STU FIIT
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

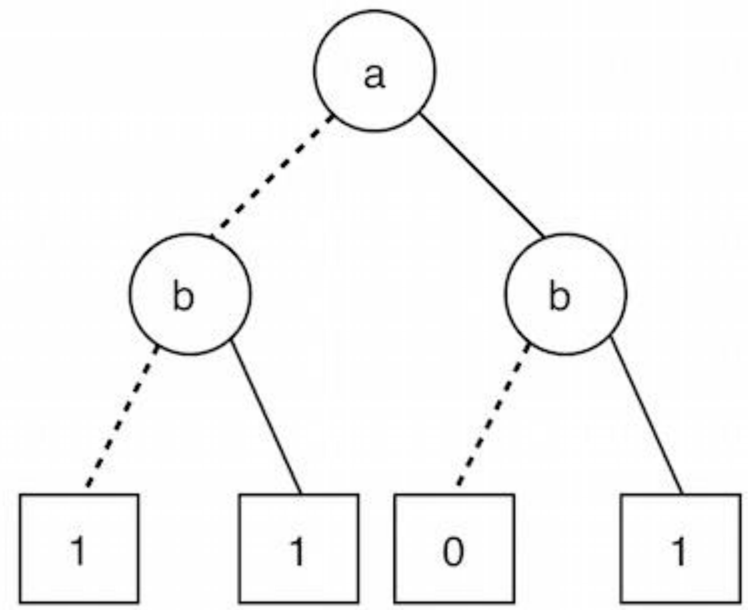
Binárne Rozhodovacie Diagramy 2

04. 04. 2023

letný semester 2022/2023

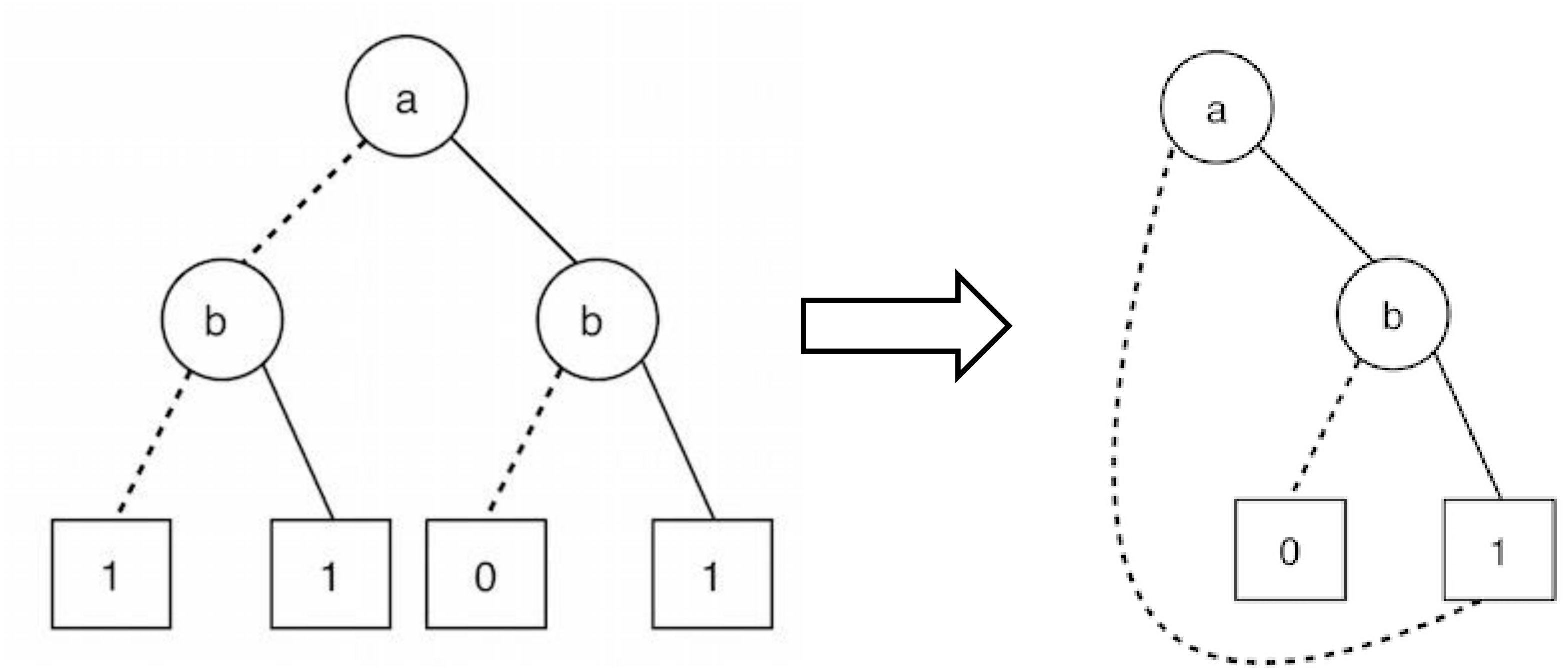
Problém Booleovských funkcií

- Pravdivostná tabuľka, vektor aj Karnaughova mapa sú reprezentácie, ktorých veľkosť je **2^N** pričom N je počet premenných Booleovskej funkcie
  - Exponenciálna zložitosť je problematická už pre N > 20
- **Rovnaký problém má aj BDD**
- Pridanie jednej premennej znamená pridanie ďalšej úrovne v BDD
- Jedná sa o úplný strom
- Neefektívne
- Ako to zlepšiť?
  - **Redukciou BDD**



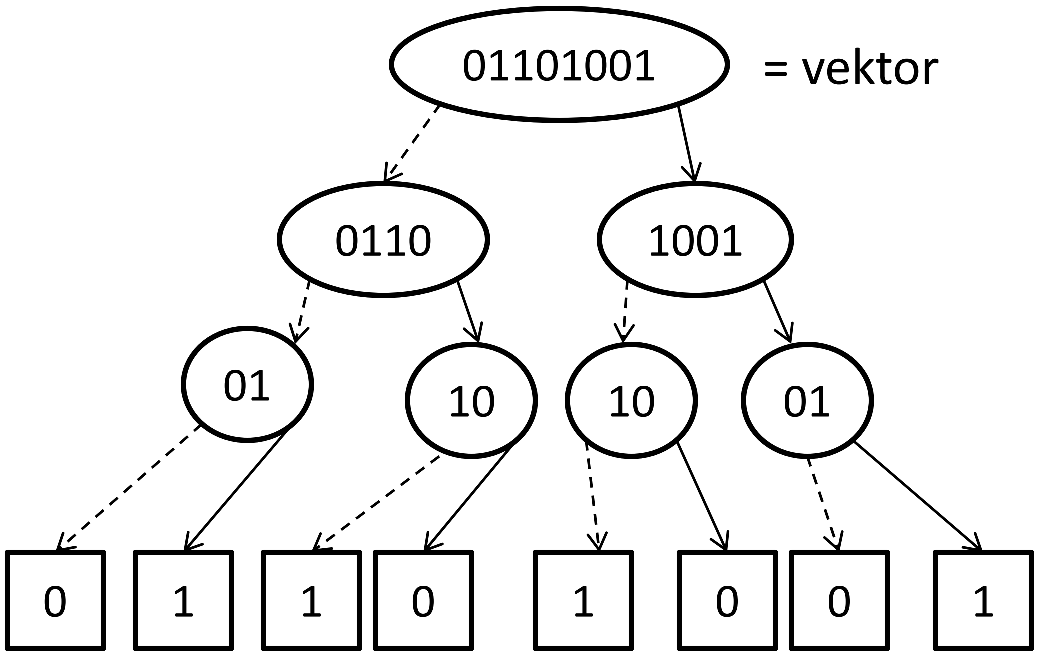
Redukcia BDD

- Počet uzlov by sme chceli minimalizovať
- Nie všetky uzly naozaj potrebujeme
- Odstránime redundantné (nadbytočné/zbytočné) uzly
- Redukcia exponenciálnej zložitosti až na lineárnu



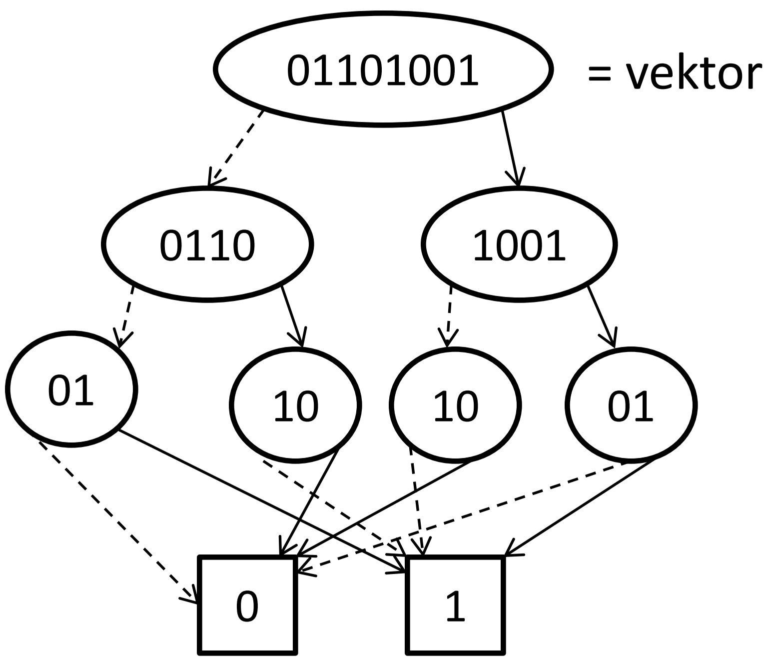
Zlúčenie koncových uzlov

- Koncový uzol (list) je vždy len 0 alebo 1 (aspoň v prípade jedno-výstupových Booleovských funkcií)
- Takže nám stačia len dva
- Zlúčime všetky jednotky dokopy a všetky nuly dokopy, upravíme pointre



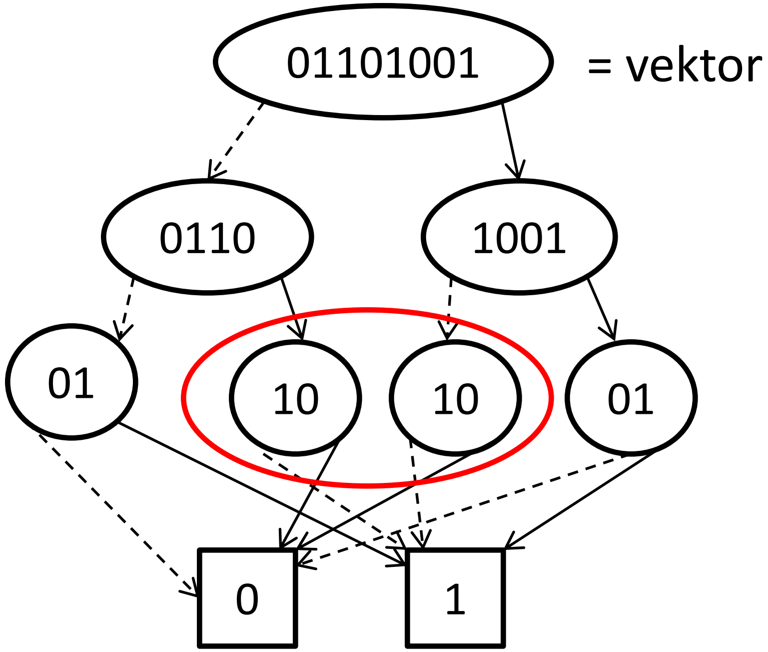
Zlúčenie koncových uzlov

- Koncový uzol (list) je vždy len 0 alebo 1 (aspoň v prípade jedno-výstupových Booleovských funkcií)
- Takže nám stačia len dva
- Zlúčime všetky jednotky dokopy a všetky nuly dokopy, upravíme pointre



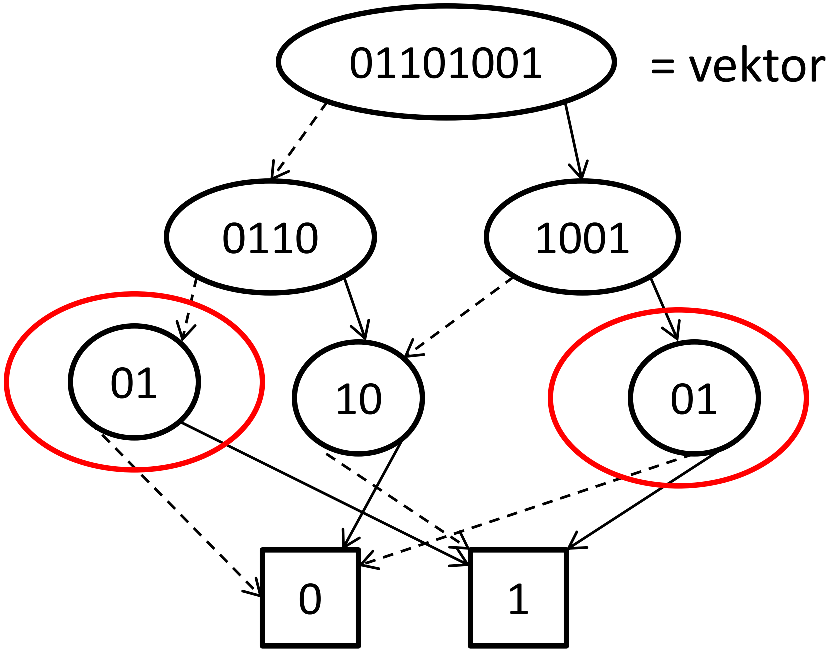
Zlúčenie vnútorných uzlov

- Ak je nejaký uzol nadbytočný, odstránime aj ten
- Ako zistíme, že uzol je nadbytočný?
  - Buď nemá žiadnu pridanú hodnotu
  - Alebo už taký istý uzol (s tou istou funkcionalitou) existuje



Zlúčenie vnútorných uzlov

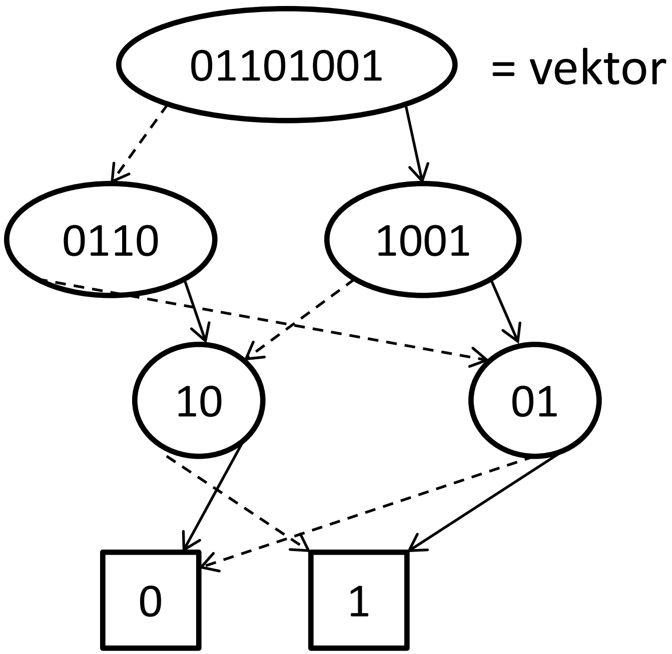
- Ak je nejaký uzol nadbytočný, odstránime aj ten
- Ako zistíme, že uzol je nadbytočný?
  - Buď nemá žiadnu pridanú hodnotu
  - Alebo už **taký istý uzol (s tou istou funkcionalitou)** existuje



Ešte nejaký?

Zlúčenie vnútorných uzlov

- Ak je nejaký uzol nadbytočný, odstránime aj ten
- Ako zistíme, že uzol je nadbytočný?
  - Buď nemá žiadnu pridanú hodnotu
  - Alebo už **taký istý uzol (s tou istou funkcionalitou)** existuje



Ešte nejaký?

Nie

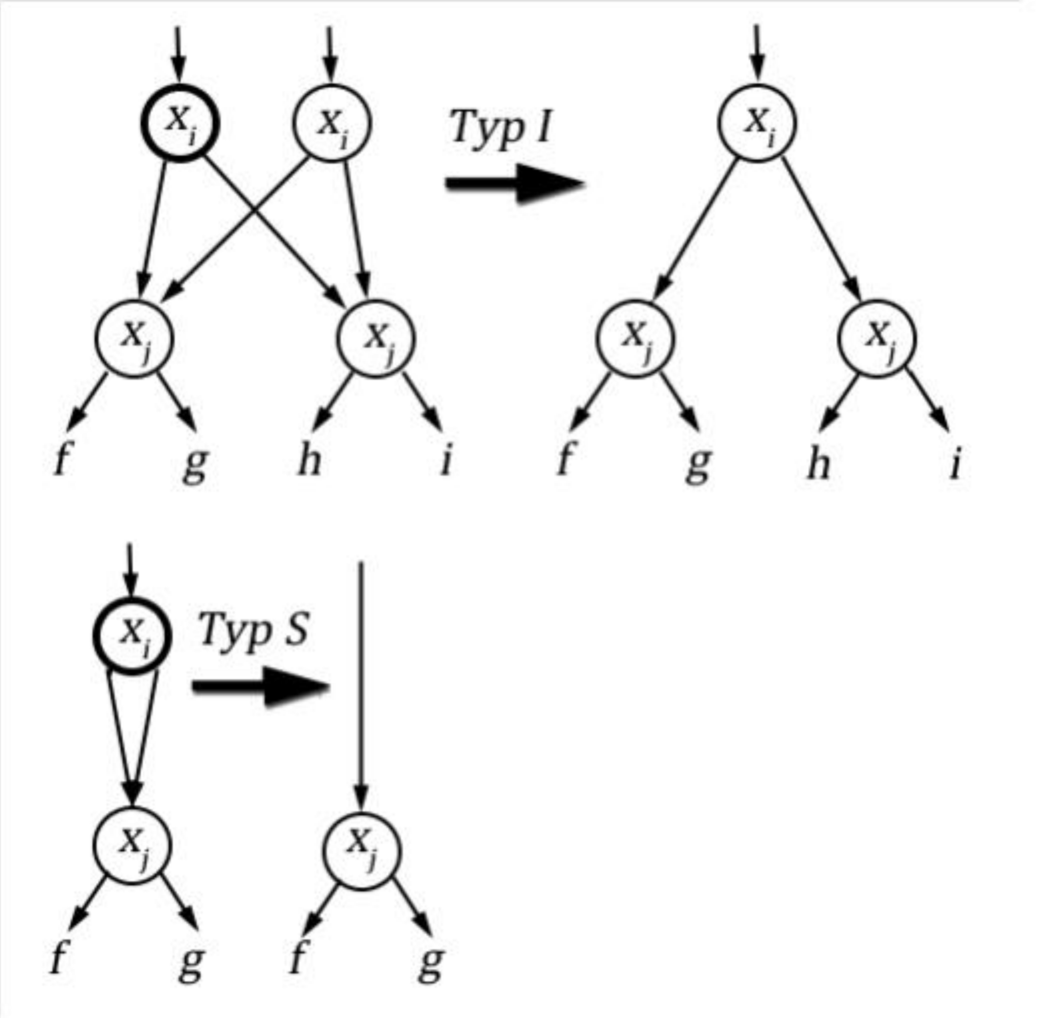
aspoň nie v tomto príklade...

Ako zistíme, že uzol nepotrebujeme?

- Buď nemá žiadnu pridanú hodnotu
  - Kedy nemá žiadnu pridanú hodnotu?
    - Keď ľavý_potomok == pravý_potomok
      - Porovnáme funkciu ľavého potomka s funkciou pravého potomka
      - Porovnáme pointre na potomky
- Alebo už taký istý uzol (s tou istou funkcionalitou) existuje
  - Treba porovnať všetky dvojice uzlov medzi sebou a zistiť, či sú rovnaké
  - Stačí porovnávať len uzly v rámci jednej úrovne (jednej premennej)
  - Kedy sú 2 uzly rovnaké?
    - Buď máme v uzle napísaný opis funkcie, ktorú uzol realizuje
      - Porovnáme priamo opisy oboch uzlov
    - Alebo zistíme, či ľavý_potomok(uzol_1) == ľavý_potomok(uzol_2) a zároveň pravý_potomok(uzol_1) == pravý_potomok(uzol_2)
- **Pozor!!!** Porovnávanie pointrov je použiteľné len vtedy, ak už nižšia úroveň bola celá zredukovaná (t.j. redukcia smerom zdola nahor)

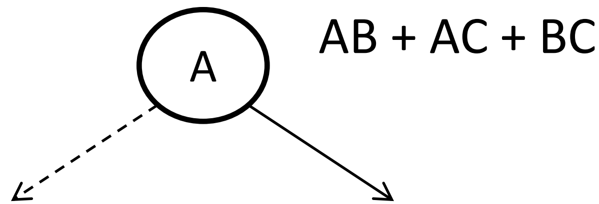
Základné pravidlá redukcie BDD

- Typ I – odstránenie nadbytočných uzlov porovnávaním dvojíc
- Typ S – odstránenie zbytočných uzlov porovnaním jeho potomkov



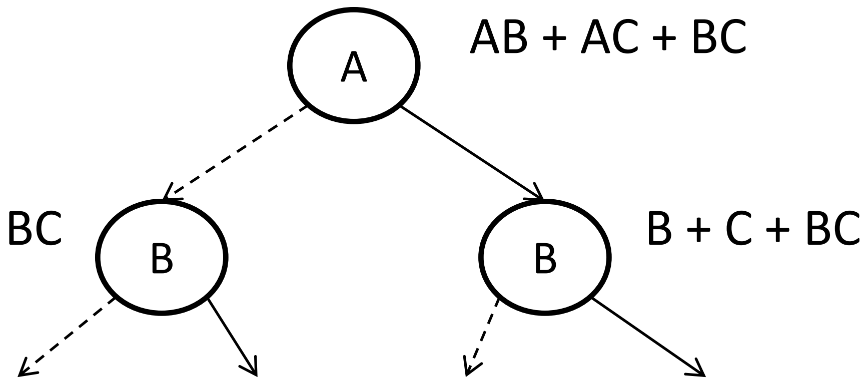
Príklad

- Y = AB + AC + BC



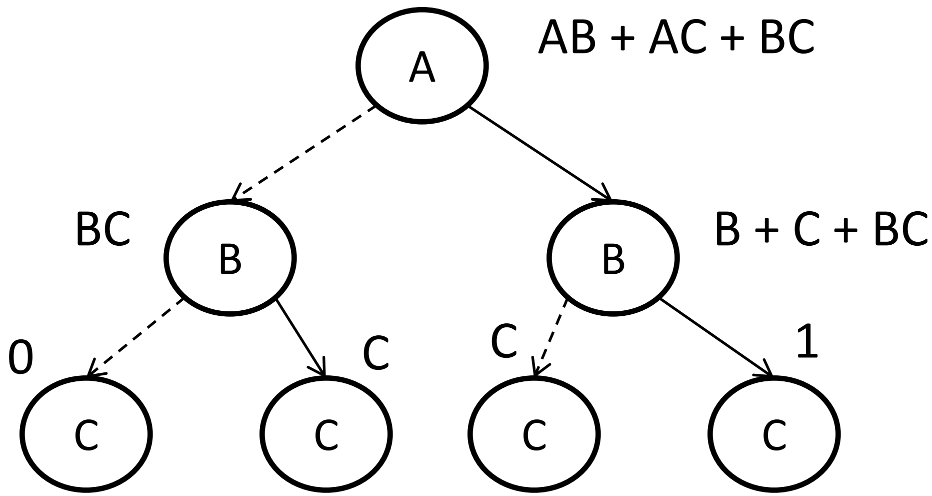
Príklad

- Y = AB + AC + BC



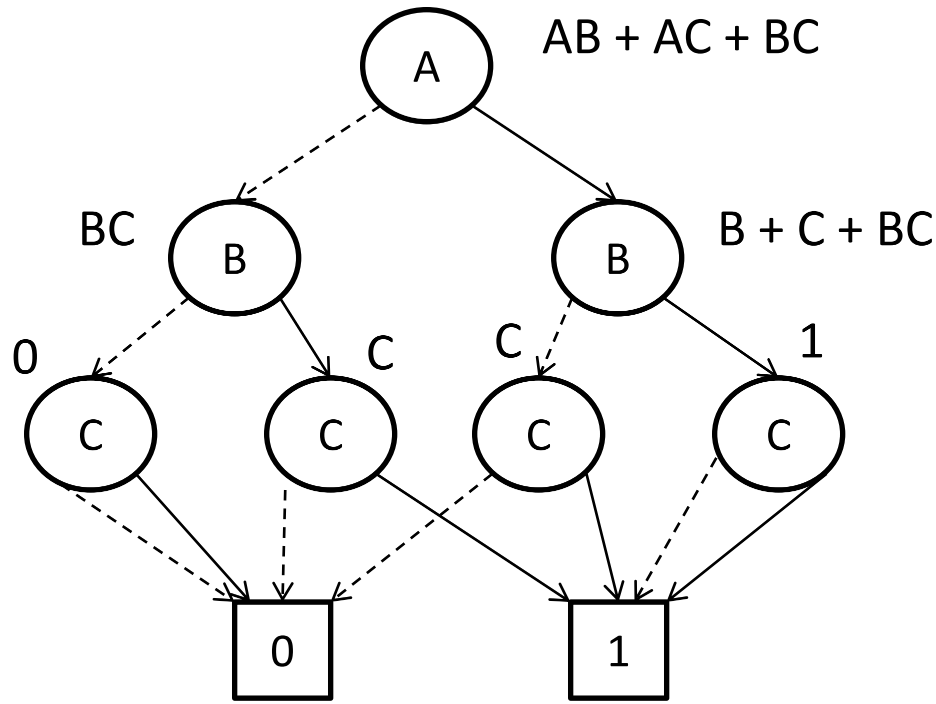
Príklad

- Y = AB + AC + BC



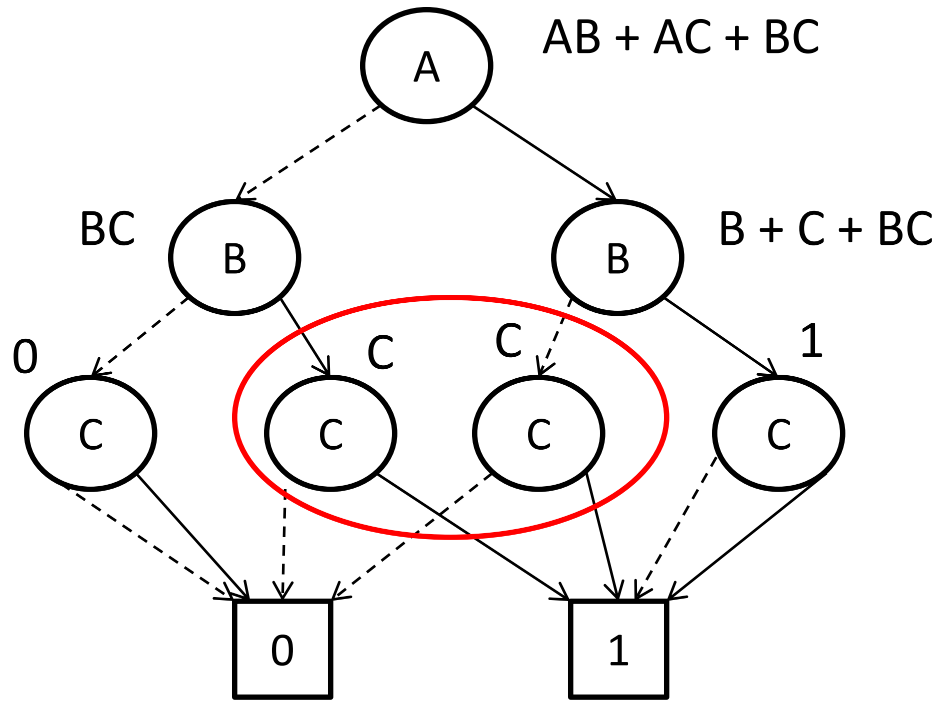
Príklad

- Y = AB + AC + BC



Príklad

- Y = AB + AC + BC

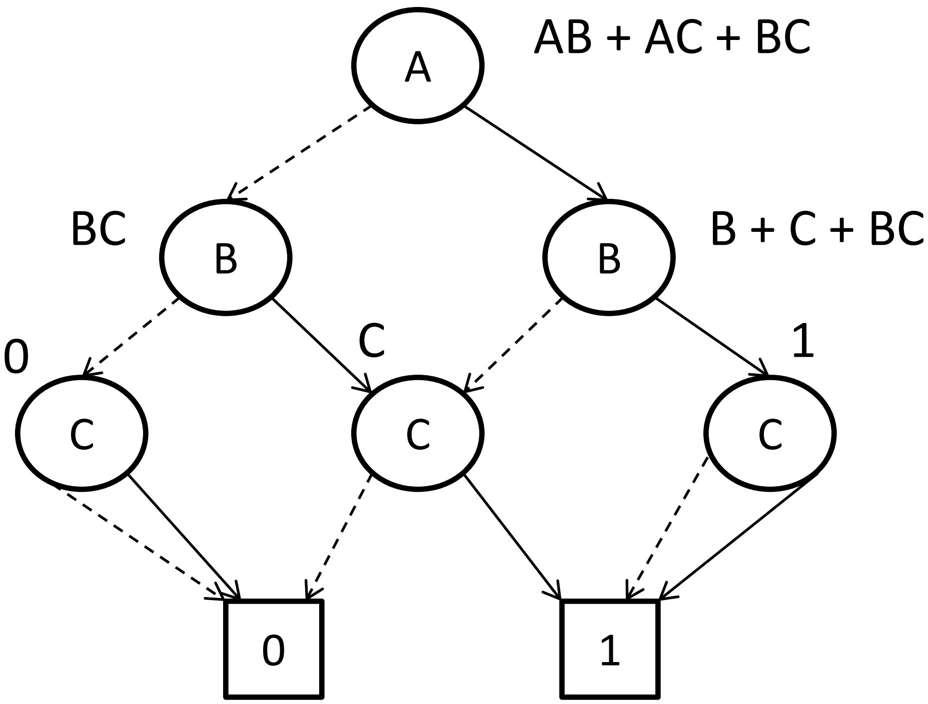


Redukcia typu I

- uzly reprezentujú rovnakú funkciu (C)
- ľavý potomok oboch uzlov je rovnaký a pravý potomok oboch uzlov je rovnaký

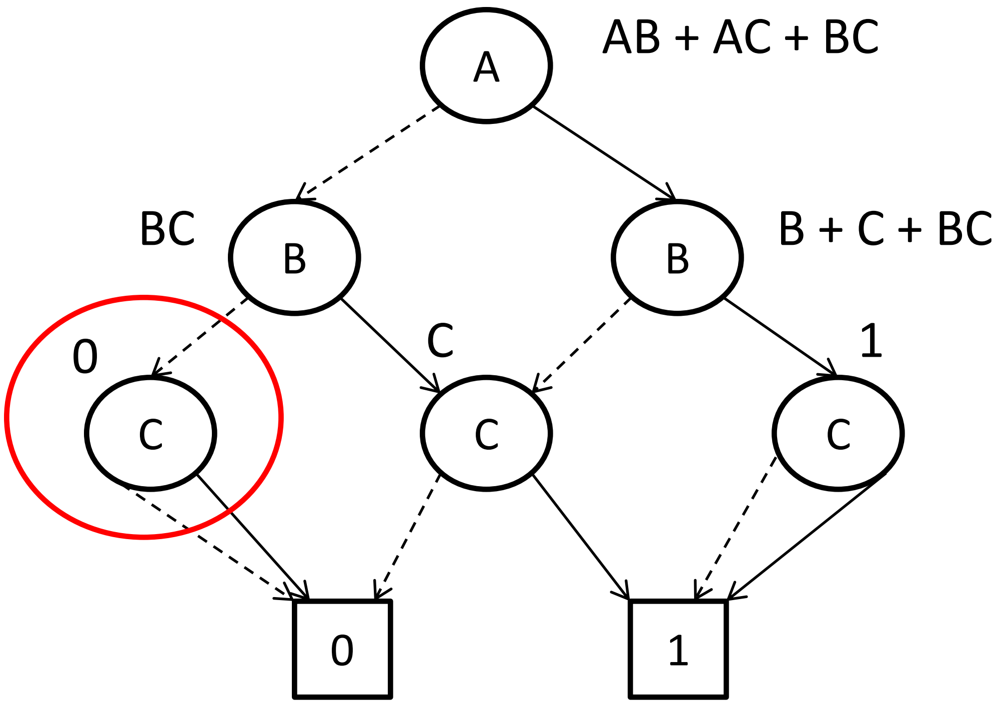
Príklad

- Y = AB + AC + BC



Príklad

- Y = AB + AC + BC

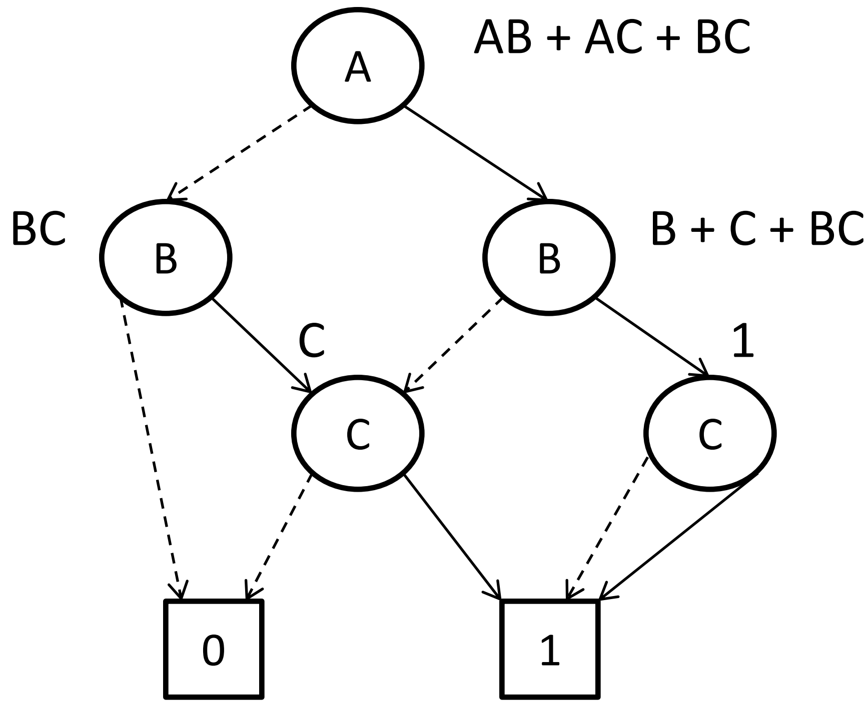


Redukcia typu S

- ľavý potomok je rovnaký ako pravý potomok (pointre sa rovnajú)
- špeciálny prípad – uzol reprezentuje konštantu (0)

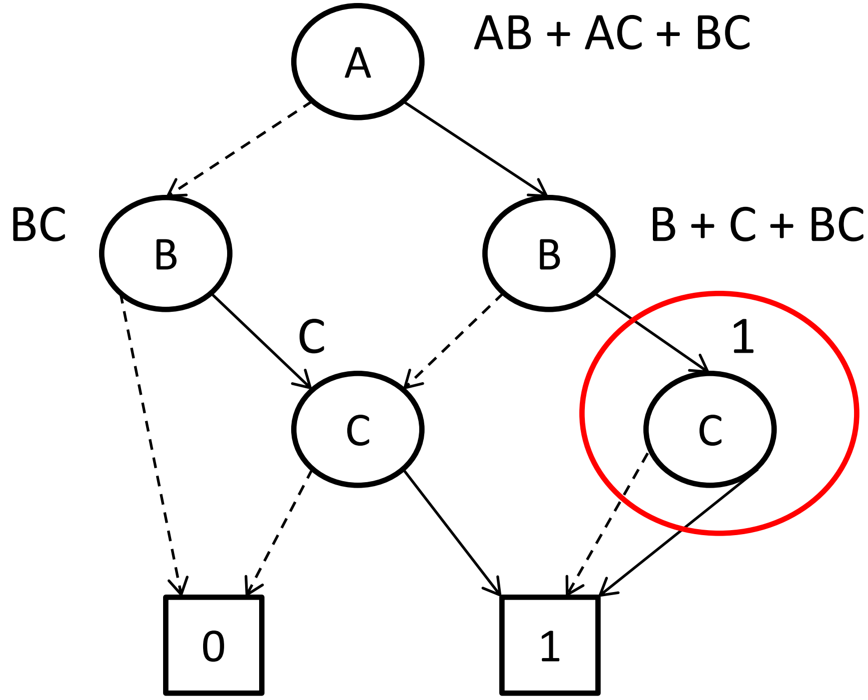
Príklad

- Y = AB + AC + BC



Príklad

- Y = AB + AC + BC

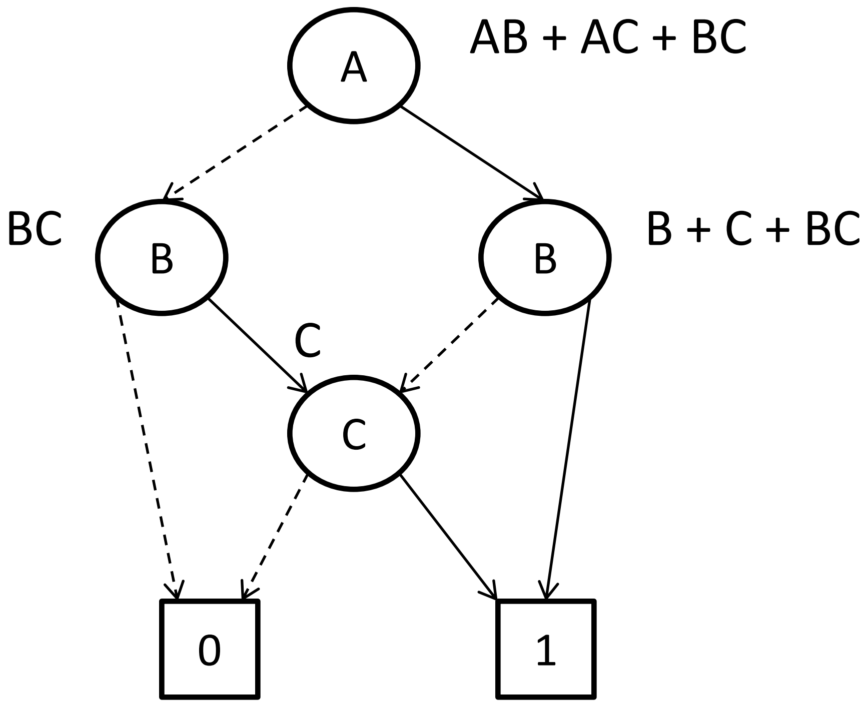


Redukcia typu S

- ľavý potomok je rovnaký ako pravý potomok (pointre sa rovnajú)
- špeciálny prípad – uzol reprezentuje konštantu (1)

Príklad

- Y = AB + AC + BC



Zložitosť redukcie typu I

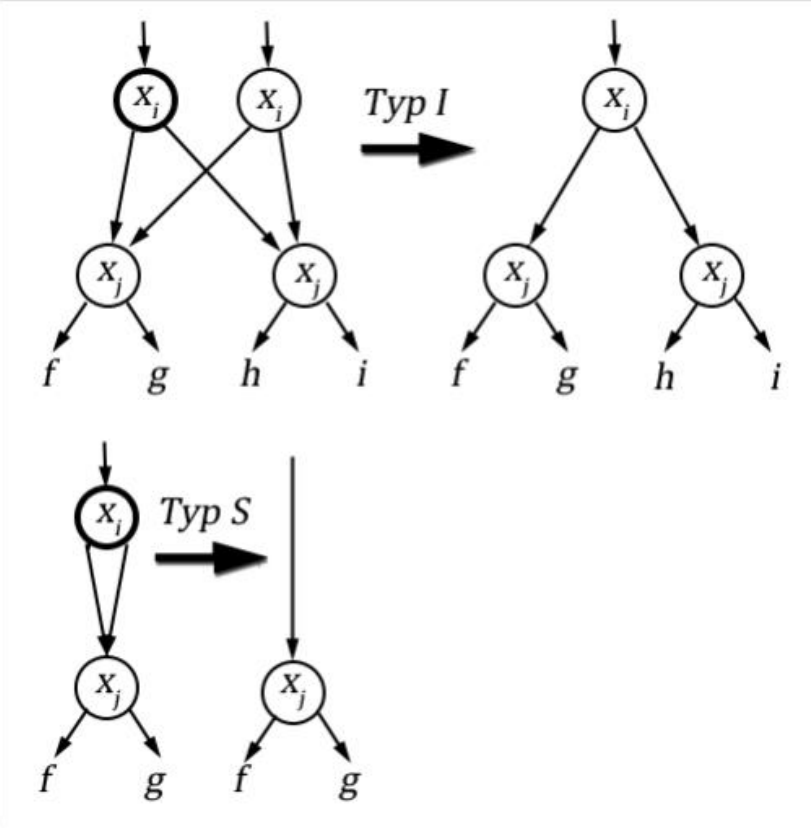
- Redukcia typu I vyžaduje porovnanie všetkých uzlov v rámci tej istej úrovne
  - T.j. všetky uzly riadené tou istou premennou
- Jedná sa teda o porovnanie všetkých dvojíc uzlov
- Koľko máme dvojíc?
  - Pre M uzlov v jednej úrovni je to M(M–1)/2
  - O(M^2) – kvadratická zložitosť
- Koľko môže byť M?
  - M ≤ 2^N, kde N je počet premenných B-funkcie
  - O(2^N)
- Celková zložitosť je tým pádom:
  - O((2^N)^2)

Zefektívnenie redukcie typu I

- Na porovnanie všetkých dvojíc uzlov si môžeme pomôcť hashovaním ☺
- Do hashovacej tabuľky s reťazením vložíme všetky uzly v rámci tej istej úrovne (tej istej premennej)
- Hashujeme funkciu, ktorú uzol reprezentuje
- Zhoda (rovnosť funkcií) dvoch rôznych uzlov zaručene spôsobí, že aj hash (index) bude rovnaký
- Porovnávame len uzly v rámci toho istého indexu v hash tabuľke
- Všetky uzly, ktoré majú iný hash (index), sú zaručene iné funkcie
  - Za predpokladu, že sme zakázali existenciu viacerých výrazov s rovnakým významom, napr. komutatívnosť, asociatívnosť a pod. (AB = BA)
  - V prípade binárnych vektorov je jednoznačnosť zaručená implicitne
- Zložitosť: namiesto O(M^2) je skôr O(M) ak je vhodne zvolená hash funkcia a vhodná veľkosť tabuľky

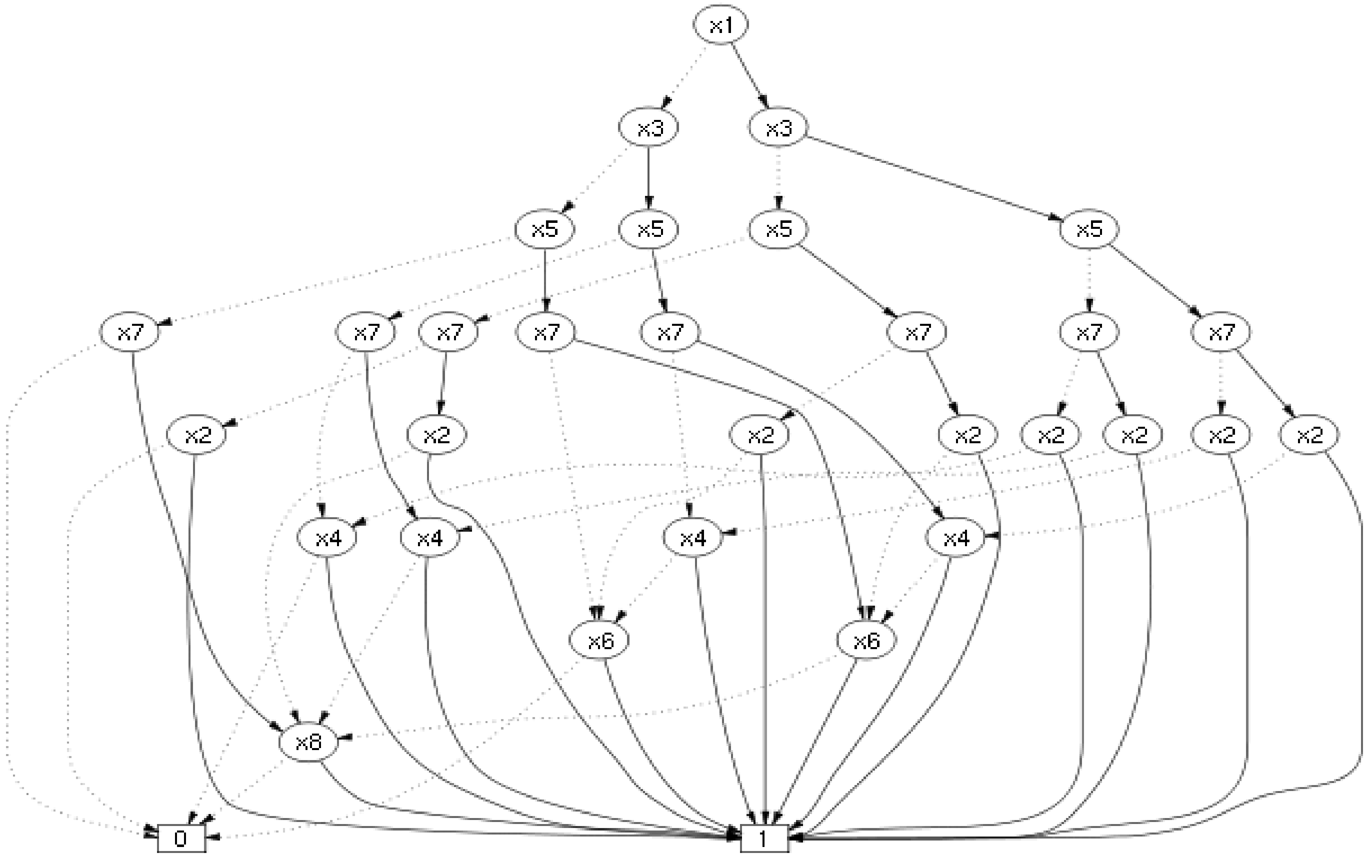
Kombinácia pravidiel I a S

- Existujú viaceré možnosti ako tieto pravidla kombinovať korektne a efektívne
- Napr. najprv vykonať typ I a potom sa vrátiť o 1 úroveň vyššie a vykonať typ S
- Alebo typ S vyhodnocovať porovnávaním funkcií a nie pointrov



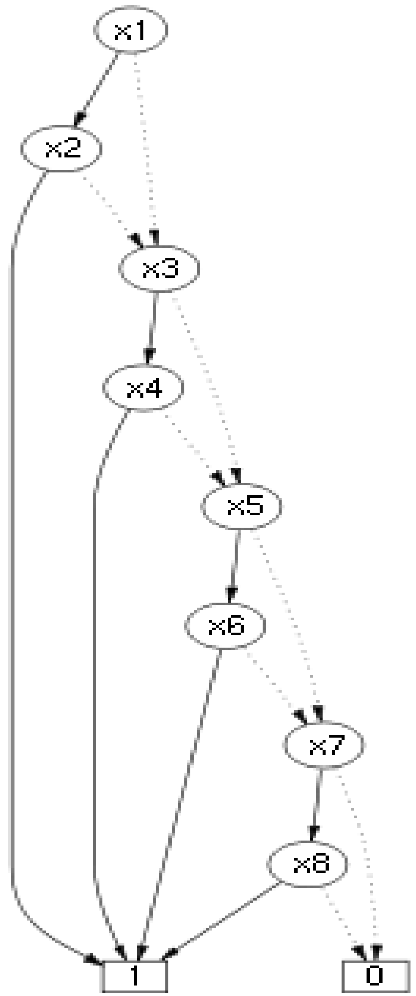
Redukovaný a zoradený = ROBDD

- Majme napr. funkciu Y = x1.x2 + x3.x4 + x5.x6 + x7.x8
- Zvolíme nejaké poradie premenných
- ROBDD môže vyzerať takto



Redukovaný a zoradený = ROBDD

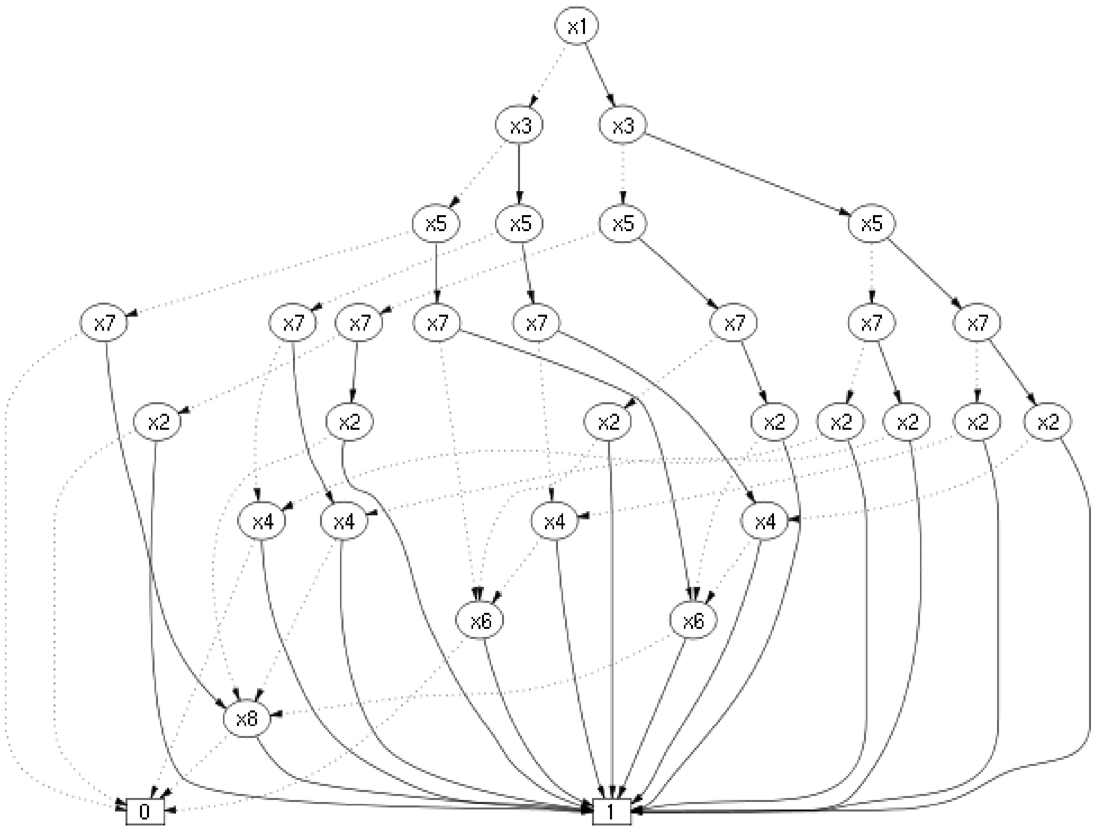
- Alebo aj takto ...

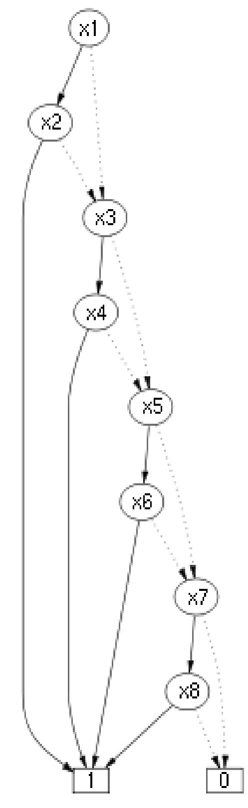


Ako je to možné?

Poradie premenných

- Poradie premenných dokáže výrazne ovplyvniť výslednú veľkosť ROBDD (t.j. počet uzlov)
- Prirodzene, chceme počet uzlov čo najmenší





Optimálne poradie premenných

- Použitím optimálneho poradia premenných dostaneme najmenší možný počet uzlov
- Ale ako zistíme optimálne poradie premenných?
- Väčšinou sa to robí systémom pokus-omyl
    - Vyskúšame nejaké poradie a pre toto poradie vytvoríme ROBDD
    - Zapamätáme si pre dané poradie počet uzlov
    - Zmeníme poradie premenných a znovu zostrojíme ROBDD
    - Porovnáme počet uzlov, ak sa zlepšil, pamätáme si toto poradie ako doteraz najlepšie
    - Opakujeme
        - Lenže koľko-krát?

Možnosti výberu poradia premenných

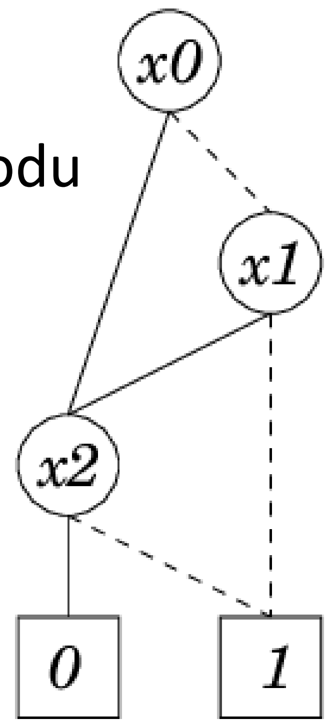
- Brute-force – vyskúšame všetky možné poradia premenných
  - Koľko ich je?
  - Máme N premenných, skúšame všetky permutácie, čiže N!
  - To je ale O(N!) – realistické tak pre N = 10 alebo menej
- Náhodne – náhodne vygenerujeme nejaké poradie (pomocou generátora náhodných čísel), X-krát opakujeme, X si zvolíme podľa toho, koľko času/úsilia sme ochotní tomu venovať
  - Problém opakovania toho istého poradia – plytvanie
- Lineárne – skúsime N možných poradí premenných, rotáciou premenných, napr. 12345, 23451, 34512, 45123, 51234
  - O(N) zložitosť, neopakujeme tie isté poradia
- Kvadratické – podobné ako lineárne, len s kvadratickou zložitosťou
- Pokročilejšie metódy (napr. hillclimbing, simulated annealing, genetic algorithms, ...)

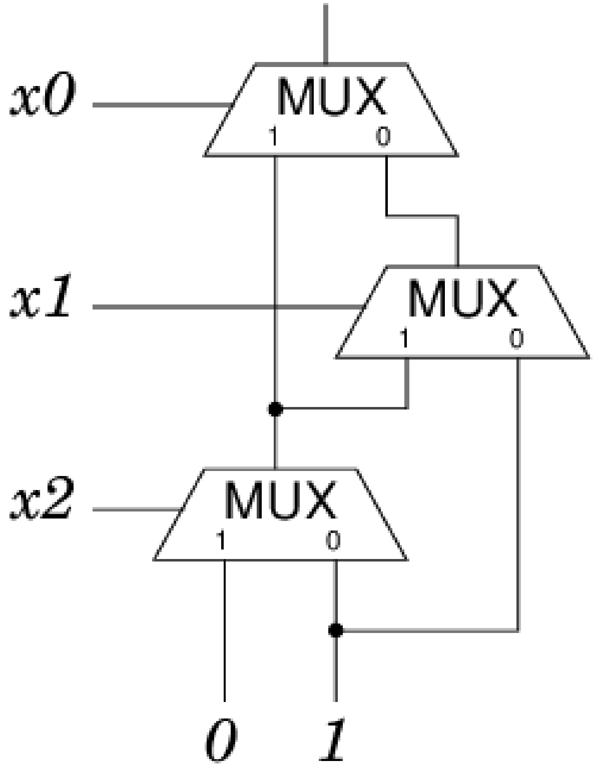
Multiplexorový strom

- Výsledný ROBDD môžeme použiť ako schému pre automatickú realizáciu (logickú syntézu) ľubovoľnej funkcie
- Výsledkom je kombinačný obvod (na čipe), ktorý dokáže realizovať zadanú Booleovsku funkciu
- Tento obvod sa nazýva multiplexorový strom
  - Strom multiplexorov (jednoduchých súčiastok)
- Mapovanie je 1:1
  - Každý uzol ROBDD sa transformuje na jeden 2-kanálový MUX
  - Hrany medzi uzlami budú vodiče spájajúce MUX-y (rovnakým zapojením)
- Riadiace vstupy do MUX-u sú jednotlivé premenné Booleovskej funkcie

Multiplexorový strom

- Čím menej uzlov má BDD, tým menej multiplexorov potrebujeme na realizáciu obvodu
  - Menšia plocha čipu, lacnejší, spoľahlivejší, s nižšou spotrebou energie
- Koreň BDD → výstup obvodu
- Uzol → MUX
- Hrana → vodič





Ďakujem za pozornosť