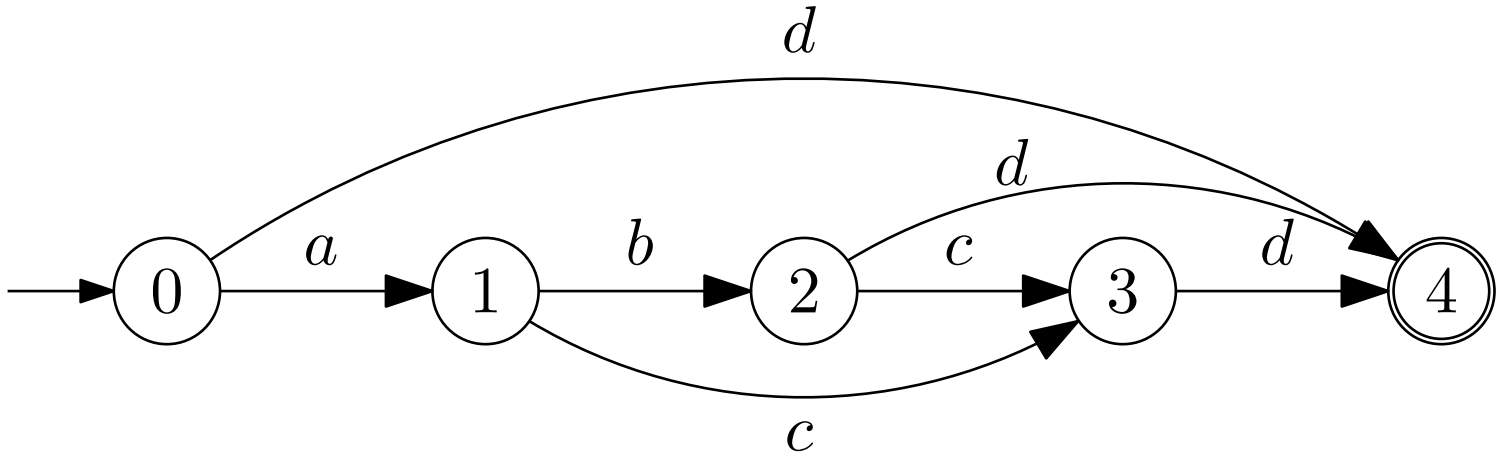


Table Driven

transition_table:

	a	b	c	d
0	1	-	-	4
1	-	2	3	-
2	-	-	3	4
3	-	-	-	4
4	-	-	-	-

```
int DFA_TD(){
  int state=0, symbol;

  while( (symbol = getchar()) != EOF ) {
    state = transition_table[state][symbol];
  }
  return is_final[state];
}
```

Hard Coded

```
int DFA_HC(){
  int symbol;

  state0: if ((symbol = getchar()) == EOF) return 0;
          switch (symbol){
              case 'a': goto state1;
              case 'd': goto state4;
              default: return(-1);
         };
  state1: if ((symbol = getchar()) == EOF) return 0;
          switch (symbol){
              case 'b': goto state2;
              case 'c': goto state3;
              default: return(-1);
         };
  state2: if ((symbol = getchar()) == EOF) return 0;
          switch (symbol){
              case 'c': goto state3;
              case 'd': goto state4;
              default: return(-1);
         };
  state3: if ((symbol = getchar()) == EOF) return 0;
          switch (symbol){
              case 'd': goto state4;
              default: return(-1);
         };
  state4: if ((symbol = getchar()) == EOF) return 1;
          return(-1);
}
```

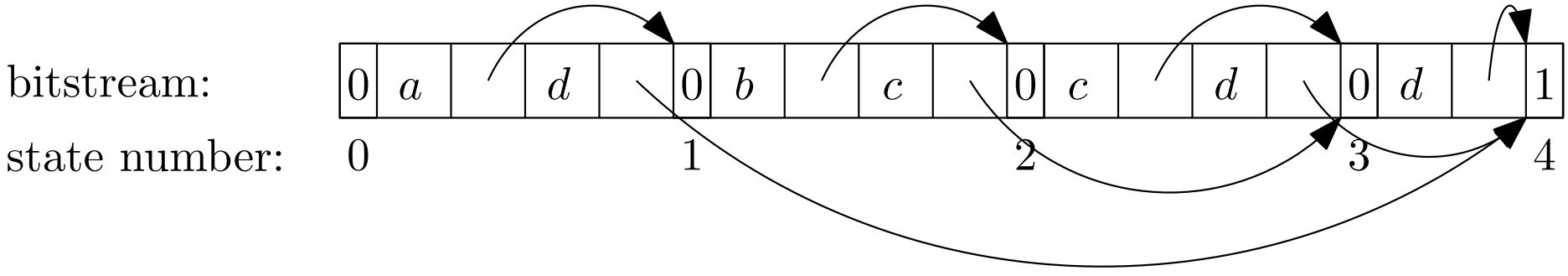
Porovnání dvou základních přístupů

	Table Driven	Hardcoded
časová složitost	$\mathcal{O}(n)$	$\mathcal{O}(n)$
velikost programu	$\mathcal{O}(1)$	$\mathcal{O}(\|Q\| * \|\Sigma\|)$
paměť pro proměnné	$\mathcal{O}(\|Q\| * \|\Sigma\|)$	1

Bitstream Implementation

Speciální případ:

- Acyklický konečný automat, stavy seřazené zleva doprava podle přechodů.
- Nevracíme se zpět, vhodné pro využití cache pamětí.



Hardwarová implementace

Co potřebujeme:

- zakódovat vstupní abecedu,
- zakódovat stavy automatu, pamatovat si současný stav,
- realizovat přechodovou funkci automatu pomocí zakódovaných stavů a vstupních symbolů,
- rozpoznat koncové stavy.

Kódování vstupní abecedy

Vstupní abecedu zakódujeme v binárním kódu

- Pro text můžeme použít 8 bitů (rozšířená ASCII).
- menší abeceda $\Rightarrow$ menší počet bitů (méně součástek, nutná konverze)

Kódování vstupní abecedy

Vstupní abecedu zakódujeme v binárním kódu

- Pro text můžeme použít 8 bitů (rozšířená ASCII).
- menší abeceda $\Rightarrow$ menší počet bitů (méně součástek, nutná konverze)

Příklad – $\Sigma = \{a, b, c, d\}$

Symbol	Kód
a	00
b	01
c	10
d	11

Pro reprezentaci v binárním kódu nám stačí $\lceil \log_2(|\Sigma|) \rceil$ bitů.

Reprezentace stavů

Stavy nejjednodušeji kódujeme binárním nebo Grayovým kódem.

- Aktuální stav je uložen v datovém registru.
- Přechodová funkce každé kombinaci kódu starého stavu a vstupního symbolu přiřazuje kód nového stavu.
- Inicializace – vynulováním datového registru (pokud pro počáteční stav zvolíme kód 0).

Reprezentace stavů

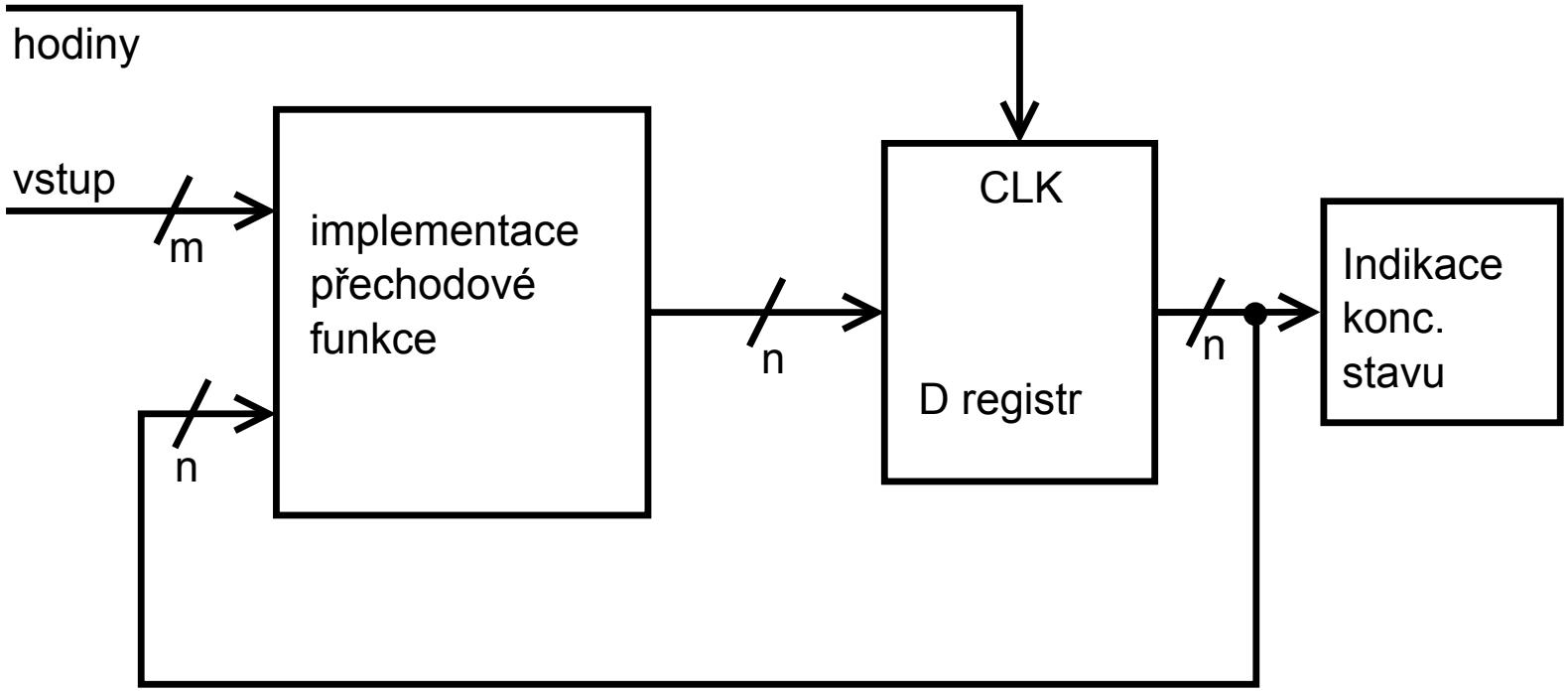
Alternativně lze použít kód 1 z N.

- Každý stav je představován jedním 1bitovým klopným obvodem. Pokud je nastaven na 1, stav je aktivní.
- Každý přechod je realizován funkcí zdrojového stavu a kódu vstupního symbolu.
- Inicializace – nastavením registru počátečního stavu na 1, ostatních na 0.

V obou případech se jedná o synchronní sekvenční obvody. Přechod do nového stavu je řízen hodinovým signálem.

Implementace – binární kód

- Aktuální stav je uložen v datovém registru
- Při každém taktu se do registru uloží nový kód stavu, který je funkcí stavu předchozího a kódu vstupního symbolu.



Implementace přechodové funkce

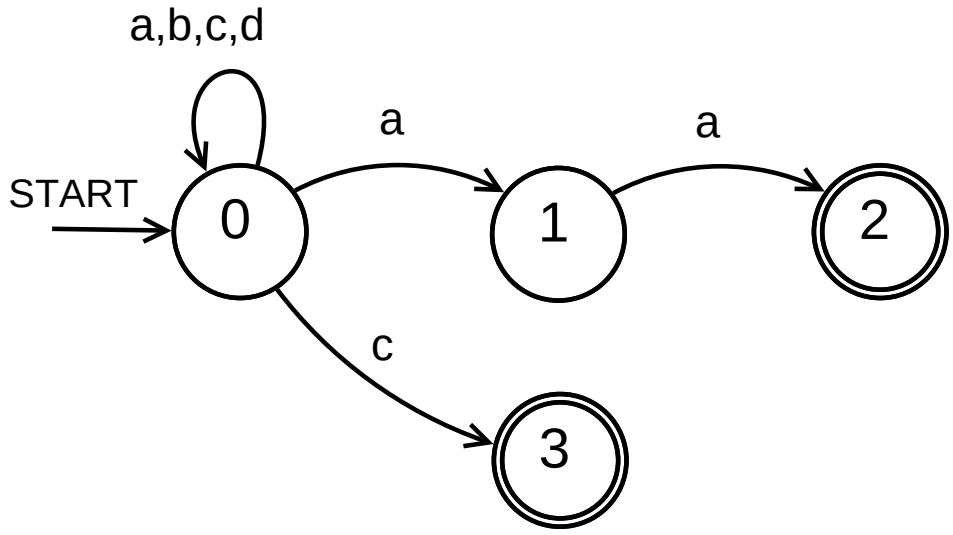
- Kombinační logikou.
  - Jednotlivé kódy následujícího stavu jsou určeny kombinační funkcí bitů kódu původního stavu a vstupního symbolu.
  - Koncový stav je indikován kombinační funkcí bitů kódu aktuálního stavu.
- Programovatelnou pamětí.
  - Kód následujícího stavu je uložen na paměťovém místě, jehož adresa je složena z kódu původního stavu a kódu vstupního symbolu.
  - Indikace koncového stavu může být součástí přechodové funkce. Hodnota této funkce je pak invariantní ke vstupu (pro všechny vstupy je v daném stavu stejná).

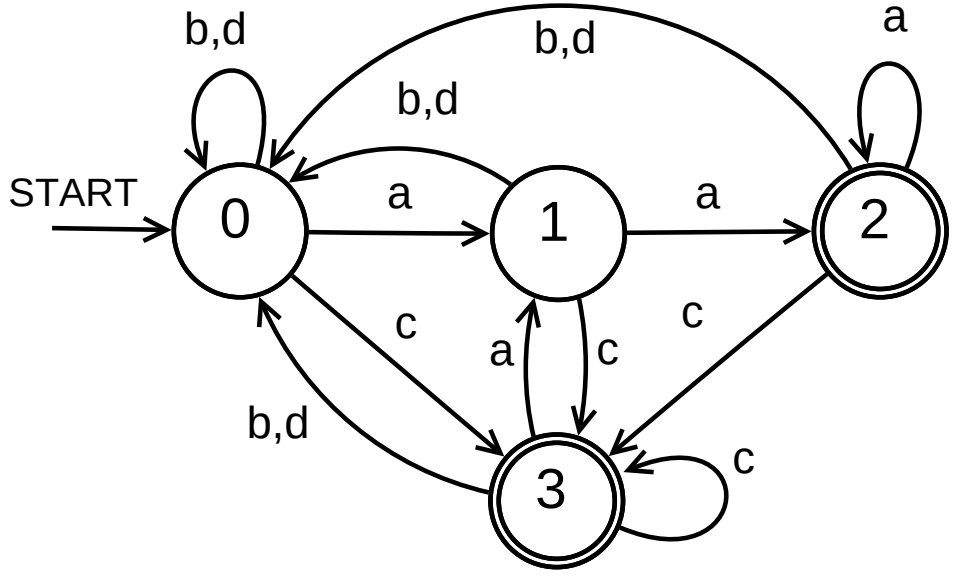
Příklad

Sestrojíme konečný automat nad abecedou $\Sigma = \{a, b, c, d\}$, který bude přijímat všechna slova, která končí na "aa"nebo "c".

Automat musí být deterministický a s úplně definovanou přechodovou funkcí.

Příklad – automat





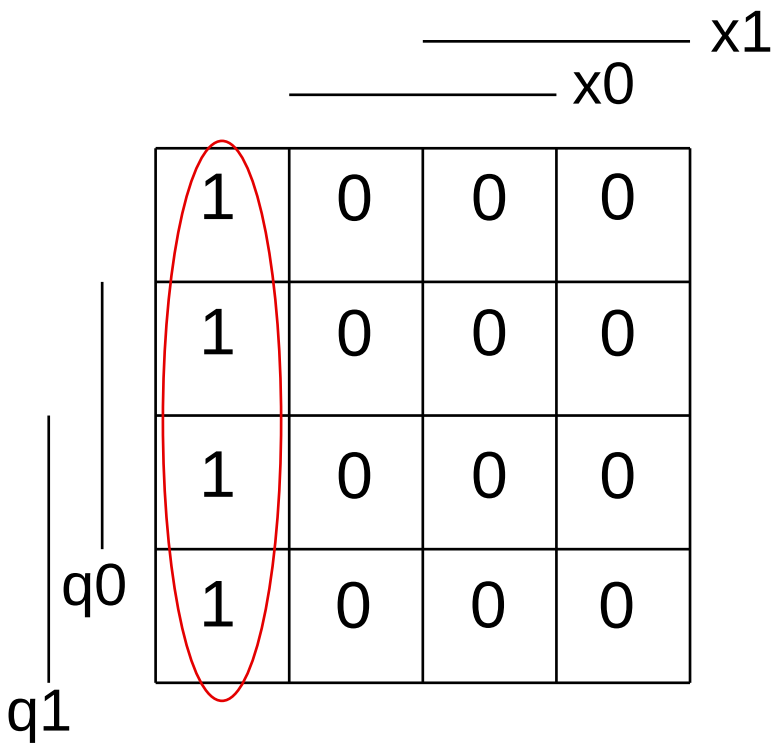
Příklad – tabulka přechodů

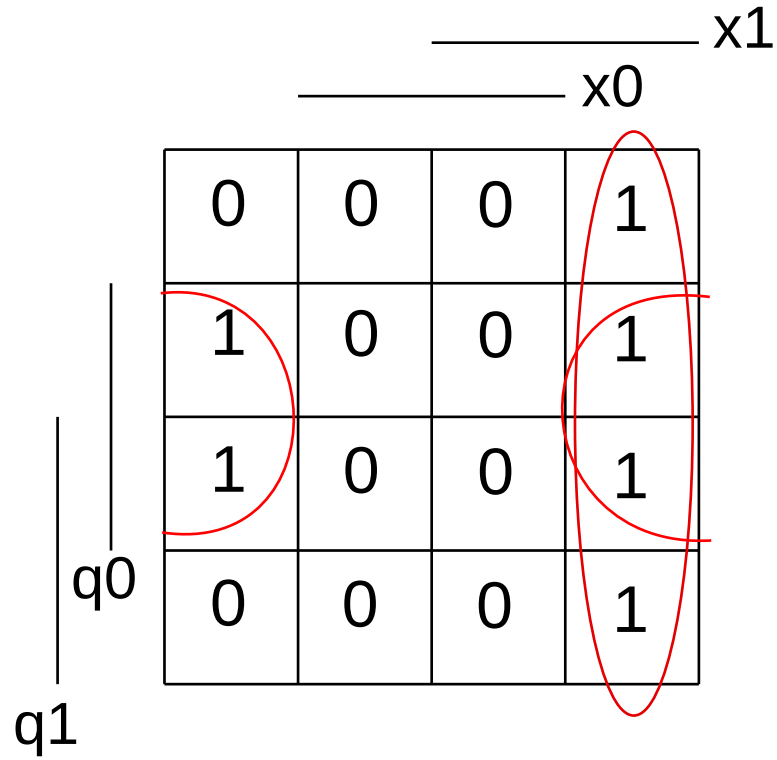
Q / X	a	b	c	d
$\rightarrow$ 0	1	0	3	0
1	2	0	3	0
$\leftarrow$ 2	2	0	3	0
$\leftarrow$ 3	1	0	3	0

Q / X	00	01	10	11
q_1q_0	$q_1'q_0'$	$q_1'q_0'$	$q_1'q_0'$	$q_1'q_0'$
$\rightarrow$ 00	01	00	10	00
01	11	00	10	00
$\leftarrow$ 11	11	00	10	00
$\leftarrow$ 10	01	00	10	00

Pro kódování stavů je v tomto případě výhodnější použít Grayova kódu. Vstupní abecedu kódujeme binárně.
V našem případě ušetříme několik bitů, ale byla by potřeba konverze na vstupu.

Implementace kombinační logikou

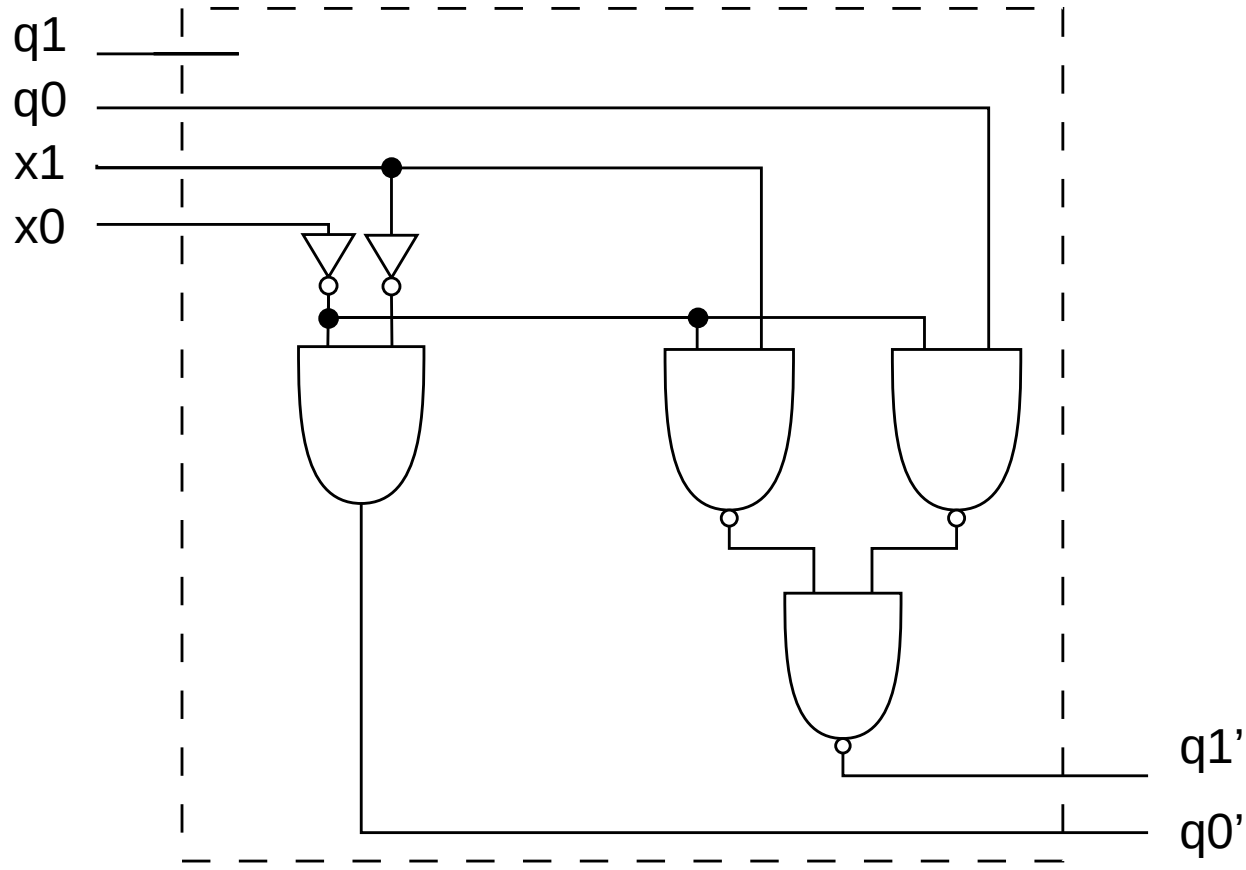




$$q'_0 = \overline{x_0} \cdot \overline{x_1}$$

$$q'_1 = \overline{x_0} \cdot x_1 + \overline{x_0} \cdot q_0$$

Implementace kombinační logikou



$$q'_0 = \overline{x_0} \cdot \overline{x_1} \qquad q'_1 = \overline{x_0} \cdot x_1 + \overline{x_0} \cdot q_0$$

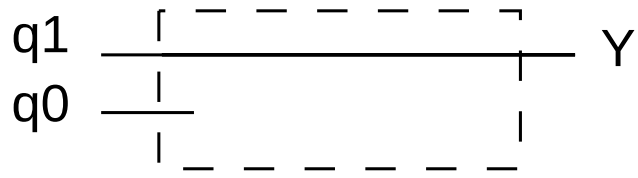
Koncové stavy

Koncové stavy indikujeme výstupní funkcí závislou pouze na aktuálním stavu (automat typu Moore).

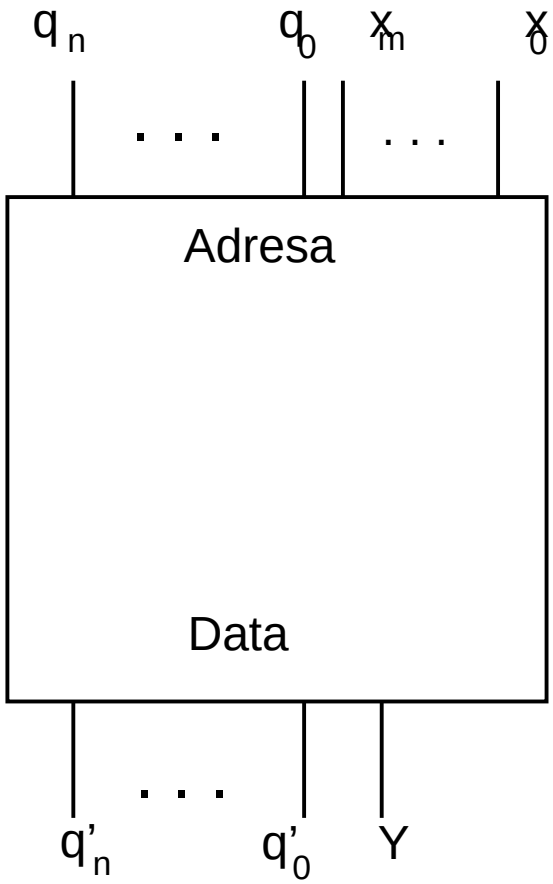
Q	Y
0	0
1	0
2	1
3	1

Q	Y
00	0
01	0
11	1
10	1

$$Y = q_1$$



Pomocí programovatelné paměti



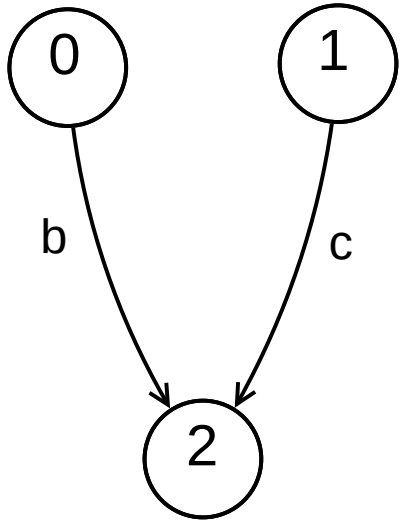
Adresa	Obsah
$q_1 q_0\ x_1 x_0$	$q'_1 q'_0 Y$
0 0 0 0	0 1 0
0 0 0 1	0 0 0
0 0 1 0	1 0 0
0 0 1 1	0 0 0
0 1 0 0	1 1 0
0 1 0 1	0 0 0
0 1 1 0	1 0 0
0 1 1 1	0 0 0

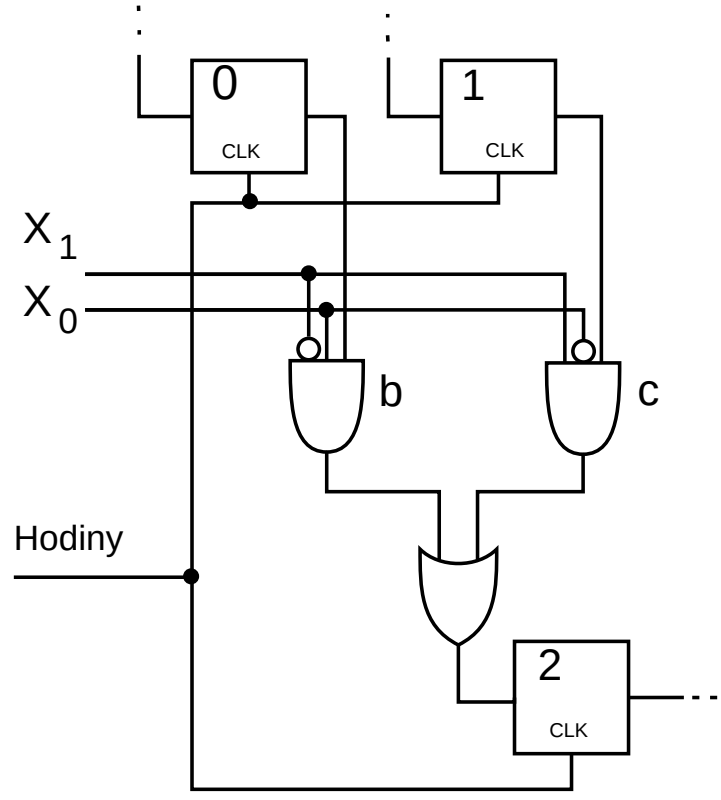
Adresa	Obsah
$q_1 q_0\ x_1 x_0$	$q'_1 q'_0 Y$
1 0 0 0	0 1 1
1 0 0 1	0 0 1
1 0 1 0	1 0 1
1 0 1 1	0 0 1
1 1 0 0	0 1 1
1 1 0 1	0 0 1
1 1 1 0	1 0 1
1 1 1 1	0 0 1

Indikaci koncového stavu Y začleníme do přechodové funkce (hodnota je invariantní ke vstupu).

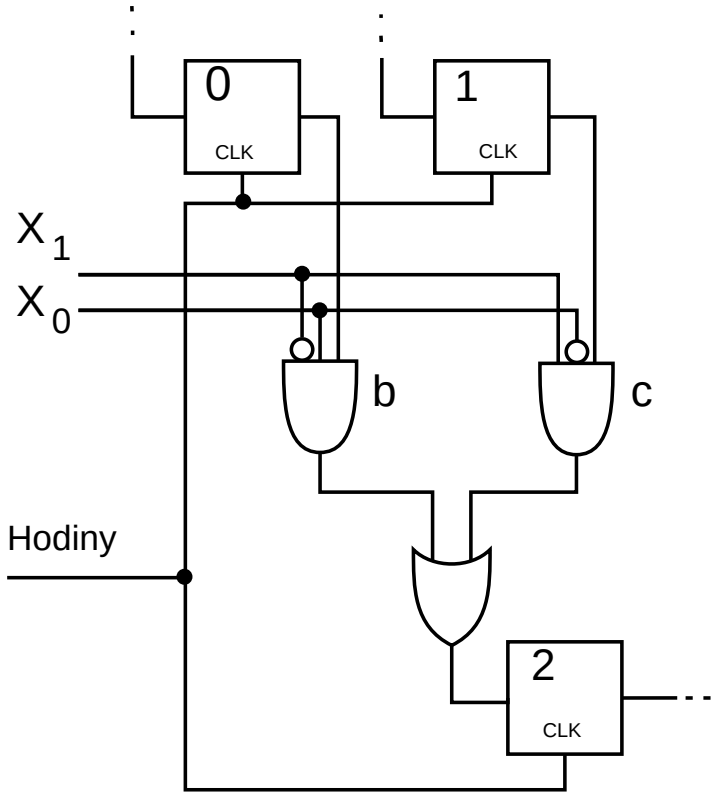
Fragment implementace – kód 1 z N

Každý stav je představován jedním 1bitovým registrem. Každý přechod je realizován vlastní logickou funkcí (vstupy b=01, c=10).





Fragment implementace – kód 1 z N



- řešíme každý přechod zvlášť (Oproti předchozímu)
- zřejmě více součástek
- Lze jednoduše simulovat nedeterminismus – aktivuje se více stavů najednou.
- Automat přijímá, pokud je některý koncový stav aktivní (velké OR hradlo na všech koncových stavech)